Online Handwriting Recognition for Ethiopic Characters

By

ABNET SHIMELES

A THESIS SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES OF ADDIS ABABA UNIVERSITY IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

June, 2005

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF INFORMATICS

DEPARTMENT OF COMPUTER SCIENCE

Online Handwriting Recognition for Ethiopic Characters

By

Abnet Shimeles

Name and Signature of Members of the Examining Board

1. Dr. Solomon Atnafu, Advisor _____
2. _____
3. _____

Acknowledgement

First of all, I thank God for being with me all the time, not only during this research, but also in my whole life. It is all His kindness that brings this happiness to my life.

What would have happened to my thesis paper without my advisor's valuable ideas and family-like support? I really thank Dr. Solomon Atnafu, my advisor, for his patience, friendly treatment, valuable ideas, and supportive advices. I, not only gained academic knowledge, but also learned humanity from him. I say, from the bottom of my heart, thank you, while I at the same time feeling that the words are not still expressing my feelings.

'Netsi', starting from pushing me to join the M.Sc. program, he was always besides me in all my activities in the last two years. I hope he will be with me forever. I know that you want me to grow more. I promise, I will!

My parents all assisted me to stand here. Without their support full of love, I would have never been 'Me'. Mom and Dad were worrying about me while I was sitting in front of the PC at home for long time. My sisters and brothers all follow every of my progress and sometimes they joked at me which make me laugh. Thank you so much for your prayer, concern and support.

My colleagues in Unity University College helped me a lot by sharing my burdens so that I can finish this research. I don't have words to thank them.

Table of Contents

List Of Figures

List Of Tables

Online Handwriting Recognition for Ethiopic Characters

MSc. Thesis by: Abnet Shimeles

Advisor: Dr. Solomon Atnafu

ABSTRACT

A new computing scheme, pen computing, which includes mobile devices and applications in which electronic pen along with pen sensitive writing pad is used as the main input tool has been emerging. To implement pen-computing applications, online handwriting recognition system should be used. Online handwriting recognition engines have been developed for various character sets. Despite that, no attempt has ever been made to build an online handwriting recognition engine for Ethiopic character set. Pen-based inputting incorporated with online handwriting recognition feature allows people to write texts and enter input data in their own natural way of handwriting on an electronic pad.

This thesis then is the first attempt to develop an online handwriting character recognition engine for Ethiopic characters. The pen-based devices are evidently unusual in Ethiopia and one reason for that is the absence of localized applications. Bringing an online handwriting recognition engine for Ethiopic character set to such devices would play an important role in making these devices available and usable for the Ethiopian society.

In this study, a model for Ethiopic online handwriting character recognition is proposed and a writer-dependent online handwriting character recognition engine for the 33+1 basic Ethiopic characters is designed. The designed engine integrates five modules: the data collection and preparation module, the preprocessing module, the feature extraction module, the training module and the classification module. Data collection is done with the aid of digitizer software named Neuroscript MovAlyzer, which samples data points along the trajectory of an input device (electronic pen or mouse) while the character is drawn. Various algorithms are designed for the preprocessing activities. In the feature extraction module, a new online handwriting data representation scheme that makes use of the X and Y coordinate observation code sequences is proposed. A training algorithm and most importantly a three-layered recognizer is designed. We are able to show that a reasonably good accuracy is obtained by implementing the proposed algorithms. On the average, a recognition accuracy of up to 99.4% is achieved for the sampled two writers. Recognition accuracy 93.4%, 99%, 99.8% are also obtained for each of the layers of the recognizer respectively.

Keywords: Online handwriting recognition, Online handwriting recognition for Ethiopic Characters, algorithms for Ethiopic online handwriting recognition, Model for Ethiopic character online handwriting recognition.

CHAPTER ONE

INTRODUCTION

1.1. Background of the Study

The development of technology is changing how people live and work. Situations, which were unthinkable, are becoming possible and practical. The powerful computers available today capable of processing a large amount of data with a great speed were scientific fictions before a century. The graceful giant machines that appeared in the 1980's had been a miracle. Out of imagination, computers became more and more powerful and at the same time more and more compact.

Today, computers have become as compact as handheld devices that fit on human hands. Not only their small size, but also their computing power to implement various applications makes them very attractive. Historically, the whole thing was started when a company called Palm produced PDAs (Personal Digital Assistant), whose main function was to replace personal organizers[30]. Small applications such as calendar, and address book were run on the PDAs. These devices were called *palmtop* since they were one-handed and fit in one's palm. More companies also involved in the production of such devices. On the other hand, Microsoft produced minimized laptop-style computers with screen of size 20cm wide and only 8 cm high and with an attached keyboard. These computers used a version of Windows, named Windows CE and called *handheld* rather than palmtops to indicate that they don't fit in palm[30]. *PocketPcs* were the other devices using the modified version of WindowsCE which are sometimes called *handheld*. This time, these terms that emerged at different times have been used interchangeably. In general, handheld computers are defined to be any small devices that provides computing and information storage and retrieval and that can be easily carried and used[30].

Though these devices turn out to be common in an increasing number of countries around the world, they are currently almost ignored in Ethiopia. Even people with full knowledge of operating personal computers do not even know their existence or are not using them due to various reasons. Localizing the applications of the devices could play an important

role in making them prevalent to the Ethiopian society. We have envisaged that devices like handheld computers and smart phones that incorporate the wireless technology would be in widespread usage in the future as the trend of people who use mobile phones is highly rising. If this happens, then it would be compulsory to have localized applications that run on such devices for the purpose of exploiting the opportunity of mobile computing in this country.

When these devices are used, data and commands are needed to be entered while users are moving. The keyboard as well as the mouse that are common in personal computers for this purpose do not go with the mobility of these devices. Thus, pen-like devices commonly called stylus will replace them. In relation to this situation, pen computing has been an emerging trend of computing.

Pen computing includes computers and applications in which electronic pen is the main input device[26]. This includes pen-based mobile computing devices such as Personal Digital Assistants (PDA), and other Palmtop devices. These sorts of devices are becoming affordable, and applicable in variety of domains. Consequently, nowadays they are becoming popular and a very large number of customers are carrying and using them[27]. As it is stated earlier, the distinctive characteristic of handheld computing devices is the use of electronic pen (or stylus)[27] to input data and commands to their system. Pen-based inputting incorporated with online handwriting recognition feature allows people to write in a natural way to input data, and provide a pen-paper like interface[8]. These kinds of systems are very useful in terms of mobility, convenience, cost, capability of accessing information at anytime and anywhere. Therefore, the above situations and many more other reasons initiate many researchers to put their effort towards developing systems that could mimic the pen-paper interface for various specific languages.

It has already been asserted that the pen-paper interface could possibly be available if handwriting recognition is implemented. Handwriting recognition is the task of transforming a language represented in its spatial form graphical marks into symbolic representation[3]. Handwriting recognition systems are broadly divided into two: namely offline and online handwriting recognition systems[3,4,24]. In offline handwriting recognition systems, the whole data will be collected and provided for the recognizer as a

bitmap. On the other hand online handwriting recognition systems run and receive the data as the user writes and they are expected to process the data and recognize in a real time[3,24]. Both online and offline handwriting recognition systems for Western languages have matured right now. These kinds of systems are also developed for other non-Latin languages such as Chinese, Japanese, Thai and Arabic[28,13,29,12]. Handwriting recognitions systems are language specific. This calls the need for localization of such system to make it available for the Ethiopian society.

Designing and developing handwriting recognition systems is far from easy[4]. In spite of the fact that a recognition system with 100% recognition rate is not still attained, a better recognition rate has been exhibited from time to time by the use of various approaches. An approach selection is another non-trivial task for which extensive study of the nature of the language and other parameters should be investigated

1.2. Problem Statement

The Ethiopic character set is unique to Ethiopia and Eritrea and it is used to write different languages including Amharic in Ethiopia. This character set contains more than 300 characters, of which the 231+7 are alphabetic letters. The remaining characters include the Ethiopian numerals, punctuation symbols and some infrequently used special characters. The Ethiopic character set, specifically the alphabetic letters set, is known by its systematic arrangement. The letters are arranged in table form with 7 columns and 33+1 rows. All characters in the first column are ***basic*** (core) characters and the remaining columns consist of the ***non-basic*** characters. The entire arrangement and this classification are based on the fact that the shape of the non-basic characters in each row is generally derived from the basic character in the same row.

To the best of our knowledge, there is no research that has ever been conducted on ***online*** handwriting recognition for Ethiopic characters. We have investigated that there are three researches around ***offline*** handwriting recognition of Ethiopic characters [6,7,8]. Thus, this area has not been even touched and this is a pioneer work, which shall open the way for other researchers to involve in the area. This thesis then addresses the problem of online handwriting recognition for Ethiopic characters.

1.3. Objectives

The general and specific objectives of this research work are outlined below:

General Objective:

The main objective of the study is to explore, analyze and design an appropriate algorithm for the purpose of online Ethiopic handwriting recognition for Ethiopic characters on a PDA Environment.

Specific Objectives:

To meet the mentioned general objective, the following specific objectives are accomplished in this research work.

- Assess the different techniques for preprocessing, segmentation, feature extraction, training and classification for online handwriting recognition task.
- Study the techniques in relation to their appropriateness for Ethiopic online handwriting recognition and propose a technique.
- Design and implement algorithms for handwriting pattern representation scheme, training, and classifications.
- Conduct experiments to evaluate the proposed algorithms for their accuracy.

1.4. Justification of the Study

Writing is a natural way of putting information. The emergence of very sophisticated digital computers with facilitated input methods do not cover up the importance of the handwriting activity. Particularly, in places in which taking short notes is necessary, keyboards or keypads are illogical. For most people, a pen and a notebook are much better[3]. On the other hand, learning and adapting to the unnatural way of feeding data to computers had been a trouble for a significant number of people (this is even worse in Ethiopia as the implementation of the available software is based on English language). To see the idea in a different view, even computer literates are not perfectly happy by the method employed to enter data in desktop computers, which doesn't generally allow mobility. However, devices that offer a pen-paper based interface mostly are very small in size that can fit to a palm.

In summary, handheld devices particularly PDAs have the following useful features:

- Convenience

- Portability
- Affordability
- Easy Information access any time/anywhere
- Wireless communication

Some of the application domains of PDAs are:

- In Schools for students and teachers.
- In hospitals for doctors
- Field data collection
- For short note taking in meetings

Currently, in Ethiopia, handheld devices for instance PDAs are not common. One of the technical reasons is that they are not still suited for local applications. The goal of this research is to localize the online handwriting recognition system feature of handheld devices so that Ethiopians can benefit from this technology.

1.5. Scope and Limitations of the Study

First attempts are always demanding. On top of this, handwriting recognition researches are generally challenging mainly due to the variability in handwriting styles. Additionally, standardized online handwriting recognition data set is not available at all. The number of characters in the Ethiopic character set is also large relative to the Latin characters for which researches have been conducted for decades to come up with improved approaches that can bring about a better recognition accuracy. Thus, we put some constraints on the recognition engine after examining these situations. Accordingly, the scope and limitations of this research are that:

- Only character-level recognition is considered.
- The online handwriting recognition system developed in this work is writer-dependent.
- Only the 33+1 basic Ethiopic characters are considered.

1.6. Methods of the Study

Online handwriting recognition engine is not a one-sep process. Thus, different techniques are used in each stage as described below:

1.6.1. Literature Review

Research papers on online handwriting recognition systems for Latin, Chinese, Japanese and Arabic character sets have been reviewed to see the state-of-the-art status and to identify the various approaches used for this problem. Moreover, the existing offline handwriting research papers have also been reviewed in order to get ideas on how they treat the Ethiopic characters.

1.6.2. Design of the Online Handwriting Recognition System

Data Collection Technique

The online handwriting data of the 33+1 basic Ethiopic characters have been collected by using the MovAlyzer digitizer software that samples data points when a character is drawn by the mouse. This data serves for training and testing the recognition engine.

Preprocessing Algorithms

Various preprocessing activities involve in any handwriting recognition system. The recognition system developed in this project employ algorithms for noise elimination, size normalization, filtering, and resampling.

Pattern Representation Method

The online handwriting data as it cannot be directly used for recognition, an intermediary pattern representation method can ease the recognition of the unknown character in many ways. For this, a new pattern recognition scheme which attempts to represent patterns in terms of X and Y observation sequences was designed.

Training and Testing

Training is a crucial action before any recognition system becomes capable of recognizing entities, which are supposed to. Thus, a training algorithm that goes with the pattern representation scheme has been designed. Experiments have also been conducted to test the system for its accuracy.

1.6.3. Development of the Ethiopic Online Handwriting Recognition Engine

The language used to implement the various algorithms is C/C++. Text files are used to store the various data processed by the system. The original handwriting data is provided by the digitizer software in the form of text files. The preprocessing module takes these text files as an input and produces other text files that store the preprocessed data. The feature extraction module, in turn, receives the preprocessed data and produces observation codes sequences and put them in individual text files during training and classification. The recognizer also takes the observation code sequences to predict what character is represented by that specific input. To draw characters from a given data, the graphics tool of the Turbo C++ compiler has been used.

1.6.4. Organization of the Thesis

This document contains a total of seven chapters. The second chapter presents general concepts about online handwriting recognition. In chapter three, related works are reviewed. The fourth chapter presents the analysis done on the shapes of Ethiopic characters for the purpose of designing the recognition engine. Chapter five is the chapter in which we present the design of the online handwriting recognition system for the basic Ethiopic characters. All algorithms designed for the various activities that are involved in the recognition system are detailed in this chapter. Chapter six presents the experimental results by incorporating discussion about the results. Finally, the conclusion and future works are presented in chapter seven.

CHAPTER TWO

ONLINE HANDWRITING RECOGNITION

The chapter presents a short introduction to pattern recognition that is the parent research stream of handwriting recognition. Then, the works done related to character recognition in general and handwriting recognition in particular followed by the classification of online handwriting systems are explained. Some issues, which deserve consideration during the development of online handwriting recognition systems, are presented next. Finally, the steps in online handwriting recognition task are detailed.

2.1. Introduction to Pattern Recognition

A pattern generally describes the way in which something is arranged to represent a given identified and classified entity. Examples of what we call pattern are a fingerprint image, a human face, a printed or handwritten character [1]. Pattern recognition is, therefore, the study of how machines can be made capable of learning and distinguishing patterns to classify them under predefined classes like humans are able to do [1,2]. Human beings are naturally gifted to learn and identify patterns easily and intelligently. For instance, it is not difficult for all of us to distinguish people we know by looking at their faces under normal circumstances. However, it is not a trivial task to teach machines to do the same thing .

Researchers have been focusing on pattern recognition for it has applications in variety of domains. It is also true that more applications are being generated due to the increasing power of data processors used to implement the pattern recognition algorithms that are known to be computationally complex[2]. Personal identification using attributes such as fingerprint or face in biometric recognition, semantic document classification and handwriting recognition are a few examples of these applications.

Various classification algorithms have been suggested and studied in relation to the particular applications. Although more than 50 years have passed in conducting researches

around pattern recognition[1] to come up with well-defined and generic methodologies, it is still a research topic having gaps here and there.
In summary, a pattern recognition system can be viewed as a black box receiving input and producing output[2]. Inputs to such systems are input patterns and the expected output is a classified and named entity.

2.2. Character Recognition

The term ‘recognition’ is a typical word used in many pattern recognition systems. Recognition systems are in general devoted to classify input patterns to corresponding entities. These entities vary from one system to another. Recognition systems, in which characters are expected to be classified, are referred as character recognition systems.

One major distinguishing feature of character recognition systems is the type of characters, which are supposed to be recognized. There are systems for recognition of printed characters, typewritten characters or written characters. Due to the varieties occurring to handwritten characters, development of the handwriting recognition systems is the most challenging.

2.2.1. Handwriting Recognition

Handwriting recognition is the task of transforming a language represented in its spatial form of graphical marks into its symbolic representation[3]. Symbolic representation refers to the digital representation of characters like in the case of 8-bit ASCII character set. Handwriting has been a basic form of communication and still a good way of expressing ones’ ideas. In relation to this fact, handwriting recognition systems are useful and are used to realize ideal applications of computers such as pen computing.

A number of classification methodologies to categorize handwriting recognition systems have been advised based on different factors. The subsequent sections explain some of these classification approaches, which are under interest.

2.2.2. Online Vs Offline Handwriting Recognition System

The basic input to character-based handwriting recognition systems is a pattern that represents a character. In fact, this pattern should be digitized before it is available to the system. The way the input is provided along with the digitization technique are taken as the two factors to classify handwriting recognition systems as online or offline.

The basic differences between these online and offline handwriting recognition systems as stated in a number of literatures[3, 4, 10, 24] are:

- **Input Method:**

  In online handwriting recognition systems, the handwriting data is obtained with the help of a transducer such as an electronic or tablet digitizer. Systems that simulate a pen-paper like arrangement for a writer are the sources of online handwriting data. Such systems record the pen-tip information as a sequence of (x,y) coordinates of data points sampled over time. On the other hand, in offline handwriting recognition systems, the data is captured optically by a scanner in the form of image. This necessitates the development of a method to identify the pixels which are part of the handwriting data pattern from pixels lie on the background [9].

- **Kind of available information:**

  For online system, the coordinates of successive points as a function of time in order is available where as in the offline case, only the completed writing is available. Additionally the speed of the writer, stroke number and order, and the pen-up/pen-down states are detected during the collection of online handwriting data. A pen-down state is detected when the pen touches the digitizer(writing pad) and when the pen is lifted off a pen-up state is sensed. This makes online systems easier to build than offline systems taking the abovementioned additional information as an advantage. Moreover, recognition rates generally have been reported to be higher for online systems than offline ones for the same reason.

- **Time of processing:**

  Clearly seen fact is that offline systems run after their data have been collected. The handwriting ought to be put on a media such as paper exhaustively and brought to the scanner, which in turn digitizes it. The case for online handwriting recognition systems is quite opposite. An online system receives handwriting data and recognizes it in real

or nearly real time. This situation adds one useful feature to online systems that is interactivity. Being interactive, online handwriting recognition systems allow users to edit what they have written and recognition errors can be corrected immediately. Online handwriting recognition systems are mainly used as an input to handheld or PDA-style computers that might replace the keyboard-based personal computers in the future[10]. Signature verification systems that are aimed at verifying identity of an individual with his/her signature, and educational software developed to aid children to learn handwriting are other applications of online handwriting recognition systems.

Offline systems could play a significant role in digitizing knowledge and information in the form of handwriting[4]. A research conducted by Wondesson Mulugeta on offline handwriting recognition for Amharic handwriting written in a special type of traditional style which is called 'Yekum Tsifet', is a good model to indicate the usefulness of offline systems by contributing towards the digitization of a number of historical and religious documents[8]. Other applications of offline systems are postal address recognition and check reading[6,7].

- **Speed Determination:**

  The speed of online systems depends on the writing speed of the user since the handwriting is analyzed and recognized in real or near real time. Conversely, in offline systems, the speed depends on the specification of the system in words or character per second.

- **Adaptation:**

  Adaptation of the writer to machine and machine to the writer is possible in online handwriting recognition systems. When the writer sees that some of her characters are not being classified correctly, she will alter the characters' shapes to improve accuracy and adapt to the system from time to time. On the other hand, some recognition engines are made to be capable of adapting to the user, basically by storing samples of the writer's characters for subsequent recognition. This two-side adaptation could never be made to happen to offline systems.

2.3. Online Handwriting Recognition

As stated earlier, online handwriting recognition systems attract researchers for their currently appreciated applications, which involve in pen computing. In the context of comparing these systems with offline ones, they take the advantage of the time information that is incorporated in online handwriting data. However, the task remains to be challenging in spite of the fact that the valuable additionally available time information stands besides them.

This section attempts in revising technical issues regarding online handwriting recognition systems. After introducing further classification of online handwriting systems, the major steps in an online handwriting recognizer will be detailed.

2.3.1. Categories of Online Handwriting Recognition

Recognition accuracy is the leading parameter to evaluate online handwriting recognition systems to decide on their usability. Looking into the history of commercial online handwriting systems, it is proven that developers started to put different constraints on the usage of their systems to get a reasonable accuracy rate[4]. Due to this reason, online systems started to fall in two types namely constrained and unconstrained systems.

2.3.1.1. Constrained vs. Unconstrained Systems

Constrained systems placed restrictions on writing styles. Some of them may want users to write in a discrete manner and some others force users to write in a given order of strokes[11]. Surprisingly enough, Graffiti, a product of Palm Computing that took the market by storm, placed the most suppressing restrictions on users by only allowing them to use only single strokes to write all the Latin characters [4]. The way users write in Graffiti is even far from natural handwriting since it obliges users to write only in a specified direction [4]. It also modifies the shapes of the characters

On the other extreme, unconstrained handwriting recognition systems allow users to enjoy writing in their own natural handwriting. Although these systems relax users, their recognition accuracy could be evidently lower than constrained systems [4]. That is the

reason why Graffiti has been more acceptable than CalliGrapher, which is stated as unconstrained system [4].

The amount of training data and its source used to train these systems is also a factor to come up with a different classification approach which introduces two types of online handwriting recognition systems: writer-dependent and writer-independent.

2.3.1.2. Writer-dependent vs. Writer Independent Systems

The goal of a writer-independent online handwriting recognition system is to recognize handwriting in a variety of writing styles, while writer-dependent systems are trained to recognize handwriting of a single individual[9]. A requirement of writer-independent systems is that they are expected to be able to recognize handwriting that the systems have not seen during training. Writer dependent systems lack this ability but still remain to be satisfactory for users of personal small devices. Yet, writer independent systems are necessary for applications like online form filling using a pen-like device.

Constructing writer independent systems is obviously harder than writer dependent systems. Moreover, writer dependent systems generally present a better accuracy rate. The difficulty of development of writer independent systems arises from the fact that the system is expected to handle much greater varieties of handwriting styles[9].

A distinguishable difference between writer dependent and independent systems is essentially in the amount of training data. It is possible to obtain large amount of training data when training a writer-independent system since many writers involve in the data collection process. For writer-dependent systems, the amount of training data will be limited, since the only provider is a single individual whose handwriting is being trained to the system[9].

2.3.2.Issues in Online Handwriting Recognition

In this section, we will explain the factors that make the construction of online handwriting recognition system challenging and the possible design goals that could be set when designing an online handwriting recognition engine.

Some of the common factors are explained below:

- **Handwriting styles: Cursive and block**

  Writers usually use different handwriting styles which could be one of the known defined ones or styles introduced by themselves. Variations resulting from this situation import challenge to the development of handwriting recognition systems. Some alphabets or languages have avoided some handwriting styles and others allow based on a rule. For example, Latin characters are written in a discrete way or cursive manner. When characters are written separately in boxes or virtual boxes created as a result of spacing between characters, the writing style is referred as discrete or block. Conversely, it is possible for writers of Latin alphabets to write characters by connecting them to each other which results in cursive writing style[10]. There is no way to control users not to write in mixed styles. However, in Amharic all characters are naturally written in discrete style.

  Variation is a source of errors in online systems and variations could occur both in time and space[10]. Variation in time refers to the variation in writing speed. Shape variation is another common difficult. These variations happen in both discrete and cursive handwriting styles. But, cursive style brings an additional problem that demands additional effort to identify the beginning and end of a character. Doing so is referred as segmenting the handwriting. Segmentation will be worse in handwriting of a mixed style[9].

- **Stroke number and order variation**

  A stroke is defined as the sequence of sample points occurring between consecutive pen-down and pen-up transitions[9]. A character could possibly be a uni-stroke character or formed from two or more strokes. The number and order of strokes is available for online systems, which is always considered as an advantage. In contrast, variation in stroke number and stroke order in a single character is a source of complication in online handwriting recognition system. Stroke order and stroke number variations are even terribly severe in particular systems such as Chinese/Japanese systems[12]. As a result, the issue of handling stroke order and number variations is one of the design goals when designing online handwriting recognition systems particularly the writer independent ones.

- **Limited Resources in small devices**

  It sounds good to remind that online handwriting recognition systems help in avoiding keyboard-based data input method in small handheld devices. It is not optional to let users of such devices input data in a pen-paper like environment by aiding the process with hardware digitizers along with pen-like devices and incorporating online handwriting recognition feature.

  It is good news that the storage and processing capability of handheld devices is improving from time to time. Limitation in resources has been a problematic phenomenon of these devices for recognition engine developers. The headache would be severe for large character sets such as the Chinese/Japanese cases [12,13]. Carefully designed algorithms to optimize the amount of data and the processing requirements are supposed to be attained for online handwriting recognition systems. A related design goal could be to make the systems platform independent and capable of dealing with different pen capture technologies that may reveal different characteristics such as varying sampling rate[5].

- **Character Set or Dictionary Size**

  Some character sets introduce additional problems to handwriting recognition system designers and developers. For instance, for character sets having a large number of characters, finding a match for a to be recognized character would be much more tough because the number of misleading prototypes would be higher in number. Evidently, more than 200 characters are included in the Ethiopic character set. This is a large number compared to the 26 Latin characters. However, there are bigger characters sets such as the Chinese character set which includes about 5000 characters when only the frequently used characters are counted [13]. Thus, character set size or dictionary size is another factor that contributes in the complicatedness of the development of online handwriting recognition systems.

2.3.3. Major Steps in Online Handwriting Recognition

Handwriting recognition is by no means a single-step process. It involves a number of steps in which a to be recognized entity (character, word or sentence) passes through before a recognition attempt. These steps are essentially used to avoid variations and are a way to identify only the important part of the data which is believably useful for recognition.

In general, recognition of handwriting patterns involves the following steps: Preprocessing, segmentation, feature extraction, classification and post processing. The following subsections detail these major steps.

2.3.3.1. Preprocessing

Handwriting data is subject to noises, which creates discrepancies and result in misclassification. The goal of preprocessing is to reduce or eliminate these variances in order to decrease the dissimilarity between a to be recognized entity and its correct prototype[9]. The noises introduced in the data are caused by the inaccuracy of the data capturing device, the erratic movement of the hand or fingers while writing and the varying writing speed of the writer [3,9,13].

There are preprocessing activities that have been implemented in various recognizers. These preprocessing measures applied on the online handwriting data could be grouped into two, based on their purposes: noise elimination (reduction) steps and normalization steps[13].

2.3.3.1.1. Noise Elimination

The term “noise” in pattern recognition systems in its broader sense is explained as:

> “anything which is part of the data that hinders the pattern recognition system from fulfilling its task.”.[2]

Naturally, the type of noise as well as the method to eliminate it is specific to pattern recognition systems. In this thesis, we are concerned about online handwriting recognition task and we narrow down our view to possible noises that may happen in online handwriting data. Noise in online handwriting data can be explained as a set of sampled

points which makes the trajectory of the pen tip somewhat jagged[9]. In another sense, additionally sampled points which are not part of the written character also tend to produce noisy data. To name a few of the common approaches to avoid these and other possibly occurring noises, we can state dehooking, filtering and smoothing[13, 18, 24].

Smoothing refers to the process of avoiding sampled points which make the trajectory rough. It is usually performed by using an averaging scheme over neighboring points[19]. Filtering means the elimination of duplicate points and hence reduction of data [24]. On the other hand, dehooking is a technique to detect and remove hooks which occur at the beginning and end of a stroke by a quick movement of the pen when it is raised or lowered[9]. Evidently more techniques exist, but as the quality of the input devices advances, only simple smoothing will be a sufficient preprocessing task [13].

2.3.3.1.2. Normalization

Variation between handwritten characters, words or sentences happen naturally. The variations that occur would be higher between in handwritings of different writers than of the same individual. In spite of that, since even a slight diversity sometimes result in a wrong classification results, techniques aimed at avoiding the variation are often implemented in any type of handwriting recognition systems. Some of these variations that probably occur include size difference, writing speed difference and difference in degree of inclination of say characters. Even in isolated character recognition which is much less complicated than word or sentence recognition, these variations greatly affect recognition accuracy.

Writing speed determines the number of data points that are captured or sampled while the pen is dragged on the electronic tablet or any other digitizer used. When the writer is writing slowly, more number of points will be samples than when the writer is writing quickly. To get rid of such variations, time normalization or most commonly called resampling is applied [9,14]. Equidistant resampling is the process of resampling the data such that the distance between adjacent points is approximately equal [13]. Besides the removal of the variance, this step essentially reduces the data which is a desirable effect.

Very noticeably, size difference between instances of the same character occur both in those written by a single writer or multiple writers. Sometimes, the extent of size variation depends on the way the writer is writing. For example, in the case of a boxed-input character recognition system, size dissimilarity would be minimal compared to the situation without this constraint. Whatever the case may be, a concern has to be given to size normalization. Size normalization is a way to reduce or enlarge the character to a predefined size[14].

Slant correction or rotation normalization is another typical normalization in which a character is rotated with the aim of avoiding writer-specific slants[14]. Translation normalization is also a normalization technique to translate all the data to the same spot relative to the origin[14].

The necessity of these normalization steps totally depend on the quality of data capturing devices, and the overall nature of the recognition engine which is supposed to be developed. It is the responsibility of the system designer/developers to decide on the relevant and important preprocessing steps.

2.3.3.2. Segmentation

At this stage, the preprocessed data is available. But, it is still a set of sampled points that are not yet meaningful. The preprocessed data needs to be segmented to parts to create meaningful entities for classification[2]. Segmentation is a way to find the representation of basic units that the recognition engine will have to process[3]. Segmentation may occur at different levels depending on the recognition algorithm used. The handwriting input may be segmented to individual characters or even into sub-character units[3].

For characters written in predefined boxes, most of the segmentation will be done by the writer. On the other side, the segmentation task would be harder and more error-prone for cursively written handwriting data[3] since it is not an easy task to determine the beginning and ending of individual characters[3]. For such cases, it is necessary to apply ***internal segmentation***, which requires some recognition to aid the isolation of the writing units[24]. Conversely, ***external segmentation*** is a type of segmentation that is entirely done prior to recognition[24].

Various segmentation techniques suited for online handwriting recognition have been advised. Some segmentation techniques exploit temporal information for the purpose of separating writing units such as by tracing the time difference between two writing units[24]. Others use spatial information[24].

In some occasions, the segmentation step might not be explicit and are tightly coupled with the previous preprocessing steps or the following steps[2]. Hidden Markov Models (HMMs) are good examples of handwriting modeling methods without an explicit segmentation step[10]. HMMs are tools to represent the continuous pattern of the handwriting data of any kind (character, word or sentence) as a single unit. It is possible to create character models, word models or sentence models as required. This makes HMMs to be a good choice to model cursively written words eliminating the difficult problem of segmentation of the words to individual characters. However, many researchers choose to put a segmentation stage before the HMM models are created.

2.3.3.3. Feature Extraction

Once the data has been preprocessed and appropriate segmentation has been done on it (if any), the data will be ready enough to apply classification. But, it is sensible that all the sampled data points are not equally useful for recognition while the possibility to do classification using the whole raw data is still possible[12,14]. Systems that rely on raw data generally have lesser recognition rate than those that rely on only the important part of the data or attribute(s) derived from the raw data[12]. Hence, the appropriateness of using selected features.

Features are high-level attributes extracted from normalized raw data[12] that describe the property of the handwriting data. For instance, number of strokes in a character is a very simple example of a feature that could be used (in combination with others) to classify characters. Many types of features are used in various recognition engines. Feature extraction, is therefore, the process of extracting only the relevant features from the online raw data with the aim of reducing within-class pattern variability and enhancing between-class variability[2].

The purpose of feature extraction is not only deriving high-level attributes for classification purpose but also reducing data. Data reduction in general has a positive effect in easing the processing of the computationally complex classification algorithms.

There are no standardized and widely recognized features for describing online handwriting pattern [12]. Due to this reason, features are usually selected manually[14]. And choosing appropriate features is a very critical issue for developers since this decision affects the performance of the system greatly.

In general, there are two types of features namely *online* and *offline* features[12]. There are advantages and drawbacks associated to each type of the features. To name a few, offline features lack the dynamic part of the data and online features are not robust to stroke order and number variations. Researchers have been incorporating offline features with online features in online handwriting recognition process and the vice versa[12]. This generally results in a better recognition accuracy rate. However, it is not straightforward to derive online features from offline data and in turn produce offline features from online data. For instance, it is required to extract the time information from an offline data to derive online features from offline data.

2.3.3.4. Classification

Classification could be regarded as the last and the most important step that produces the final output of the recognition engine unlike the previous steps that produce only intermediate outputs. The effort done to carefully design the techniques for the previous activities is to facilitate the success of the classification step[12].

The ultimate goal of this step is to carry out some form of comparison between a given unknown handwriting pattern to reference handwriting patterns to assign one of the references to the unknown one. In literature, several types of classification methods are advised in spite of the fact that none of them are perfect. Some systems divide the classification step into two: coarse classification and detailed classification [13]. Coarse classification is to assign a pattern to a group or multiple groups and detailed classification is to find out the particular entity associated to the unknown input from the group to which it is assigned [13]. Classification will be more detailed in the next chapter.

CHAPTER THREE

REVIEW OF THE STATE OF THE ART

This chapter reviews the state-of-the-art in online handwriting recognition briefly. A generalized description about the well-known approaches that have been applied in classification of handwriting data will be given. Then, a subsection reviews the relevant research papers on the trend of online handwriting recognition in three selected character sets, which are Latin, Japanese/Chinese and Arabic. We believe that this section presents the effort made to evaluate approaches used in recognition systems that deal with different characters sets and with diversified properties to come up with the better methodology for Ethiopic characters. Finally, the works on offline handwriting recognition systems for Ethiopic characters are reviewed.

3.1. Major Classification Approaches for Character Recognition

The most crucial step in handwriting recognition systems is classification. Classification is the task of classifying an unknown input pattern to one of the prototype patterns. It is evidently true that the effectiveness of the steps that have been carried out on the data before it is presented to the classifier have a direct impact on the performance of the classifier and hence on the recognition engine. The choice of the classification approach may also deteriorate or improve the recognition accuracy rate.

Classification approaches generally depend on the method used to represent the handwriting pattern which is referred by the feature extraction step[13]. Two issues are considered in any classification algorithm. These are pattern representation and matching method, which are discussed in brief in the following subsections.

3.1.1. Pattern Representation

Online handwriting data is essentially a set of sampled points where each point is described by its x and y coordinate values apart from the other information detected by the digitizer

such as pen up/pen down information and the pressure of the pen tip (See Fig. 3.1a & b). The raw data even after preprocessing will not be in the appropriate state to be compared with prototype or reference data and classified accordingly. This demands representation methods for the handwriting pattern.

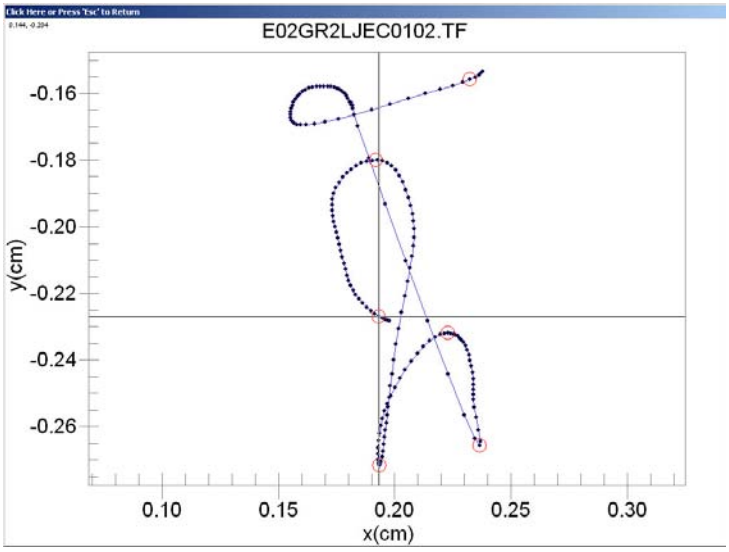

Fig. 3.1a) A handwritten Ethiopic character called '𝔉' (Collected by a digitizer software named 'MovAlyzer' using mouse as a writing device)

198	-228	99
197	-228	99
196	-228	99
195	-228	99
193	-227	99
191	-226	99
190	-225	99
188	-224	99
186	-223	99
185	-222	99
183	-220	99
182	-219	99
181	-218	99
178	-211	99
.		
.		

Fig 3.1b) Part of the collected data for the letter shown in Fig 3.1a. Each triple includes the x and y coordinate values and a number to indicate the pen up/pen down information (**99** for ***pen down*** and **0** for ***pen up***)

According to C.L.Liu, S. Jaeger, and M.Nakagawa [9], the pattern representation schemes could be divided into three groups: structural, statistical, and hybrid statistical-structural. Structural representation deals with expressing complex patterns in hierarchical perspective where a pattern is viewed as being composed of simpler sub patterns that are themselves built from yet other simpler sub patterns [1,13]. For instance, line segments could possibly be built from a set of sampled points and the line segments may combine to form strokes. Strokes, in turn, combine in a defined rule that specifies their relation to create characters. Hierarchical structure could also be created on top of the character model to organize the available strokes to share them to construct the character models of all categories [13]. This is quite a good example to show the level-based description of the pattern in which the lower components are used as primitives to build the upper components. The difficulty in representing patterns structurally is to detect primitives from

pattern data (which is prone to noise and variations) and to formulate the rules to combine primitives from a given training data set [1].

The other method for pattern representation used statistical techniques. These techniques attempt to represent the pattern in terms of *d* features or measurements[1]. This creates a feature vector representing the pattern. The feature vector could be viewed as a point in a d-dimensional space. Feature set selection is the main issue in this methodology, since it is expected that the feature set should enable the well separation of patterns into their corresponding classes in the feature space. Statistical representation schemes are becoming common in many recognition engines. To demonstrate feature vector description of patterns in statistical representation, let us pick one of the referred recognition engines systems. In [11], feature vectors are generated for each data point (x_i, y_i) as (x_i, y_i, x_i-x_{i-1}, y_i-y_{i-1}) where x_i-x_{i-1} and y_i-y_{i-1} will be set to 0 if i<2 and i is the index to refer to the points. This is a feature vector having four features. Actually, the system is an Hidden Markov Model based system and these feature vectors are later quantized to generate observation symbols that are known to be an input for the HMM [11].

The third representation scheme is a hybrid one that takes the advantages of both methods which are described above. A pattern model is represented by a string, using tree or graph structure, with the primitives and/or their relationships measured probabilistically by replacing the attributes of primitives and/or relationships with a probability density functions [13]. A good example is the Hidden Markov Model, which is a graph with nodes and the between-node transition, are measured probabilistically.

3.1.2. Pattern Matching Techniques

All recognition systems are equipped by a set of prototype patterns corresponding to entities or collection of primitives to build entities that are to be recognized. These prototype patterns are obtained mostly by learning or occasionally integrated to the recognition system when they are first constructed like in the case of Graffiti in which all letters are represented by a single predefined stroke[4]. In any case, there will be a matching operation to compare the unknown pattern to the existing prototype patterns for the purpose of assigning it to one of the classes. Once pattern representation scheme is designed, the next decision would be the way the matching must be done.

Pattern representation and pattern matching are related in many ways and usually treated together and even integrated and referred by one name in many literatures. However, we prefer to discuss them separately to match with our own classification approach that is developed in this work, and that separately considers them. Structural matching, probabilistic matching and statistical matching are generalized names for the three types of matching techniques[13].

Structural matching is the process of finding a match for unknown pattern represented in structural representation scheme from set of prototype patterns, which are also represented structurally. The match will be searched by computing matching distances between the unknown pattern primitives and the primitives of the candidate patterns and take a class with the minimum distance[13]. Since a grammar or rule is defined in structural representation of patterns, the match is found according to the rule. Structural matching is sometimes called syntactic matching.

Probabilistic matching is computing distance between models and unknown patterns, which are, represented structurally where the attributes are probabilistic [9]. In statistical matching, a character is recognized by choosing either the most probable character or choosing the character with the least probability of wrong classification [1].

Specific recognition algorithms are discussed in the following section while looking into existing online handwriting recognition systems designed for various character sets. These include structural string matching, template matching, Hidden Markov Model and neural network.

3.2. Review of Relevant Papers on Online Handwriting Recognition Systems

Naturally, a difference exists between characters in different character set. This difference is in number, shape, degree of similarity between characters of the same set, and many more other behaviors. Consequently, selected approaches should take the nature of the characters in the character set to which a recognition engine is going to be developed into consideration. Basically, the difference in the usage of classification approaches is inherited from the fact that characters in different character sets possess diversified nature.

In this section, we perform a comparison of recognition systems for three character sets namely Latin, Chinese/Japanese and Arabic characters.

These three character sets are chosen for the reason that they exhibit a great difference that will reveal which approach is convenient to which type of character set. The Latin character set contains 26 alphabetic letters and 10 digits. Each of the alphabetic letters have two forms namely capital and small letters. Without including punctuation and other symbols that involve in writing, there are 62 characters in the Latin character set.

The Japanese writing system is well described in [8]. It is a mixture of 3 types of characters: ***Kanji*** (Chinese traditional characters), ***Hiragana*** and ***Katagana***. The total number of characters is roughly 6000-7000. Moreover, Japanese characters are divided into two classes namely JIS first level (most common characters) and JIS second level (less common ones) for the purpose of computer processing. The paper also stated that the main difference between Japanese and Latin handwriting recognition systems is caused by the large number of Japanese characters. The Chinese character set is divided into three subsets: ***Traditional Chinese characters***, ***Simplified Chinese character*** and ***Japanese Kanji characters***[13]. Moreover, there are three writing styles: ***Regular script***, ***Fluent script***, and ***Cursive script***. The main problem in Chinese online handwriting recognition is stroke-order and stroke-number variability. Additionally, recognizing fluent style handwriting has been a difficult task.

Arabic characters are 28 in number and all of them are cursive [14]. As indicated in [14], the shape of the 28 characters is constructed based on 18 distinct shapes that vary according to their connection to the preceding or following letters or by putting dots and other symbols above or below them. The key difference between Arabic and Latin characters is the fact that many letters differ only by a '**.**' while the primary stroke is the same.

3.2.1. Latin Online Handwriting Recognition Systems

In this subsection, we present four papers relevant to our work and for which summarized explanations are offered.

3.2.1.1. Kam-Fai Chan et al.: Flexible Structural Matching

The research goal in this paper[16] is to build a handwriting recognition system, which is speedy, accurate and flexible. The system is expected to tolerate variations and maintain reusability hence facilitates extensibility.

The main approach is a structural string matching technique. Characters are viewed as they are made up of line segments and curves. The line segments and curves incorporated with directional information help to define primitives that are building blocks of strokes. These primitives are *line*, *up* (curve going counter-clockwise), *down* (curve going clockwise), *loop* (curve joining itself at some point, and *dot* (a very short segment). A hierarchical structured representation is employed for the purpose of the handwriting pattern representation in which a character is represented by stroke set, stroke set is a collection of strokes, a stroke is a collection of primitives which are listed earlier. To model this, a 4-tuple grammar is also introduced.

Once a character is handwritten, what we get is a set of points from which the *primitives* that make up the character are going to be *extracted*. The structure extraction and reconstruction mechanism is described for each of the primitives. After extracting these primitives, their description will be compared with the model set to find a match. If an exact match is not obtained (due to the variability of handwriting style), *flexible structural matching* will be performed to change the shape of the character slightly in order to find the closest character. Thus, the matching process will be continued until a match is found for a character. Four levels of structural deformation are included namely no deformation, primitive type deformation, directional deformation, and simultaneous type and directional deformation. However, some ambiguities may still occur in flexible structural matching and an additional step *post-processing* to find the most probable character is also implemented.

The online handwriting data set collected by the MIT Spoken Language Systems [31] is used to test the system and 98.60% accuracy for digits, 98.49 for upper case letters, and 97.44% for lower case letters are reported. The overall accuracy of the system is 97.40%.

The authors also claimed that reasonable speed has been achieved which is 7.5 characters per second when running in Prolog on Sun SPARC Unix workstation and the memory requirement is also reasonable low.

3.2.1.2. G. Loudon et al.: Discrete Hidden Markov Model

This system[11] is developed to run on smart phones and the authors stated that the system considers the limitations of the device and the parameters to make recognition systems usable. Accordingly, the following requirements are outlined:

- The system should run on slow processors and low memory,
- The system should have high accuracy,
- Fast text entry has to be facilitated,
- An easy text correction mechanism should be incorporated.

The authors claim that to achieve these goals, some restrictions (constraints) are placed on the users of the system. These are:

- Users should write using only small letters (can change to upper case letter later when editing).
- Users must use a predefined style of writing even though it is not like Graffiti, which forces users to write a letter by a single stroke. In this system, the shape of the letters is kept as their natural shape. However, every letter is written in a predefined way as indicated in the paper.
- Users are only allowed to write in a hand printed manner (not cursive).

The main approach used is based on discrete Hidden Markov Model. The Hidden Markov Model (HMM) is a statistical-structural pattern representation method. Hidden Markov models are used extensively and successfully in speech recognition and are good to represent sequential temporal data.

The system has two modes: *Training* and *Recognition*. In the **training** mode, isolated characters are provided as a training data set. Character pre-processing; which includes *character normalization*, *stroke joining*, *stroke re-sampling*, *feature extraction* and finally

vector quantization is applied. Character normalization is also done to avoid shape distortion of characters. Stroke joining is the process of joining two strokes, which are part of the same character (for eg. **f**, the ⌠ and –). Stroke re-sampling is necessary to make the average speed of writing a character is a constant across all characters. Feature extraction is another important step to pick four features on each data point which are (x_i, y_i, x_i-x_{i-1}, y_i-y_{i-1}). Vector quantization is to convert a multi-dimensional vector to a discrete symbol and it is essentially a data-compression technique that will reduce computation complexity in the HMMs. After the pre-processing tasks are accomplished, each character is modeled by two ten-state HMMs from which one is based on x_i, y_i and the other is based on x_i-x_{i-1}, y_i-y_{i-1}.

In the **recognition** mode, sentence recognition is supported. Four steps are going to be followed namely *segmentation* which segments the sentence to isolated characters, *character pre-processing* (same as the training case), the *Viterbi search* in which probability of match between the input data and all characters will be calculated and sorted and then the first five characters will be stored, and finally *re-scoring* which uses simple language knowledge to improve accuracy.

The system has also an additional feature that is text correction. The module provides a very interesting method of correction by allowing the user to write on top of the already written text or delete by crossing words.

Experiment results are also reported. The training data consists of 11984 different characters written by 13 writers. The test data includes 5559 characters written by 7 writers. Accuracy rate is reported to be 98.3%.

3.2.1.3. H. Shu: Hidden Markov Model

The paper[10] emphasized the fact that HMMs are initially applied to speech recognition tasks and it had also been proven that they could also be used to build an online handwriting recognition system. The main goal of this research is to improve the accuracy of a system named BYBLOS by adding features. The BYBLOS[32] online handwriting recognition system was built by adapting the BYBLOS CSR[33] speech recognition system. Moreover, there is an attempt to build a real time handwriting recognition system

of the baseline BYBLOS handwriting recognition system without sacrificing much accuracy. The BYBLOS recognizer is a writer-independent mainly concerned with cursive handwriting. Moreover it is sentence-based recognition. No constraints are imposed on the users except the following regulations while collecting the BBN data corpus [35] by which the research is based on.

- All small letters should be connected which means mixed style (discrete and cursive) is not allowed.
- The dots above letters like **i** should be written after the word is completely written i.e. it is not allowed to lift the pen without finishing a word.

The BYBLOS online handwriting recognition system has three modules: *Front-end*, *trainer* and the *decoder*. The ***front-end module*** is responsible for generating observation sequence for the HMM from the input handwriting data. It includes three steps: *pre-processing* (constructing invisible stroke and padding filter), *computing feature vector* with 6 features (pen up/pen down bit, delta-x position, delta-y position, writing angle, delta writing angle, and sgn(x-max(x))) and *computing the observation sequence* (by converting the feature vector to the observation sequence which could be the input for the recognizer). The ***trainer module*** is implemented for the purpose of training the HMM. The ***decoder***, employs the fast match search and the Viterbi beam search to produce the most-likely sentence. It is reported that it has achieved a good accuracy having only a 13.8% error rate.

Then, the paper suggested that it is still possible to improve the recognition accuracy to an acceptable and usable level by adding more features which were not included in the feature vector. Accordingly, four features are added: the *vertical height* (will help to distinguish characters like **‘** and **,**), the *space feature* (a new way of handling inter-word spaces by which space features will be calculated only for invisible strokes), hat stroke (to handle the misleading situation of putting dotes and crosses once the word is written like in **i** and **t**) and *sub-stroke features* (strokes are further divided to sub-strokes to incorporate information within a bigger neighborhood of the sample). After adding these features, the error rate is reduced by 34% and it becomes 9.1%. The improvement after each features is added is summarized in Table 3.1 below.

Feature added	Feature vector size	Previous error rate	Current error rate	Error-reduction percentage
Vertical height	7	**13.8%**	13.6%	Not significant
Space feature	8	13.6%	11.3%	Significant
Hat stroke	9	11.3%	10.7%	Significant
Sub-stroke features	10	10.7%	**9.1%**	Significant

Table 3.1. Improvement of accuracy by adding features

A real time online recognition system that run on an SGI Indy workstation and GRiD. PC is also built.

3.2.1.4. E. Gomez et al. :Fuzzy Neural Network

The following research goals are set in the paper[15].

- Processing time should be low enough to run on a Pentium 120MHz and to complete the tasks within specific time (276ms/char for the test process, 7ms/char for segmentation, 5ms /char for feature extraction and 256ms/char for the actual test).
- Training time should be low.

The main approach is applying the neuro-fuzzy system named FasArt to build the recognizer.

The system is presented in the paper to have three stages: preprocessing (basically character segmentation), feature extraction and classification. The character segmentation is done in two different approaches one of which is based on geometric features and the other is based on biological models. Both methods try to find the optimal segmentation points from a given character. Then feature extraction is conducted in two ways: Shannon entropy and clustering maps. The features that they use are length of stroke, two phase points, total phase and a feature to tell whether it is the beginning or the end of a stroke. Finally the classifier based on Fuzzy logic system is applied.

The test is conducted based on the UNIPEN data set. The system achieved an accuracy rate of about 85.39% for digits, and 59.57% for lower case letters and 66.6% for upper case letters when geometric segmentation is used. On the other hand, employing biological model segmentation, an accuracy rate of 82.52% for digits, 76.39% for upper case letters and 58.92% for lower case letters is achieved.

Processing time required by the proposed system when tested on a Pentium 120MHz PC, measured with Linux time, is 276 ms/char for the whole test process, including 7 ms/char for segmentation, 5ms/char for feature extraction, and 265 ms/char for the actual test. An overall speed of 0.28 sec/char is achieved by the proposed system.

3.2.2. Japanese/Chinese Online Handwriting Recognition Systems

Two papers that compare and contrast the currently used techniques in Japanese and Chinese online handwriting recognition systems with the methods applied in Western (Latin) systems are summarized below.

3.2.2.1. S. Jaeger et at.: The state of the Art in Japanese Online Handwriting Recognition

This paper compares the techniques used in Japanese online handwriting recognition systems with those used in Western systems[12]. The recognition is divided into three processing steps: preprocessing, classification and post processing. It is also stated that many statements made in the paper are also valid for Chinese character recognition, or Asian character recognition in general, since the Japanese character set comprises a subset of the Chinese character set. The set of techniques for each step in Japanese and Western systems are analyzed as follows:

Preprocessing Techniques:

In this section, the paper compares the techniques used for the preprocessing tasks *normalization*, *re-sampling*, *slant correction* and *data compression* in the two recognition systems.

- The most common normalization in JOHR[1] is size normalization, which could be linear or nonlinear. It is also reported that nonlinear normalization techniques (such as line density equalization technique) are better that linear methods as they improve recognition accuracy.
- It is usual to compare Japanese characters and Western words in terms of complexity. But, normalization techniques used for Japanese characters are not well suited for Western words. For western words, size normalization is done by first identifying lines such as base line, corpus line, ascending line, descending line and etc. Then the corpus

[1] This is to say Japanese online handwriting recognition system.

height (height between baseline and corpus line) will be mapped to a fixed height to normalize the words.

- Re-sampling is common in Western systems where as it is not usually implemented in JOHR.
- Slant correction is also another normalization task which is common in Western systems, but are not generally implemented in Japanese systems.
- Data compression is a common task in western as well as in Japanese systems.

Feature Computation:

Feature computation is the next task once the preprocessing is done to the raw handwriting data. Features are high-level attributes derived from normalized raw data. There are two types of features namely online and offline features. These two types of features have their own positive and negative sides.

In general, offline systems (solely based on offline features) are less accurate than online systems and this implies that online features have positive impact. Conversely, online recognizers suffer from stroke number and order variations (even worse in Japanese and Chinese systems). Thus, incorporating online features with offline features in offline handwriting recognition systems or vice versa is a common practice to improve accuracy. Regarding this, the western people focus on incorporating online features with offline ones for offline handwriting recognition systems. But in Japanese, the reverse is true. As stated above, Japanese online handwriting recognition systems try to use offline features to handle stroke order and number variation found in online data. But extracting offline features from online data is not straightforward. Histogram features are statistics describing absolute and relative frequencies of feature values in handwriting trajectories. Histograms help to extract offline data from online data by avoiding the dynamic part of the online data. The trajectory in the handwriting will be divided into equal cells and the selected feature values (such as directional feature, directional feature) will be counted within the cells, leaving their order of occurrence i.e of course the dynamic part of the data.

Classification:

Obviously, the final target of recognizers is to assign a character to unknown pattern. The paper reviews some classification methods in Japanese and western recognizers.

Template Matching techniques:

Most JOHR systems are based on nearest neighbors classifiers (template matching techniques). This technique will compute the distance between the unknown pattern and a template stored in a database and assign the template with shorter distance to be the classified character. More elaborated template matching techniques like ***elastic matching techniques*** are also used. Elastic matching (sometimes called Dynamic Programming matching-DP matching) finds the matching template, which requires the least deformation necessary for transforming it to the unknown input pattern. DP matching is mostly used in post-processing in western systems, but in Japanese systems, it is a core technique for the classification step.

Hidden Markov Models:

HMMs are becoming common approaches in Western systems, however there are only a few number of Japanese recognizers that use HMMs. The reasons are:

- HMMs are good for implicit segmentation and are used for cursive Latin words, but segmentation is not a main problem in JOHR.
- HMM modeling will be very complicated for Japanese characters due to stroke order and number variations. Even those JOHR systems applying HMMs use different modifications.

AI Methods:

AI methods such as neural networks have shown the same level of applicability in both western and Japanese systems. However, it is reported that, these methods were very much appreciated around the 1990s and became faded during the middle of the last decade.

Pattern Representation:

Application-specific dictionaries in western systems facilitate recognition. The dictionary will be represented by a tree whose nodes are characters in the word. This will make searching very fast because it avoids flat searching techniques. But in the case of Japanese, such a tree will be prepared for ***characters*** (**not** words) where the leaves of the tree represent the subparts of the characters usually called radicals. It is here fair to note that

Japanese characters are usually compared with Western words (not western characters) in terms of complexity.

Recognition rates:

Talking about recognition accuracy of systems really needs a standardized data on which training and testing could be conducted. Considering this fact, there are different projects around collecting online data set. For western systems, we can state two of them: UNIPEN and IRONOFF. Japanese are also doing the same thing and two of the benchmark data sets are mentioned in the paper. These are Kuchibue (having around 1,435,440 handwritten Japanese characters) and Nakayosi. The two of them together contain more than 3 million characters written by 283 writers. Recognition rates based on Kuchibue is around 90%. For western systems, the paper reported the recognition rate only for the published Npen++ system[34] as 96% (WI mode & 5000 word-dictionary), 93.4% (20,000 word-dictionary) and 91.2%(50,000-word dictionary).

Post-processing:

Postprocessing is the process of improving recognizer output by means of additional information sources such as vocabulary, spelling etc. Most western handwriting recognition systems employ ***bigrams*** and ***trigrams***. Bigrams and trigrams provide statistical information on character level or word level. They are generally used to indicate the probabilities of combinations of characters in a word or words in a sentence. In Japanese systems, a similar technique of employing bigrams and trigrams and even n-grams for postprocessing has been applied only sometimes. Another postprocessing activity is combining classifiers such as combining online and offline systems. Improvement by combining online/offline systems has already been reported for Asian character recognition.

The following points are emphasized in the paper regarding major classification approaches applied in Japanese/Chinese and Western systems:

- Most western systems adhere to HMM, but most Japanese handwriting recognition systems use nearest neighbor technique.
- HMMs and neural networks are not commonly applied in Japanese handwriting recognition systems due to the high number of Japanese characters and stroke order

and number variations occurred during writing. But substroke HMMs could be good options.

3.2.2.2. C.L.Liu et al.: The state of the Art in Chinese Online Handwriting Recognition

According to C.L.Liu[13], A typical Online Chinese handwriting recognition system passes through the following steps:

a) *Character segmentation*: The handwriting input is segmented into character patterns, but the segmentation is not for sure 100% correct at this early step until contextual processing is done. So, what we get from this step is a set of segmented (candidate) patterns.

b) *Preprocessing*: It includes noise elimination (by smoothing, filtering, wild point correction and stroke connection), data reduction (equidistance sampling and line approximation [feature extraction]) and normalization (linear, moment, and nonlinear where nonlinear is the efficient technique for Chinese characters).

c) *Pattern description (Feature Extraction)*: This step is a very important one since the classification is actually carried out by finding a match between the pattern description of the input data and the reference model. Three representation schemes are used: statistical, structural, and hybrid statistical-structural.

d) Character classification: The character classification is divided into two stages namely ***coarse classification*** and ***fine classification***. During the coarse classification, multiple classes are assigned to the input pattern either by using class set partitioning or dynamic candidate selection techniques. The techniques used in fine classification are grouped into three: structural matching, probabilistic matching and statistical classification.

Structural Matching: The input pattern is matched with the structural model of each candidate class and the class with the minimum matching distance is taken as the recognition result. Various approaches are used. ***DP matching*** (choosing minimum edit distance), ***Stroke correspondence*** (the distance is computed by summing up the between-stroke distances), ***relational matching*** (the correspondence is searched under the constraint of relationships) and ***knowledge-based matching*** (the prior knowledge of character structure and writing is used as a constraint) are mentioned.

Probabilistic Matching: Matching distance between the input pattern and the references will be computed and the character with lesser distance will be taken as recognized characters.

Statistical Classification: The input pattern will be described in feature vector and various statistical techniques are applied to find a match.

e) *Model Learning and Adaptation:* During classification (whether it is coarse or fine), the model database is consulted. The model database can be built by model learning or manually using prior knowledge. But model learning improves accuracy than manually built models. Different learning approaches are described. These are mean prototype learning, multi-prototype learning, structured learning and writer adaptation.

f) *Contextual Processing:* This activity helps to find the optimal match by using linguistic knowledge. Moreover, it also helps in assuring segmentation.

Performance Evaluation:

The performance of recognition systems is measured in terms of recognition accuracy, memory requirement, and recognition speed. The evaluation reported on the paper is based on the Nakayosi and Kuchibue data set. And in summary,

- The authors believe (based on collected information) that it is currently possible to achieve a recognition rate above 98% for regular scripts. But for fluent (or fluent-regular) scripts, it is difficult to achieve a recognition rate above 90%.
- Statistical methods use large amount of memory (For eg. over 30MB) and offer high speed where as structural methods are slower but use a small amount of memory (for eg. 166KB).

3.2.3. Arabic Online Handwriting Recognition Systems

A paper by T.Klassen[14] that presents Arabic online handwriting recognition system developed by using neural networks is summarized below. The good thing about the paper is that it also reviews prior works.

Review of Previous Works:

The author of the paper reviewed six previous works. The following table summarizes the review. (See Table 3.2)

No.	Developer(s)	Technique used	Accuracy
1	Al-Sheik & Al-Taweel	***Hierarchical Rule-based***	100% for isolated letters, but unclear whether this rate is obtained on the training or test data
2	El-Emami & Usher	***Segmented Structural Analysis***: (Structural analysis for feature selection and a decision tree for classification)	Tested on 10 writers. They instructed one of the tester (with 86%) to change his style of writing and they get 100% accuracy.
3	Bouslama & Amin	***Structural and Fuzzy:*** Structural analysis is for classifying different characters & fuzzy approach handle the variability within people's handwriting for the same class.	It is reported on the paper that no test result had been listed.
4	Alimi & Ghorbel	***Template matching and Dynamic Programming***	Using 9 prototypes, 96% accuracy on test data for one author.
5	El-Wakil & Shoukry:	***Hierarchical Template Matching and k-nearest Neighbor Classification***	84% by testing 7 writers on sets of 60 characters.
6	Alimi	***Evolutionary Neuro-Fuzzy***	89% without dot or diacritical information

Table 3.2: Previous works on Arabic Online Handwriting (adapted from [14])

Then the paper also identifies the strength and weakness of the reviewed works. Another table is provided below to outline the pros and cons of the methods (Table 3.3).

No.	Method	Strength	Weakness
1	Hierarchical Rule-based	Good divide-and-conquer strategy	Sensitive to noisy data
2	Segmented Structural Analysis	Automatic segmentation of words	Sensitive to rotation and less data was used.
3	Structural and Fuzzy	Had perfect training results	No test result is reported
4	Template matching and Dynamic Programming	Accuracy (96%) was very good	Only one test subject involved and difficult to generalize
5	Hierarchical Template Matching and k-nearest Neighbor Classification	Good accuracy	Sensitive to noisy data and difficult to generalize since the test set contains only 60 characters
6	Evolutionary Neuro-Fuzzy	Was more robust, and segment in a novel way	The test set was constrained to only one word and a small subset of 7 different letters and the system is writer-dependent.

Table 3.3. Strength and Weakness of Approaches for Arabic online handwriting system.

General Description of the System:

The phases of this system are *data collection*, *file storage*, *segmentation*, *sampling*, *normalization*, *feature extraction* and *classification.*

Data collection: The data collection was done using Wacom Graphire Model ET-0405-U. 2769 samples were collected. The program used to collect data is WinTab during data collection, noises were evident to appear and handled in different ways.

File Storage: Though the ultimate goal is to make the system work on real time, it is difficult to train and test the system repeatedly by bringing people every time an experiment needed to be conducted. So, it could be simulated by storing the data in a persistent form like a file. The data was stored in standard format namely in UNIPEN.

Segmentation: Since isolated characters are considered, word segmentation to letters is assumed to be done. But segmentation of letters into primary and secondary strokes is implemented.

Sampling (Critical Point Extraction): To reduce the amount of data, only critical points will be extracted that can characterize the character. The procedure for ChkLOS (check line of sight) is used to choose the critical points.

Normalization: The main purpose of normalization is to make the size of the letters the same and put in the same position. *Scaling normalization* (size normalization) is done to handle the size variability of people's handwriting by mapping all handwritten letters to a pre-defined size. *Translation normalization* is another normalization activity to translate the letter to the same position relative to the origin. Another type of normalization is *time normalization*, which is used to normalize the temporal sequence data. *Rotation normalization* is the other normalization tasks included and used to make the data set rotation-invariant. Moreover *skew normalization* is considered to correct slants.

Feature Extraction: The two purposes of feature extraction are to see that all sampled points are not useful for pattern recognition and to reduce the amount of input data for the case of neural networks. The system uses a neural network called SOM (Self-Organizing Maps). It is reported that SOM has a capability of extracting relevant features used to classify characters avoiding the manual extraction of features. Justification to use this tool and its design are presented in the paper in detail.

Classification: Multi-layer perceptron (a kind of neural network) are employed to carry out the classification task. The paper stated that online recognition of Arabic characters is a non-linear problem and these neural network are appropriate. Optimization methods such as pruning and partitioning are also applied.

Experimental Results:

The data for training and test were collected as described above. The experiments had 3 trials on 3 disjoint data sets: **training** (1656 letters), **validation** (1113 letters) and **test** (692 letters).

Trial	Training data set	Testing data set	Accuracy
Trial 1	Training set	Validation set & test without partitioning and pruning	72%
Trial 2	Partitioned & pruned on Training set	Validation set & test set	80%
Trial 3	Trained with training set & partitioned and pruned on validation test	Test set	85%
Average			79%

Table 3.4 Recognition Accuracy reported in [14]

3.2.4. Handwriting Recognition Systems For Ethiopic Characters

As far as we know, there are only three researches [6, 7, 8] done on handwriting recognition for Ethiopic characters. All of them are offline handwriting recognition systems. Though the previously mentioned systems are all offline, we presented them here to see how Ethiopic characters are handled.

3.2.4.1. Amharic Handwriting Recognition for processing of Bank Checks

Nigussie Tadesse[6] conducted a research on handwritten character recognition for the first time. He attempted to adopt a recognition algorithm for handwritten Amharic text so as to apply the result of the research to the processing of bank checks. After collecting data (amounts of money written on various checks in Amharic words), a pre-processing step had been taken to minimize the intra-class variability through underline removal, stroke width normalization and slant normalization if any. After extensive and careful pre-processing activities were carried out, the segmentation of characters had been done by extracting connected components first, then examining the aspect ratio and proximity of each connected component from its neighbor.

A multilayer feed forward neural network with error-back propagation is applied for recognizing the character after normalizing each segmented image of the character to a 16×16-pixel matrix. Then, the neural network is trained by giving specific values for parameters such as target error, learning rate and momentum and this training took about 12 hours (See Table 3.5). Unfortunately, the recognition accuracy of the network was very poor when it was tested after the training. This situation initiated the researcher to increase the hidden layer neurons to 20 and the number of sample characters for training by 7 in an attempt to get a better result. But no significant improvement was exhibited. For the third time, a neural network is created and trained with a different value of target error and with more characters. However, in spite of all the trials, the neural networks didn't give any better achievement. The result of the work is summarized in Table 3.5 below.

The parameters in the table (Table 3.5) will affect the recognition capability of the neural network and they are part of the architecture of the neural network. The researcher increases the number of hidden neurons to improve the accuracy of the neural network.

Parameters to determine characteristics of the neural networks	1st neural network	2nd neural network	3rd neural network
Input neurons	256	256	256
Hidden neurons	7	20	20
Output neurons	8	8	8
Target error	0.05	0.05	0.18
Learning rate	0.6	0.6	0.6
Momentum	0.8	0.8	0.8
No. of characters for training	135	498	498
No. of hours taken for training	12	24	14
No. of characters for testing	38	38	38
No. of characters correctly recognized	2	*	16

Table 3.5: Summary of result of [6]

3.2.4.2. Amharic Handwriting Recognition for Postal Address Interpretation

Messay Hailemariam[7] attempted to apply OCR technology to the domain of postal addresses interpretation by exploring and testing the application of simple geometric calculations and line fitting techniques for handwritten Amharic character recognition.

In this work, pre-processing activities are conducted in a limited way to remove noises (undesired artifacts) caused by interferences of strokes of some characters into other characters. After the image of the document is scanned and digitized, segmentation technique, which is known as stage-by-stage segmentation in which line, word and character segmentation are done in order to come up with the image of the individual characters. A 32×32 pixel size is chosen as a standard size so as to normalize the characters. Once the normalized characters are produced, the feature extraction is done by local line fitting (LLF) technique which is based on simple geometric operations. The method is known to be efficient and fast because it avoids some pre-processing steps made on the input. Finally, the neural network is trained by taking the extracted features as an input vector.

* The number of characters correctly recognized is not clearly stated, but it is reported that a significant improvement did not exhibited.

The research is completed by getting the following results:

1. Testing the neural network for the data that it is already trained with results in having a success rate ranging from 73.25%-91.9%.
2. The neural network was able to recognize new handwritten characters which were not in the training set with a success rate ranging between 33.25% and 88.6%.

3.2.4.3. Traditional Amharic Handwriting ("Yekum Tsifet") Recognition

In 2004, Wondwossen Mulugeta[8] extended the research on Amharic handwriting recognition using OCR. He intended to apply the OCR technology to recognize special type of handwritten Amharic text which is traditionally called 'Yekum Tsifet' using all the 231 basic character sets [Wondwossen, 2004]. Wondwossen studied the basic features of the traditional handwriting style and applied the artificial neural network approach to develop a recognition engine for this purpose. He also investigated the application of the result of the research by claiming that converting the existing large number of handwritten historical and religious document written in 'Yekum Tsifet' would help to access the information digitally. In his work, Wondwossen used the global thresholding approach to binarize the documents. Stage-by stage segmentation techniques were found to fit remarkably for the extraction of 'Yekum Tsifet' characters since they are written at the same level without ascent and descent features [Wondwossen, 2004]. Visual C++.Net built in function is used to normalize the extracted characters to a standard size of 20×20 pixels or alternatively 16×16 pixels. Though this process may deform some of the characters, it is a vital step to create an appropriate standard sized input vector for the neural network recognizer. This step is followed by the production of the binary representation of the normalized characters in a text file.

Wondwossen designed and developed an artificial neural network character recognition engine that accepts the binary representation of a segmented character as an input and produces the corresponding recognized character (or 'Fedel'). The architecture of the neural network, which is a feed forward neural net with back propagation algorithm, is a multiplayer perceptron that has three layers namely the input, hidden and output layer.

The result of the research is reported as follows:

1. The artificial neural network recognizes most of the patterns it is trained with in a recognition rate of 95.24%-98.7%.
2. The artificial neural network recognizes new patterns with a recognition rate of 16.1%-31%.
3. The artificial neural network is tested with original training patterns with different level of random noise imposed on it and a recognition rate of ranging between 27.27% and 94.8% is achieved.

3.3. Summary

To wind up the survey, we have provided the following points to summarize what we obtained from the survey:

- Due to the cursive nature of Latin handwritten words and sentences, discrete Hidden Markov Models are increasingly used in Latin online handwriting recognition systems. The reason is that HMMs facilitate implicit segmentation which is the most difficult problem in online word-based and sentence-based Latin recognition systems. Limitations that are related to HMMs are the necessity to create individual models for all entities (character, words or sentences) and stroke order dependency. The number of Latin characters is minimal relative to Japanese/Chinese or Ethiopic character sets and Latin online systems benefit from the fact that their characters are not as many and as complex to be modeled by HMMs. However, as a direct consequence, HMMs are not commonly adopted in Japanese/Chinese systems due to the large number of characters and the complexity of the characters. Another reason for not considering HMMs for Japanese/Chinese systems is that Japanese/Chinese writing is subject to stroke number and stroke order variations. In spite of this, some researchers claimed that substroke HMM are good options for Japanese/Chinese systems.
- The complexity of Japanese/Chinese characters make them to be compared with Latin words (*not* characters). Structural pattern representations and template matching are common techniques in Japanese/Chinese recognition systems. Hierarchical representation such as tree representation scheme is employed to represent a Japanese/Chinese character where the leaves of the tree are the strokes in the character unlike the case of Latin systems in which trees are sometimes used to hierarchically model words where leaves are letters.

- The analysis done on online systems for Arabic characters show that structural methods are commonly used to build Arabic online handwriting recognition systems. However, it is also reported that good recognition accuracy is obtained by applying neural networks.

The Ethiopic character set is not as simple as Latin character set. However, it is not also as complex as Japanese/Chinese character set. This conclusion has been reached based on the fact that the number of characters in the Ethiopic character set is more than 300 which is a big number as compared to the 26 Latin characters. The Japanese character set contains 6000-7000 characters that makes it more complex than the Ethiopic character set. Moreover, the complexity of Japanese/Chinese characters is expressed in terms of the large number of strokes used in a single character. The main difference between Ethiopic and Arabic characters is that Arabic characters are cursive in nature and Ethiopic characters are not.

From the lesson learnt from the trend of building online handwriting recognition systems for the three character sets (Latin, Japanese/Chinese, and Arabic), we have set the following guideline as to continue the development of the recognition system.

The structural approach seems to be more appropriate for Ethiopic character set as HMMs are good options for cases in which segmentation is a big problem and the number of character is less in number. Since Ethiopic characters are naturally written in discrete manner, segmentation would not be a big problem. Moreover, the number of characters is above 300, which is not a small number. From experience, we observed that different Ethiopic writers write similar characters with different stroke type, number and order. Thus, we underlined that the structural approach should not be based on prior identification of strokes in the Ethiopic characters. Rather the set of strokes must be obtained from the writer's style. This also makes the system to be writer-dependent and evidently increase the recognition accuracy.

CHAPTER FOUR

THE ETHIOPIC CHARACTERS

As the main concern of this work is the Ethiopic character set, in this chapter we discuss about the Ethiopic character set as a whole and the analysis done on the shape of the individual characters while conducting this research.

4.1. Introduction

People started to represent ideas in marked symbolic form long ago. Writing has been a means of communication and a persistent tool to represent information. Writing systems, in particular full writing systems, have been constructed for different languages for the purpose of expressing unambiguously any concept that can be formulated in the language[20]. Archaeological discoveries suggested that the Egyptian hieroglyphs might be the oldest form of writing which was active from about 3300 or 3200 BC[20]. Writing systems evolved from logogram systems in which complete words are represented by signs called logograms. To overcome the disadvantages of logogram systems such as their ambiguous nature of representing words, syllabic systems were developed which are concerned about representing sound by using signs. More improved systems are alphabetic systems in which consonant and vowels are separately written[20]. Alphabets are set of symbols or letters to represent sounds of a language so that words in the language could be written by combining the alphabets[21]. Alphabetic writing systems are the most common writing systems, though some writing systems like Chinese and Japanese still do not adopt them[21].

The Ge'ez script, which dates back to AD 300, is one of the native African writing systems [22]. The original Ge'ez script has only 26 consonants and has no vowel indications until around 350 A.D.[7]. But, later, vowels were incorporated by adding six non-basic characters for each consonant symbol. These were created by undergoing structural changes to the existing consonant symbols and this made the total number of characters in Ge'ez script 182[7]. Eventually, the Ge'ez script gave birth to the Amharic writing system,

which now is used to write Amharic and other Semitic languages like Tigre, Harari and Gurage in Ethiopia[23].

In addition to the 26 consonants and their six non-basic forms that exist in Ge'ez system, the Amharic writing system later added 7 other consonants whose shape derived from some of the existing Ge'ez characters[7] (See Table 4.1.). This increases the total number of characters to 231 (33×7) in the newly born Amharic character set.

Derived Character	Derived From
ሸ	ሰ
ቸ	ቀ
ኘ	ነ
ኸ	ሀ
ዠ	ዘ
ጀ	ደ
ፏ	ጠ

Table 4.1. Derived Characters in the Amharic Character Set

4.2. The Ethiopic Character Set

The total number of characters in the Amharic character set is 306 if the Ethiopian numerals (Table 4.3), punctuation symbols, and the extended character group (፩,፸ etc) are counted with the alphabetic characters. From these, 231+7 of them are alphabetic letters (See Table 4.2). Basically, the number of basic characters is 33 and the total number of alphabetic characters will be 231. But, the character ቨ is included as a special character to represent the sound 'v' from Latin-based languages and it has also six non-basic forms. That is the reason why the number of alphabetic characters is referred as 231+7. The Ethiopic numerals contain 20 symbols defined for each of the first 10 natural numbers (1 to 10) and for numbers that are multiples of 10 in the range 11-1000. Other numbers are written by combining these 20 distinct digits. For example, the number 25, is written as ፳፭, which is obtained by writing **20** (፳) and **5(**፭) side by side. But, these digit characters are not frequently used by individuals to write numbers. For this reason, the Ethiopic numerals are not considered in our recognition system. The other extended group of characters such as ፩,፸, and etc are also not considered in our work.

This leaves us with the remaining 231+7 alphabetic characters. The beauty of the Ethiopic character set, particularly the alphabetic letters, is not only in its appearance [23] but also

in its systematic organization. The characters are arranged in the form of table with 33+1 rows and 7 columns. The characters in the first column are called *basic characters* and they are 33+1 in number. The other six columns include the *non-basic characters* whose shape is derived from the basic characters. More importantly, the shape derivation is quite interesting for this research as character recognition is the main issue. We have analyzed the way the shapes for the non-basic characters are derived for the purpose of designing a better strategy or method of character recognition. The next section presents the analysis.

ሀ	ሁ	ሂ	ሃ	ሄ	ህ	ሆ
ለ	ሉ	ሊ	ላ	ሌ	ል	ሎ
ሐ	ሑ	ሒ	ሓ	ሔ	ሕ	ሖ
መ	ሙ	ሚ	ማ	ሜ	ም	ሞ
ሠ	ሡ	ሢ	ሣ	ሤ	ሥ	ሦ
ረ	ሩ	ሪ	ራ	ሬ	ር	ሮ
ሰ	ሱ	ሲ	ሳ	ሴ	ስ	ሶ
ሸ	ሹ	ሺ	ሻ	ሼ	ሽ	ሾ
ቀ	ቁ	ቂ	ቃ	ቄ	ቅ	ቆ
በ	ቡ	ቢ	ባ	ቤ	ብ	ቦ
ተ	ቱ	ቲ	ታ	ቴ	ት	ቶ
ቸ	ቹ	ቺ	ቻ	ቼ	ች	ቾ
ኀ	ኁ	ኂ	ኃ	ኄ	ኅ	ኆ
ነ	ኑ	ኒ	ና	ኔ	ን	ኖ
ኘ	ኙ	ኚ	ኛ	ኜ	ኝ	ኞ
አ	ኡ	ኢ	ኣ	ኤ	እ	ኦ
ከ	ኩ	ኪ	ካ	ኬ	ክ	ኮ
ኸ	ኹ	ኺ	ኻ	ኼ	ኽ	ኾ
ወ	ዉ	ዊ	ዋ	ዌ	ው	ዎ
ዐ	ዑ	ዒ	ዓ	ዔ	ዕ	ዖ
ዘ	ዙ	ዚ	ዛ	ዜ	ዝ	ዞ
ዠ	ዡ	ዢ	ዣ	ዤ	ዥ	ዦ
የ	ዩ	ዪ	ያ	ዬ	ይ	ዮ
ገ	ጉ	ጊ	ጋ	ጌ	ግ	ጎ
ደ	ዱ	ዲ	ዳ	ዴ	ድ	ዶ
ጀ	ጁ	ጂ	ጃ	ጄ	ጅ	ጆ
ጠ	ጡ	ጢ	ጣ	ጤ	ጥ	ጦ
ፈ	ፉ	ፊ	ፋ	ፌ	ፍ	ፎ
ፐ	ፑ	ፒ	ፓ	ፔ	ፕ	ፖ
ቨ	ቩ	ቪ	ቫ	ቬ	ቭ	ቮ

Table 4.2. The Amharic Core Character Set

፩	፪	፫	፬	፭	፮	፯	፰	፱	፲
1	2	3	4	5	6	7	8	9	10
፳	፴	፵	፶	፷	፸	፹	፺	፻	፼
20	30	40	50	60	70	80	90	100	1000

Table 4.3. The Ethiopic Numerals

4.3. Analysis of the Ethiopic Characters

The characters in the Ethiopic character set are broadly divided into two groups as basic characters and *non-basic characters*. The number of the basic characters is 33+1 and they are listed in the first column of Table 4.2 and are referred as first order characters afterwards. The characters in all the other six columns in Table 4.2 are non-basic characters and each of them are derived from their basic relatives. These characters are referred as second-order, third-order etc depending on the order that they appear in Table 4.2.

4.3.1. First Order (Basic) Characters

The basic characters are almost distinct in shape except some of them. In fact, the 26 original characters from the Ge'ez script differ much in shape. Even though it gives sense to say that few characters from this set are similar (such as ረ and ፈ), we prefer grouping the 26 original characters in one group and the remaining derived characters and the special character ፘ in another category as indicated below.

ሀ	ለ	ሐ	መ	ሠ	ረ	ሰ
ቀ	በ	ተ	ኀ	ነ	አ	ከ
ወ	ዐ	ዘ	የ	ደ	ገ	ጠ
ጰ	ጸ	ፀ	ፈ	ፐ		

Table 4.4. Group1 (Basic character)

ሸ	ቸ	ኘ	ኸ
ኰ	ጀ	ፏ	ቨ

Table 4.5. Group2 (Basic character)

4.3.2. Second Order Characters

The shape of these characters is generally formed by attaching a horizontal stroke to the middle of the right side of the basic characters. Examples are ሁ (from ሀ), ኈ (from ኀ), ዱ (from ደ), ጁ (from ጀ), ፁ (from ፀ), etc. The second-order characters are consistent in following this single rule. Exceptional to this rule are ሩ (derived from ረ) and ፉ (derived from ፈ).

4.3.3. Third Order Characters

According to our observation, two rules are mainly applied to derive the shape of the third order characters. These are:

Rule 1: If the basic characters have leg(s), then extend the right leg horizontally to the right. Examples are ሏ(from ለ), ሐ(from ሐ), ሸ(from ሸ), ገ(from ገ).

Rule 2: If the basic characters have no legs, attach a leg and extend it in the similar way stated in Rule 1.

Letters following this rule are ሃ(from ሀ), ማ (from መ), ሣ (from ሠ), ዋ (from ወ), ዓ (from ዐ), etc.

Only three characters from the third order character do not follow one of these rules. These are ሪ(from ረ), ዩ(from የ), ፊ(from ፈ).

4.3.4. Fourth Order Characters

It is possible for us to see that the fourth order characters are derived by applying the next rules:

Rule 1: If the base character has two or more legs, then shorten the left legs or all legs except the right most one. For instance, ላ(from ለ), ኣ (from አ), ዳ (from ደ), ጫ (from ፈጨ) are constructed using this rule.

Rule 2: For base characters with one leg, extend it to the left (or connect a horizontal stroke running to the left to the leg). Characters like ቃ (from ቀ), ታ (from ተ), ቻ (from ቸ), ፃ (from ፀ).

Rule 3: If the base character has no leg at all, reduce the size of the character by aligning to the upper corner and attach a long vertical stroke to the right side of the base character. Examples: ሃ (from ሀ), ማ (from መ), ዋ (from ወ), ያ (from የ).

However, the characters ራ, ና, ኛ, ፋ seem to follow none of the abovementioned rules and are considered exceptional in their group.

4.3.5. Fifth Order Characters

The fifth order characters mostly add loops to the base of the right legs of the characters. Particularly, the following rules are applied to construct them.

Rule 1: For base characters with two or more legs with the right leg free from loop, attach a loop to the right leg. Examples are ሉ (from ለ), ቱ (from ተ), ጤ (from ጠ).

Rule 2: If the basic characters have no legs, reduce the size of the character by aligning to the upper corner and attach a vertical stroke with a loop at its bottom end to the right of the base characters. For instance, ሂ (from ሀ), ሚ (from መ), ዊ (from ወ) are constructed this way.

Furthermore, the characters ዬ (from የ), ፌ (from ፈ) are derived by adding a horizontal stroke with a loop at its right end and ሬ (from ረ), ፌ (from ፈ) are formed by attaching a loop to the horizontally lying end of the base characters.

4.3.6. Sixth Order Characters

Constructing rules for the sixth and seventh order characters is difficult as they are highly irregular. Thus, we grouped the characters by putting the characters whose shapes are constructed in a similar way in the same group. The following table (Table 4.5) shows the categorization.

Group No.	Characters
Group 1	ሀ, ክ, ኽ, እ
Group 2	ል
Group 3	ሕ, ቅ, ኅ, ኾ, ጥ, ቍ, ኵ
Group 4	ብ
Group 5	ም, ሥ
Group 6	ር, ፍ
Group 7	ስ, ሽ, ዕ, ፅ
Group 8	ጎ, ን, ኝ
Group 9	ው
Group 10	ዝ, ዥ
Group 11	ይ
Group 12	ድ, ጅ, ጵ, ጽ
Group 13	ግ

Table 4.6. Grouping the Sixth order character

4.3.7. Seventh Order Characters

The majority of the seventh order characters are constructed by shortening the right one or two legs of the base characters with two or more legs. A few examples are ሶ, ቦ, ኮ, ዞ etc. For characters with no leg, a straight (like in the case of ሟ) or left-declined vertical line (as the case of ሦ, ዎ, ዩ, ዪ, ዧ) is connected to the bottom of the characters. Other seventh order characters such as ሆ, ኆ, ቆ, ኰ are formed by putting a loop at the top of the base characters. Further, it is observed that the letters ዏ, ዮ, ጎ, are unique in derivation of their

shape than the others. In general, it can be said that the seventh order characters follow different patterns and recognizers must consider these different patterns differently.

4.4. Conclusion

The shape and pattern formation of the Ethiopic characters is categorized and presented in the subsections above. Using this information as a background, we are able to come up with points of decisions discussed in the following paragraphs.

Though, some irregularities are seen in the non-basic characters particularly in the sixth and seventh order characters, it is generally possible to say that the shapes of non-basic characters is more or less similar to the basic ones. In other words, the non-basic characters add some minor changes to the shapes of the basic characters. This fact lead us to a strategic move to first consider the 33+1 basic characters whose shape is somehow distinct. Then, an online Ethiopic handwriting recognition engine that recognizes the basic characters can also easily recognize the associated non-basic characters. This strategy leads to the approach of taking the basic characters as fundamental and primary set of characters in a recognition system. This idea is strengthened by looking at the experiences of other systems developed for Ethiopic characters. Though not exactly the same, identical strategy was used for the construction of the manual Ethiopic/Amharic typewriter keyboard layout, the traditional Ethiopic text entry for modern computer keyboards and the phonetic Ethiopic text entry on computer keyboards. Then, the non-basic characters can be considered as a secondary set of characters that can be recognized based on the basic characters.

Ethiopic characters entering via keyboard in phonetic mode is one common example. Most systems assign each key for English consonant to a Ge'ez basic character of similar sound. In some cases, there will not be a single consonant for some Ethiopic characters with similar sound. For example, the sound of the character ቸ (read as '*che*') cannot be associated to any of the consonants in English. Handling these cases differ from system to system. In any case, all basic characters will be assigned to some consonant. The most interesting case is the case of non-basic characters. Non-basic characters are usually entered by using combination of keys, of which one is always the key assigned to their basic character and the other corresponds to a vowel, which could probably create their

sound. For example, if **k** is assigned to ක (read as '*ke*'), then කු (read as "*ku*") can be entered by pressing **k** and **u** together (k+u). This indicates that there is a habit of looking at the basic characters and non-basic characters in different views. Our approach is no different because we claim that it is also possible to create a two-level character entry for non- basic characters by providing an interface that looks like the one shown in Fig. 4.1.

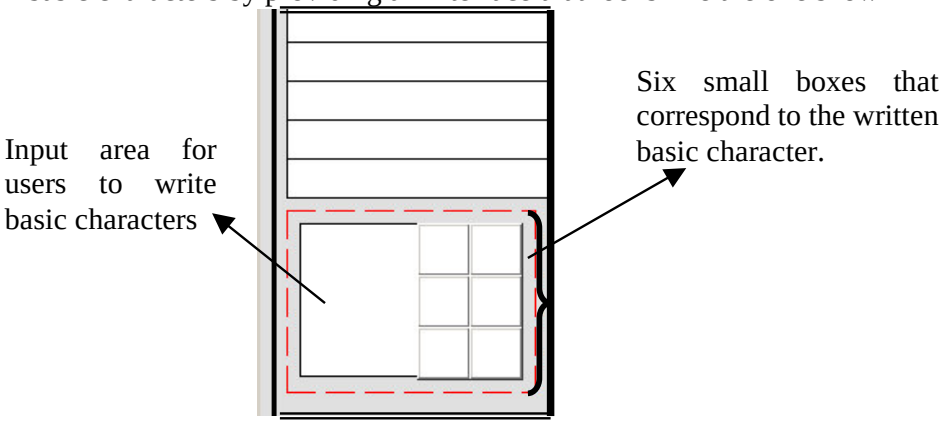

Fig. 4.1. A Two-level Character Entry Interface

Users may write basic characters in the larger box as shown in Fig. 4.1. If the users are interested to enter one of the non-basic characters, then they are expected to tap one of the six boxes that correspond to the non-basic characters. Thus, the recognition of non-basic characters will be based on the recognition of basic characters.

Another alternative to justify this strategy is by arguing that segmentation algorithms could extract the shape of basic characters and the additional strokes used to form the non-basic characters given the non-basic characters (See Fig. 4.2). This implies that an additional effort is needed only to recognize the additional strokes and their position to recognize the non-basic characters if the recognition of basic characters is done.

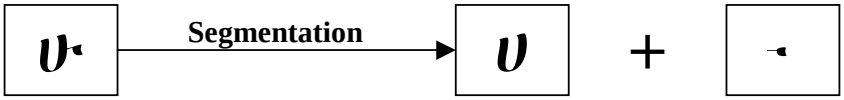

Fig. 4.2. Segmentation applied on කු (extract basic character and the stroke)

However, data entering via keyboard and using pen like devices should be handled differently despite the similarity of the goals of the two systems which is interpreting the input data to a character. In keyboard-based data entering scheme, once the keys are assigned to the characters, the remaining task will be displaying characters by examining the keys pressed. In online handwriting recognition system, a carefully designed algorithm should be employed to handle the variability.

CHAPTER FIVE

ONLINE HANDWRITING RECOGNITION SYSTEM FOR ETHIOPIC BASIC CHARACTERS

5.1. Introduction

The decreasing cost, the increasing capacity, the pervasive feature, and the increasing availability of applications are the major factors that initiated us to think that an Ethiopic online handwriting recognition system is a requirement of its usage in Ethiopia. The usage of increasing applications and usage of handheld computers of demands an online handwriting recognition system as one of the major input methods Taking this initiative, we started our work by investigating the existing systems developed for other characters such as Latin, Japanese/Chinese and Arabic. We have also studied the nature of the Ethiopic characters in relation to these character sets and their existing online handwriting recognition systems. This has given us the way to choose a general approach, and to design appropriate algorithms that could help to realize the development of online handwriting recognition systems for Ethiopic characters. The recognition system we designed is a writer-dependent system. The reason why we give priority for writer-dependent online handwriting recognition systems, is because the target devices usually belong to individuals and we believe that attaining higher recognition accuracy is more important than writer-independency.

Due to the variation in people's handwriting styles, it is obviously true that it is a challenging task to develop handwriting recognition systems. On top of this, other external reasons made our research difficult. One of the main reasons is that no online handwriting data has ever been collected and standardized. This restricted us to use a limited amount of data to test our system for its usability. However, the system was tested for highly varying handwriting styles and we strongly believe that it works for more number of writers.

Of the many and diversified approaches used in online handwriting recognition systems that are discussed in chapter 2, we have chosen the structural approach for our recognition

engine. The structural approach is concerned about the various primitives that make up a character and their relationship. It is the most natural way of treating characters, however the approach has a disadvantage, which is the difficulty to extract the primitives and their relationship from a given character [1].

5.2. Structural Approach for Recognition of Ethiopic Characters

In spite of the fact that structural approach is usually criticized and regarded as an older technique, it has got positive aspects, which would compensate its weak points. Some of these are:

- If proper algorithms are developed for primitive extraction, this approach is the most natural way of defining characters.
- It does not require huge amount of data to train the system [3].

5.3. Appropriateness of the Approach

The nature of Ethiopic characters itself guided us to use the structural approach where we design a simple but a robust algorithm to represent primitives. Most of the Ethiopic characters have defined shape. The set of the alphabetic characters in the Ethiopic character set contains 33+1 basic characters and six non-basic characters with each basic character. The shapes of these non-basic characters are derived from the related basic character by adding a secondary stroke or by deforming their shape. This structural relationship is a great advantage. After doing the analysis described in chapter four of this document, we have concluded that if the recognition of the 33+1 basic characters is done successfully, then the system could be easily extendible to recognize the remaining non-basic characters.

5.4. Design of Online Handwriting Recognition System for the Basic Ethiopic Characters

The design of the online Ethiopic handwriting recognition engine developed in this work is shown below (Fig. 5.0). The unknown pattern in general is acquired through the data points collection process. This is achieved by using the digitizer software, *MovAlyzer*. The collected data is brought to the preprocessing package to eliminate variations. The preprocessed data is provided to the feature extraction module, which will produce the observation code sequences from the given handwriting pattern. This code could be used to train or test the system as required. In this case, training is the process of teaching the recognition engine to make it aware which observation code sequences belong to which strokes or characters. While testing, unknown handwriting recognition patterns represented by observation code sequences are presented to the recognition engine and the recognizer is expected to recognize what character is represented by the given observation code sequences.

No explicitly defined segmentation is stated. The reason is that we have considered only segmentation of characters into strokes which are already separated by pen-up points in the collected online handwriting data. However, implicit segmentation is done in each step of the online handwriting recognition system by looking for pen-up points to find the beginning and end of strokes.

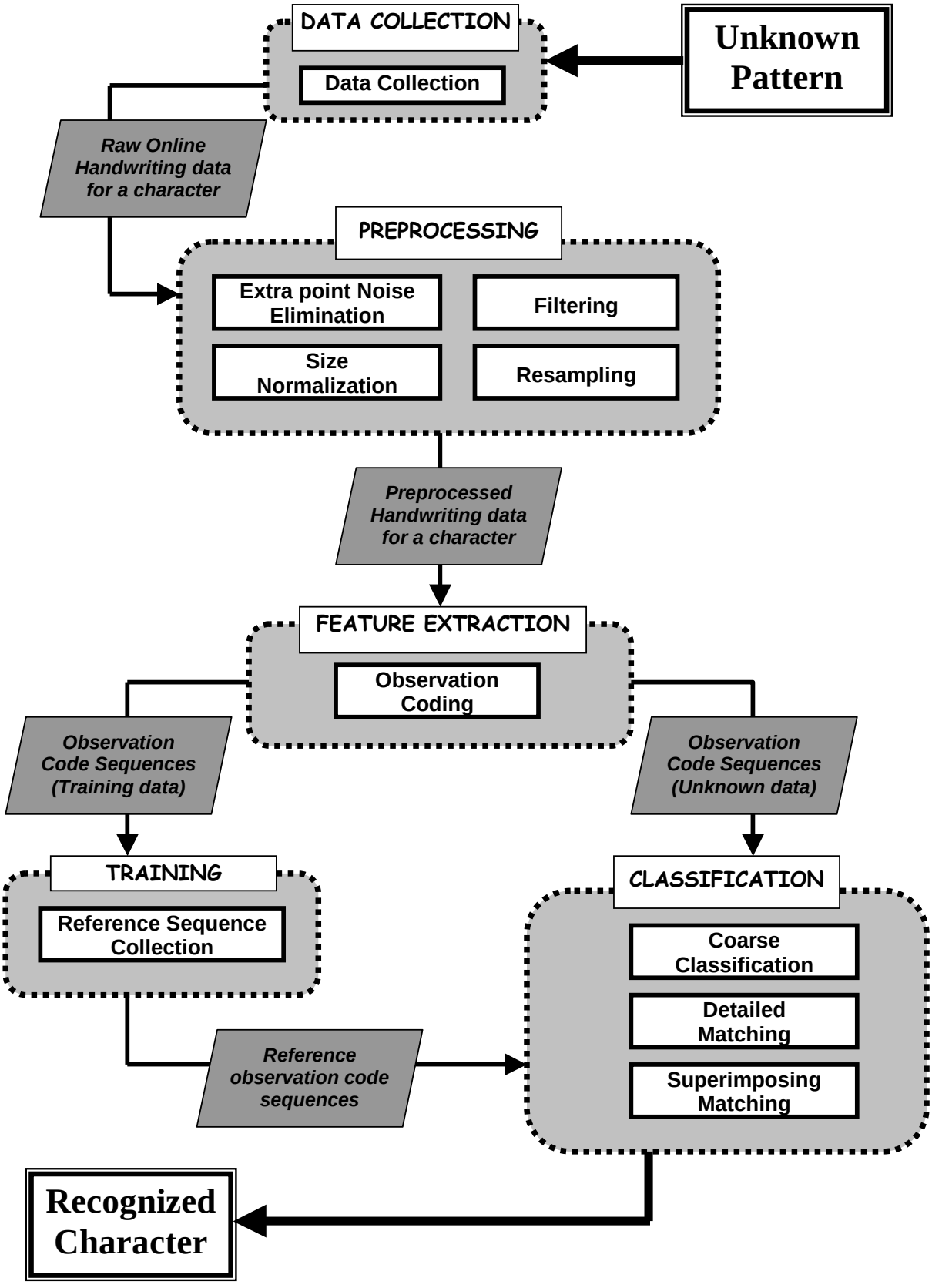

Fig. 5.0. Design for online handwriting
Recognition system for Ethiopic characters

5.4.1. Data Collection

Under normal circumstances, online handwriting data for this kind of research is collected using tablet digitizers. However, for the purpose of our experiment, handwriting data is collected using digitizer software named *Neuroscript MovAlyzer* that is developed to sample points on the trajectory of the mouse or an electronic pen (digitizer) if any. When digitizer is used to collect data, the common sampling rate is 100-200Hz and it is 50-60Hz when the mouse is employed. The capability to use the mouse as an input device to draw characters enables us to collect data using mouse in place of an electronic pen.

This software includes many features including segmentation and word extraction. However, we used only the most important feature of the software, which is collecting online handwriting data using mouse. Whenever data is needed to be collected, MovAlyzer is started and an experiment is run. Then, *MovAlyzer* presents an interface (Fig. 5.1) that is used to write characters by dragging the mouse over the mouse pad.

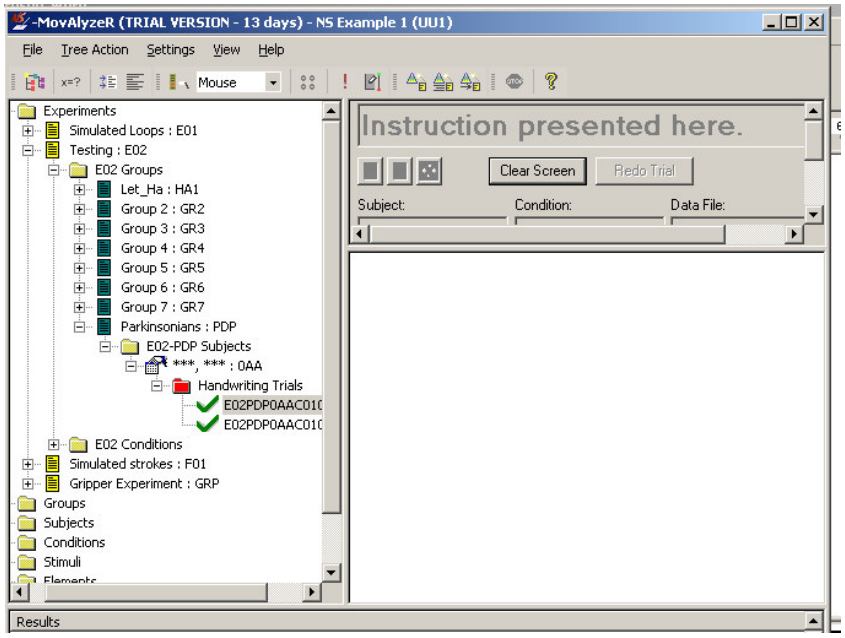

Fig. 5.1. MovAlyzer Recording Window

The writer then starts writing anywhere on the given blank screen and the system records points along the path of the drawing. For a single sampled point, MovAlyzer records three values, which are the x coordinate, y coordinate and the pen-pressure. The pen pressure is equivalent to the pen-up or pen-down information where it has only two possible values while using the mouse as an input device. These values are 99 and 0 which correspond to

pen-down and pen-up respectively. If a digitizer/tablet were the input device, the pen pressure would vary to reflect the amount of pressure applied by the pen.

The difficulty during data collection is the inappropriateness of the mouse to write. In spite of the fact that the mouse is simulating the light pen, it can by no means replace the pen and is unnatural. Due to this reason, the data could be noisy and the trajectory is more or less jagged. Related to this, writers had also difficulty in writing the same character consistently using a mouse. The good thing is the system is supposed to be writer-dependent and there is no need to collect a large amount of data to handle varieties. But, it was tried to test the system for two writers whose handwriting vary much. These writers are referred as writer A and writer B from now on.

Fig. 5.2. illustrates the appearance of the data recording window after a character is written. Electronic ink marks all the pen-down sampled points and the pen-up points are not visible though they are recorded. This capability makes the environment natural and convenient for writers.

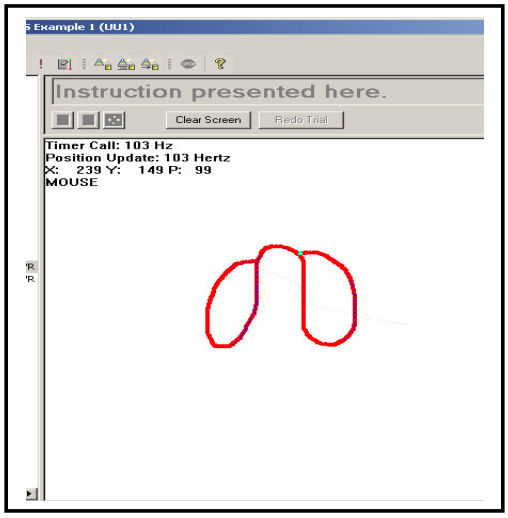

Fig. 5.2. MovAlyzer Character Writing pad (after the character "ØÞ" is written)

5.4.2. Data Preparation

As stated earlier, MovAlyzer produces the raw (x,y) coordinates with a pen up/pen down information of a data point. A data point is a point that is sampled along the path of the mouse while the writer is writing the character. Thus, for a single character, number of data

points would be sampled. The software then presents the set of data points for a drawn character collectively in a text file format.

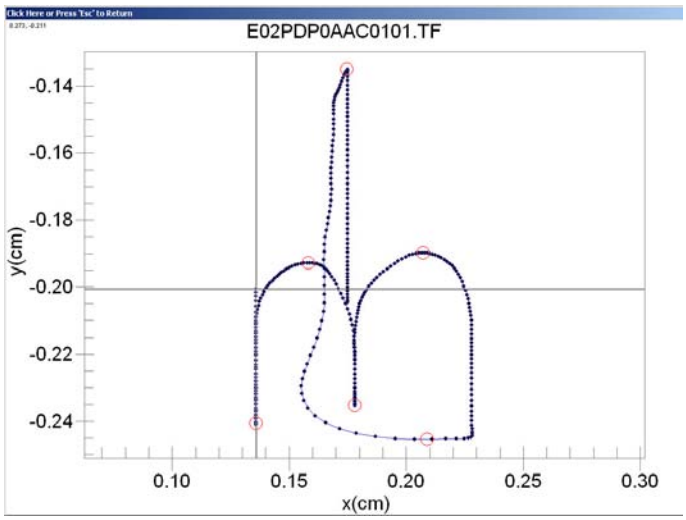

Fig. 5.3a) The Character '𝒽' as it is written on the MovAlyzer writing pad

136	-200	99
136	-201	99
136	-203	99
136	-204	99
136	-206	99
136	-208	99
136	-209	99
136	-211	99
136	-212	99
136	-214	99
136	-215	99
136	-216	99
136	-218	99
136	-220	99

Fig. 5.3b) The set of data points during the formation of "𝒽' the character

Fig. 5.3a shows the character written by writer A and part of the data points collected for this character is shown in Fig. 5.3b. For this character, 354 points are sampled by MovAlyzer and each are recorded in terms of the x and y coordinate values and the pen (mouse) pressure. This set of data points is saved in a text file. This is the original raw data and used as an input to the recognition engine.

Even though online handwriting recognition systems are known to be real-time, the online handwriting data collected in such researches is stored in a persistent way to carry out repetitive experiments on the data during the development of the recognition engine. Following the same trend, the sampled data points set collected for each character are kept in a text file being accessible to the recognition engine.

The collected data should be grouped as training data and testing data. The training data is supplied to the recognition engine while it is being trained and the test data serve to test the system for its accuracy. The way the training is conducted as well as the categorization of the data is explained in a subsequent section.

5.4.3. Preprocessing

The preprocessing steps play a very significant role to improve recognition accuracy by avoiding the inter-character variation, that is variation that occurs between different occurrences of the same character. In this system, we have found that the following preprocessing stages are significantly necessary and are implemented.

5.4.3.1. Extra Pen Up Data Points Noise Elimination

At this stage, we eliminate only one type of noise that is caused by additional pen-up data points recorded after the character is completely drawn. These points are not necessary and more importantly create a series problem during the feature extraction process. Thus we implemented the algorithm in Fig. 5.4 to eliminate them.

```
Input: File that contains the set of data points sampled for a character
        //index: points to each data point
        //NoOfPts: number of pen up and pen down data points collected for a given character
        //OrigX, OrigY, OrigZ: set of x, y and z components of data points
        //LastIndex: points to the last data point

        index ←1
        NoOfPts ← 0
        Do
              Read x, y, z of a data point from file
              OrigX [index] ← x
              OrigY [index] ← y
              OrigZ [index] ← z
              index ← index + 1
              NoOfPts ← NoOfPts + 1
        Until (end of file is reached)
        LastIndex ← NoOfPts              //Keep the last value of NoOfPts in LastIndex
        While (OrigZ[LastIndex] = 0)
           LastIndex ← LastIndex – 1     //Decrease LastIndex if the data point(s) is a
                                           pen up.
        index = 0
        While (index < LastIndex)
        Do
              Write Origx [index] to file
              Write  Origy [index] to file
              Write Origz [index] to file
              index ← index+1
        Until (index < LastIndex)        //Unwanted data points are dropped
```

Fig. 5.4. Extra Data point Noise Elimination Algorithm

5.4.3.2. Size Normalization

The digitizer software *MovAlyzer* used for the data collection process presents a window, which is evidently large enough to write lines of sentences. However, the writers are told to write only a single character in an occasion for the reason that we are concerned about character recognition and data of a single character is an input to the system. This made the users free to write anywhere on the writing pad in any size.

This situation necessitates the application of size normalization to map all characters to a box of standard size (40×60). It is observed that this size preserved the proportion between the length and width of most of the characters hence do not alter their shape much. In spite of this, few characters became abnormally longer in proportion to their width (See Fig. 5.5a and Fig. 5.5b) after size normalization has been applied. But we didn't do anything about it because it doesn't create any problem since our classification approach depends on direction features.

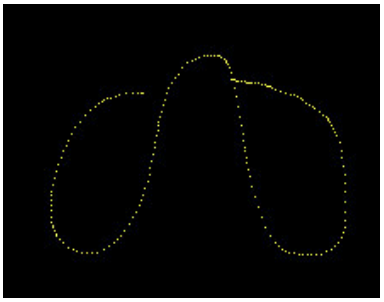

Fig. 5.5a. The letter "ಅ" with its original size (173×116)

Fig. 5.5b. The same letter "ಅ" after size normalization (40×60).

Size normalization is done by translating each data point of a character to a box of a standard size. This is accomplished by first finding the box that encloses the character with its original size and then computing the *width* and *length* of the box. These values are later used to find (x,y) coordinate values of newly produced data points in the standard sized box that correspond to each data point from the original box. The complete algorithm is presented below (Fig. 5.6).

Input: File that contains the set of data points sampled for a character (noise eliminated)

//*MaxX, MinX*: Maximum and minimum x values among the data points sampled for a character

//*MaxY, MinY*: Maximum and minimum y values among the data points sampled for a character

```
//Find the block that encloses the character.
    Compute MaxX,  MinX
    Compute MaxY, MinY
    Width ← MaxX-MinX       //width of the box that encloses the character
    Length ← MaxY-MinY      //Length of the box that encloses the character

//Normalize character stroke-by-stroke
    Do
        Read x, y, z of a data point from file
        if (z = 0)                  //pen-up data point
            Write  the triple 0, 0, 0           //Stroke separation (segmentation)
            do
                Read  x, y, z of the next data point        //To skip all other pen up points
             Until (z ≠ 0)
        else                        //pen-up data point
            //Translate the data point to a box of size 40 × 60
            newx ← (40 × x) / width
            newy ← (60 × y) / length
            Write newx, newy, z  in a new file
    Until (end of file is reached)
```

Fig. 5.6. Size Normalization Algorithm

This method results in having an equal number of pen-down data points, but replace consecutive pen-up data points by the triple (0, 0, 0) to indicate separation of strokes. Note that pen-up data points are recorded when the writer lift up the pen (mouse) to draw an additional stroke in the character. This is clearly seen in Fig. 5.3a. It is also observed that some data points are translated to the same point in the newly created point set. This creates redundant data and further preprocessing, filtering is essentially implemented. For instance, part of the collected data points for letter "ಅ" shown in Fig.5.5a and the translated data points in its normalized version which is also shown in Fig. 5.5b are provided below (Fig. 5.7a and Fig. 5.7b).

220	-172	99
219	-172	99
217	-172	99
214	-172	99
210	-172	99
206	-173	99
204	-174	99
201	-175	99
199	-175	99
197	-176	99
196	-177	99
194	**-178**	**99**
192	-180	99
189	-182	99
186	-185	99
183	-189	99
	•	
	•	
	•	

Fig. 5.7a. Original data for letter "ω"

50	-88	99
50	-88	99
50	-88	99
49	-88	99
48	-88	99
47	-88	99
47	-89	99
46	-89	99
46	-89	99
45	-90	99
45	-90	99
44	-91	99
44	-92	99
43	-93	99
43	-94	99
42	-96	99
	•	
	•	
	•	

Fig. 5.7b. Data points after normalization applied to the Original data shown in Fig. 5.7a

From Fig. 5.7a and Fig. 5.7b, it is possible to see that the first three data points in the original data set are mapped to the same data point which is (50, -88, 99). The preprocessing step discussed in the next section avoids such points from the normalized data.

5.4.3.3. Filtering

Filtering is another preprocessing technique that we have considered in this recognition engine. It is the process of avoiding redundant data points that appear consecutively. These data points are undesirable and have no use at all. Thus, eliminating them is essential. Moreover, the action also contributes towards having a reduced data, which is one goal of preprocessing.

Filtering the normalized data is accomplished by implementing the algorithm in Fig. 5.8 .

```
Input: File that contains size normalized data for a character

    Read x, y, z of the first data point from file
    PreviousX ← x
    PreviousY ← y
    Do
        Get x, y, z of next data point
        if (x = PreviousX and y = PreviousY)
            do nothing
        else
            Write x, y, z to a file
        PreviousX ← x
        Previousy ← y
    Until (end of file is reached)
```

Fig. 5.8. Filtering (Data Reduction) Algorithm

5.4.3.4. Resampling

Relatively, the handwriting character data is to some extent normalized and reduced by the previous two steps. But, the data points are not yet time normalized or equally spaced. Furthermore, the trajectory is still jittered due to the improper arrangement of the data points. This is corrected by the resampling procedure presented in Fig. 5.9.

```
Input: File that contains size normalized and filtered data for a character
    Read x, y, z of the first data point from file
    PreviousX ← x
    PreviousY ← y
    Write x, y, z to file                    //Accept the first point
    Do
        Read x, y, z of the next data point from file
        //Compute d, distance between the previous data point and the current data point
        d ← (x-PreviousX)^2 + (y-PreviousY)^2   .
        if (d ≥ 25)                          //25 is a threshold value
            Write x, y, z to a file          //Accept the point
            PreviousX ← x
            PreviousY ← y
        else
            do nothing
    Until (end of file is reached)
```

Fig. 5.9. Resampling Algorithm

The algorithm computes the distance between two consecutive data points and avoids one of them (the second one) if the distance is less than a threshold value. After trying many threshold values, it is concluded that 25 is a good threshold value. However, the sampled

data points will not be exactly equally spaced since points with distance greater than 25 are accepted. This situation occurs rarely and if it occurs, the distance will not be much greater than 25. Thus, the algorithm is left as it is. Fig. 5.10a and Fig. 5.10b show the appearance of the letter "ಯ", that we have considered in Fig. 5.5a and Fig. 5.5b, before resampling and after resampling.

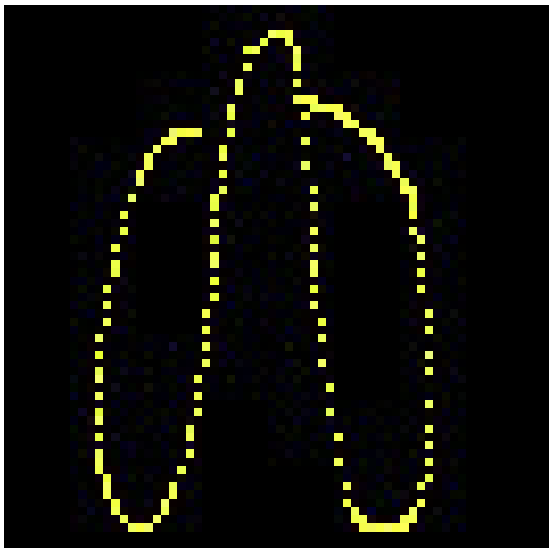

Fig. 5.10a. Magnified view of "ಯ" after size normalization and before resampling

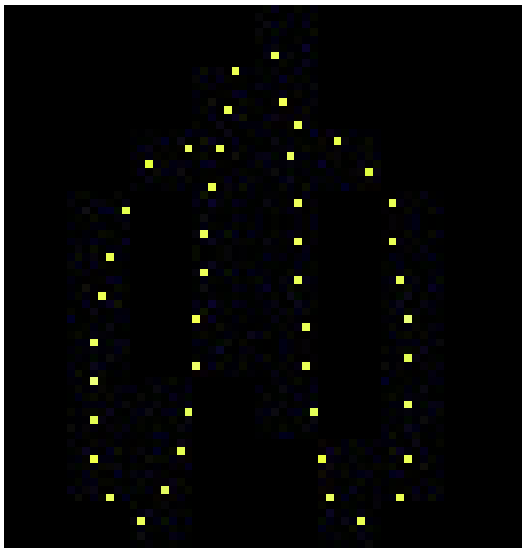

Fig. 5.10b. Magnified view of "ಯ" after resampling

After resampling, the data is significantly reduced and the irregularly placed data points that create the jitter on the trajectory are somehow removed. This makes the resampling step very useful in noise elimination as well as data reduction. The number of the original sample points in the raw data set for the character "ಯ", for example, is 189. By chance, the data for this character do not include unnecessary pen-up points which are sampled after the character is written completely and hence no change has been exhibited in the original data after applying the noise elimination procedure discussed in section 5.6.1.1. After size normalization, the number of points remains the same since the character is written by a single stroke. Note that the size normalization step plays its own role in reducing the data by replacing the pen-up points between strokes by a single (0,0,0). Filtering which avoids repeated data points reduced the number of data points to 160. Finally, the resampling action greatly reduced the data size to be only 45 data points.

5.4.4. Feature Extraction

The original data has passed through the preprocessing steps and refined to an extent described in the previous sections. But this doesn't mean that the data points for different instances of the same character become similar or identical. The following figures (Fig. 5.11a and Fig.5.11b) illustrate the preprocessed data set for two instances of the same character "ಕಿ".

91	-108	99
100	-109	99
107	-110	99
114	-110	99
120	-110	99
125	-110	99
130	-110	99
0	0	0
114	-112	99
114	-117	99
114	-125	99
114	-132	99
114	-137	99
114	-142	99
114	-147	99
114	-152	99
114	-158	99
114	-163	99
113	-168	99
0	0	0
97	-136	99
103	-135	99
108	-135	99
113	-135	99
118	-135	99
123	-134	99
128	-134	99

Fig. 5.11a. Preprocessed data for the letter "ಕಿ" (instance 1)

72	-79	99
77	-80	99
82	-82	99
88	-82	99
94	-83	99
99	-84	99
104	-85	99
109	-85	99
0	0	0
90	-82	99
90	-87	99
90	-93	99
90	-98	99
89	-103	99
89	-108	99
89	-113	99
89	-119	99
89	-124	99
89	-129	99
89	-134	99
89	-139	99
0	0	0
77	-110	99
82	-110	99
87	-111	99
92	-111	99
97	-111	99
102	-111	99
107	-111	99

Fig. 5.11b. Preprocessed data for the letter "ಕಿ" (instance 2)

Three strokes are used by the writer (Writer A) to write this letter as it is indicated by the triple (0,0,0) that separate the various strokes. As can be seen in the data collected from Writer A, she wrote many other letters such as "ಹಿ" with three strokes. Thus, the number of strokes alone cannot be a distinguishing factor to measure the similarity between two or

more characters. This implies that there is a need to examine and compare the corresponding strokes and identify their order of occurrence to quantify the similarity between characters.

In any classification task, an unknown pattern must be compared with the existing reference patterns to pick one, which is believed to be the most similar. Comparing patterns in their original form is error prone and computationally complex. Therefore, formulating pattern representation method that reflect similarity between identical characters and difference between characters from different classes is a crucial matter to make the comparison efficient and effective.

We have observed that the similarity between strokes could be measured by looking how the x values and y values of the data points change as moving from one data point to the next one. We have identified that the change can be expressed by three possible conditions. These are *increasing*, *decreasing* and *constant* (or nearly constant). Despite the variations, this property will be more or less maintained between strokes that involve in different instances of the same letter. This idea is illustrated by the samples given in Fig. 5.11a and 5.11b. The first stroke in the instance 1 and instance 2 of the letter "ঈ" have seven and eight sampled data points. The x values of the data points in the strokes that appear in both instances increase as one pass from one data point to another until the last data point. The y values in the data points generally do not change much and it is possible to conclude that the y values are not changing (nearly constant). A similar analysis could be done to the other strokes and each of them would be compared to the prototype strokes on this basis. The order of the strokes occurrence will also be taken into account to determine what character they form when combined. But, the classification result from this approach shouldn't be taken for granted, as it doesn't consider the relationship between strokes. For example, the characters in Fig. 5.12a and Fig. 5.12b are often confused by this approach.

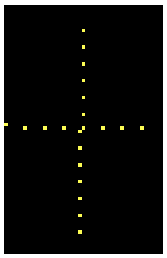

Fig. 5.12a. Letter "ঈ" (Writer B)

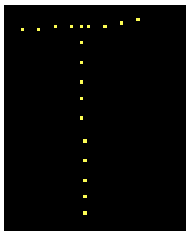

Fig. 5.12b. Letter "ঈ" (Writer B)

The reason is because writer B wrote the two characters by combining two similar strokes in the same order. In both cases, she drew the vertical line and then drew the horizontal line as a secondary stroke. The second stroke crosses the vertical line in letter "ধ" but is placed at the top of the vertical line in letter "*T*". By following the approach explained previously, there is no way to identify this distinction.

Therefore, examining the consecutive data points to see whether their x and y values increase, decrease or remain constant is a good approach to derive a common representation for the strokes, while more features need to be considered to refine the classification result of the character. Note that similar strokes are commonly found in different characters in the same order or different order. Thus, we have used this feature only for coarse classification, which produces a number of characters with higher degree of similarity with the unknown pattern from the whole prototype set. After getting the 'Top characters', detailed matching is performed to pick the finally classified character.

The next section presents the way to derive a common representation method for the strokes involved in character patterns to make them ready for comparison and hence for coarse classification. We called this process *observation coding*. Then, the technique of training and the classification of characters are presented in subsequent sections.

5.4.4.1. Pattern Representation /Observation Coding

The three possibilities that occur between x and y values of consecutive data points are encoded as shown in Table 5.1 below.

Condition	Number Code
Beginning of a new stroke	**1**
Increasing	**2**
Decreasing	**3**
Constant (Nearly constant)	**4**
Stroke separator	**5**

Table 5.1. Codification of observation

All handwriting patterns will be represented by sequences of these number codes appearing according to what happens between the x and y values of a pair of consecutive data points. Two observation code sequences are created for a single stroke representing the way the x and y values vary or remain constant. While extracting the sequences, a number code will not be assigned to each individual pair of data points. If series of data points exhibit the same characteristics in the nature how their x or y values vary, a single number code is enough to represent them. The algorithm to extract the sequences is shown in Fig. 5.13. This algorithm is particularly for obtaining an observation sequence code to represent the characteristics of the x values. The same algorithm is also applied to extract observation code sequence for the y values.

The algorithm in Fig. 5.13 starts by assigning **1** as the first element of the code sequence. This is to indicate the beginning of the first stroke (See Table 5.1). Then, the first data point is considered and its x value is kept. The next procedure is to pass to the next data point and compare its x value with the previously kept x value to determine whether a value that is less than, greater than or equal (nearly equal) is encountered. Accordingly, the values **2**, **3**, or **4** are assigned to indicate the situations increasing, decreasing or constant (nearly constant) respectively. If two x values in consecutive data points are different only by the value 1, they will be regarded as if they are the same and as a result an observation code **4** (for nearly constant) is recorded. In each step, if a formerly recorded observation code and the observation code that is currently to be assigned are the same, the assignment will be canceled to avoid repetitive observation codes in the sequence. This means that for set of x values having a uniform way of changing, only a single observation code represents the whole set. However, if this were the only task of the algorithm, it would have been impossible to properly extract observation code sequences from characters having more than one stroke. Thus, additional observation codes **1** and **5** are included to show the beginning of a stroke and as a separation mark between strokes respectively. That is the reason why x is checked if it is 0 (to identify whether the stroke ends and the triple (0,0,0) is came across). The value of PreviousX is also checked if in case it is 0. If it is 0, it implies that the current data point is the first point in the new stroke. The algorithm handles these two conditions by placing the values **1** and **5** in the sequence respectively.

Input: File that contains preprocessed data for a character
//CntObsX represents the number of extracted code sequences and ObsX is the code sequence itself.

```
CntObsX ← 1
ObsX [CntObsX] ← 1                    //Beginning of the first stroke
Get x value of the first data point from file
PreviousX ← x
Do
    Get x value of the next data point
    if (x = 0)                        //It indicates that the triple (0,0,0) is encountered
        CntObsX ← CntObsX+1
        ObsX[CntObsX] ← 5             //separation of strokes
    else if (PreviousX = 0)           //starting a new stroke
        CntObsX ← CntObsX+1
        ObsX[CntObsX] ← 1             //Beginning of a new stroke
    else
        if (x != PreviousX and |x-PreviousX|>1)
                if (x > PreviousX and Obsx[CntObsX] != 2)
                    CntObsX ← CntObsX+1
                    ObsX[CntObsX] ← 2          // Increasing
                else if (x < PreviousX and Obsx[CntObsX] != 3)
                    CntObsX ← CntObsX+1
                    ObsX[CntObsX] ← 3          //Decreasing
        else
                if (Obsx[CntObsX] != 4)
                    CntObsX ← CntObsX+1
                    ObsX[CntObsX] ← 4          //Constant or nearly constant
    PreviousX = x
Until (end of file is reached)
```

Fig. 5.13. Observation Code Sequence Extraction Algorithm

In Fig. 5.14a-5.14d, two instances of the letter "***U***" (written by Writer A) and the preprocessed data for the letters are shown. The data points are sequentially numbered to facilitate the discussion. The observation code sequences derived (from the two preprocessed data set) for x and y values are shown in Fig. 5.14e-5.14h. The sequences in Fig. 5.14e and 5.14f are analyzed in the following paragraphs.

Both sequences start with the observation code **1**. The x values among the first few data points (data point 1 – data point 9) are constant (or nearly constant) and hence observation code **4** is used to represent them in the x sequence (See Fig. 5.14e). In the next few data points (data point 10 – data point 18), x starts to increase. This makes the sequence extractor to record the observation code **2**. The x values in the last six points are constant or nearly constant and hence the observation code **4** represents them in the observation code sequence.

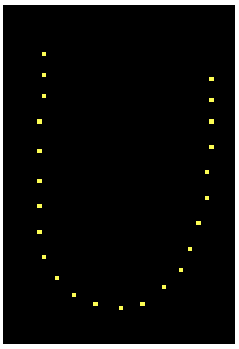

Fig. 5.14a. The letter υ (instance 1)

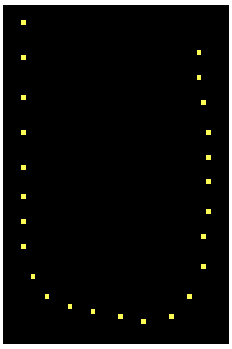

Fig. 5.14b. The letter υ (instance 2)

1.	102	-51	99
2.	102	-56	99
3.	102	-61	99
4.	101	-67	99
5.	101	-74	99
6.	101	-81	99
7.	101	-87	99
8.	101	-93	99
9.	102	-99	99
10.	105	-104	99
11.	109	-108	99
12.	114	-110	99
13.	120	-111	99
14.	125	-110	99
15.	130	-106	99
16.	134	-102	99
17.	136	-97	99
18.	138	-91	99
19.	140	-85	99
20.	140	-79	99
21.	141	-73	99
22.	141	-67	99
23.	141	-62	99
24.	141	-57	99

Fig. 5.14c. The preprocessed data for the letter υ in Fig. 5.14a

1.	42	-51	99
2.	42	-58	99
3.	42	-66	99
4.	42	-73	99
5.	42	-80	99
6.	42	-86	99
7.	42	-91	99
8.	42	-96	99
9.	44	-102	99
10.	47	-106	99
11.	52	-108	99
12.	57	-109	99
13.	63	-110	99
14.	68	-111	99
15.	74	-110	99
16.	78	-106	99
17.	81	-100	99
18.	81	-94	99
19.	82	-89	99
20.	82	-83	99
21.	82	-78	99
22.	82	-73	99
23.	81	-67	99
24.	80	-62	99
25.	80	-57	99

Fig. 5.14d. The preprocessed data for the letter υ in Fig. 5.14b

1
4
2
4

Fig. 5.14e. X observation code sequence for the data in Fig. 5.14c

1
3
4
2

Fig. 5.14f. Y observation code sequence for the data in Fig. 5.14c

1
4
2
4

Fig. 5.14g. X observation code sequence for the data in Fig. 5.14d

1
3
4
2

Fig. 5.14h. Y observation code sequence for the data in Fig. 5.14d

The y values between the data points in the set (Fig. 5.14c) can be analyzed similarly to derive the sequence shown in Fig. 5.14f. In the y observation code sequence, **3** represent the characteristics of the y values in the first 11 data points. The y values in the next three data points are almost the same and the observation code **4** is recorded. The rest of the data points contain y values, which increase from time to time and hence the last observation code **2**.

Identical sequences are being extracted from the second data set as illustrated in Fig. 5.14g and 5.14h. This example has shown the possibility of coming up with identical representations for different data sets. In general, these two sequences *together* characterize these patterns and other similar patterns irrespective of the actual values of the coordinates. Note that letter ***U*** is written by a single stroke (as Writer A wrote) and the observation code sequences do not include code **5**, which is a stroke separator code.

Fig.5.15 demonstrates how the extractor algorithm treats characters with multiple strokes.

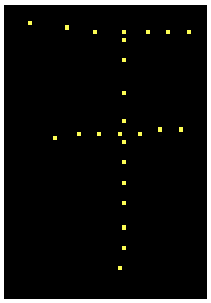

Fig. 5.15a. The letter "ः" (Writer A)

1.	91	-108	99
2.	100	-109	99
3.	107	-110	99
4.	114	-110	99
5.	120	-110	99
6.	125	-110	99
7.	130	-110	99
	0	0	0
1.	114	-112	99
2.	114	-117	99
3.	114	-125	99
4.	114	-132	99
5.	114	-137	99
6.	114	-142	99
7.	114	-147	99
8.	114	-152	99
9.	114	-158	99
10.	114	-163	99
11.	113	-168	99
	0	0	0
1.	97	-136	99
2.	103	-135	99
3.	108	-135	99
4.	113	-135	99
5.	118	-135	99
6.	123	-134	99
7.	128	-134	99

Fig. 5.15b. The preprocessed data for letter "ः" (Writer A)

1
2
5
1
4
5
1
2

Fig. 5.15c. X observation code sequence for the data in Fig. 5.15b

1
4
5
1
3
5
1
4

Fig. 5.15d. Y observation code sequence for the data in Fig. 5.15b

In general, the algorithm in Fig.5.13 produces identical, if not similar sequences for handwriting patterns that belong to the same character. However, it has been observed that the algorithm in its original form produced very diversified sequences for characters with rough trajectories. It is also very sensitive for slight variations that occur between instances of the same character. The following sequences (Fig. 5.16) are *X observation sequences* which are produced for different instances of the letter "ధ".

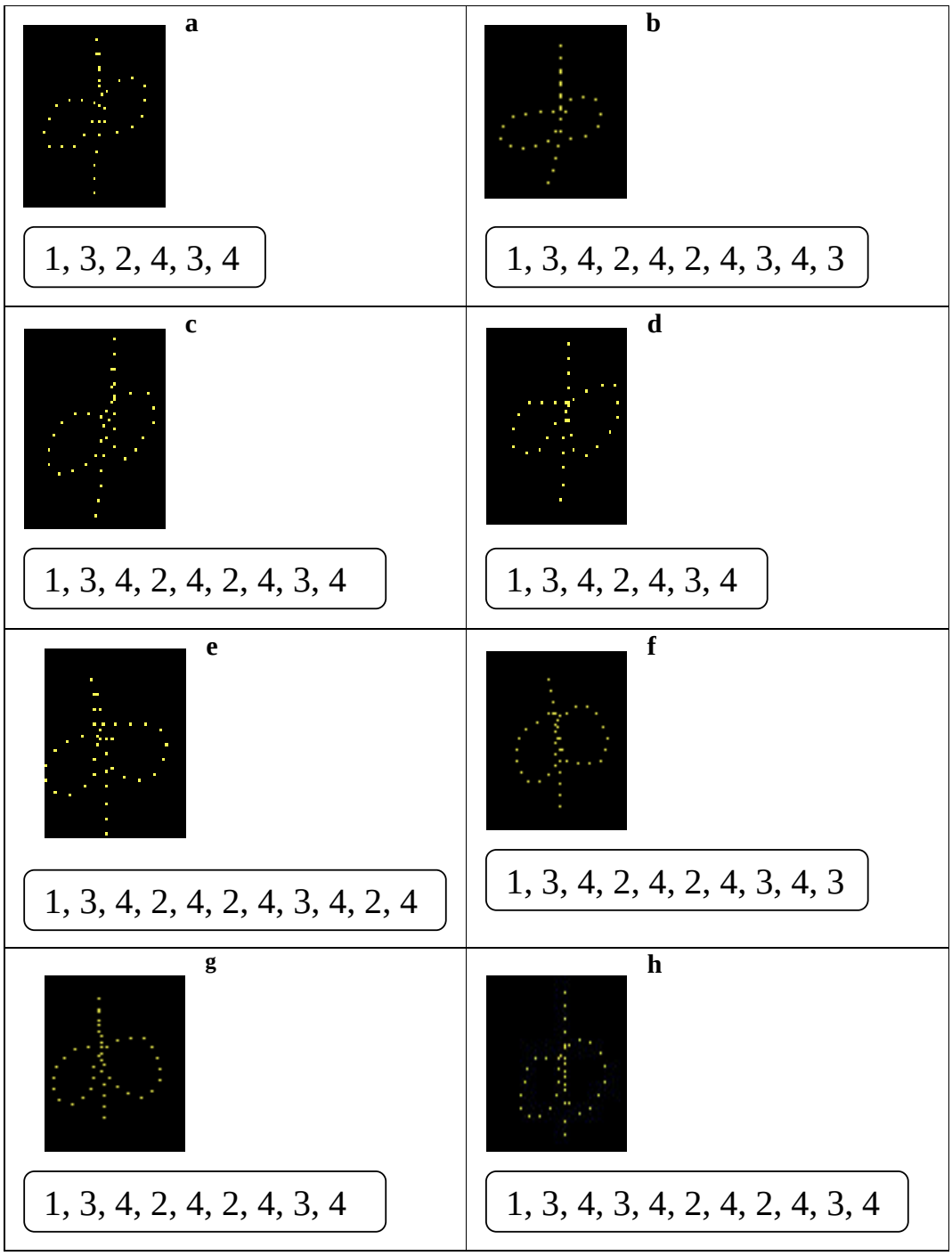

Fig. 5.16. X Observation Code Sequence for different instances of "ధ" (Writer A)

The eight sequences are produced for instances of the character "***Ф***" which are written by the same writer (Writer A). Contrary to the expectation, these sequences are different. This is mainly due to the shape variations. To overcome this problem, the algorithm (Fig. 5.13) is modified to reject observation codes, which last for too short time. To accomplish this, it is required to keep the *length* of each observation code in the sequence, which refers to the number of data points that cause the observation code to appear in the sequence. The length of an observation code denoted by Length(ObsvCode) is computed based on the following rule:

$$\textbf{Length (ObsvCode)} = \begin{cases} \textbf{0} & \textit{if ObsvCode = 1 or 5} \\ \textbf{Number of data points that cause the code to appear} & \textit{if ObsvCode = 2, 3 or 4} \end{cases}$$

where Length(ObsvCode) is the length of the observation code,
ObsvCode is an observation code.

By incorporating this rule with the algorithm, the length of each observation code along with the observation code sequence is obtained. Observation codes with length **1** are considered to be codes with shorter length and they evidently increase inter-class variations while reducing intra-class variations. Further, their role in defining the shape of the character is minimal. Therefore, discarding such observation codes from the sequence is advantageous in enhancing the matching between sequences that belong to similar characters.

Likewise, after the observation code sequence is extracted, the sequence will be refined by removing observation codes with length **1**. By doing so, the sequence in Fig.5.16b, can be refined as shown in Fig. 5.17.

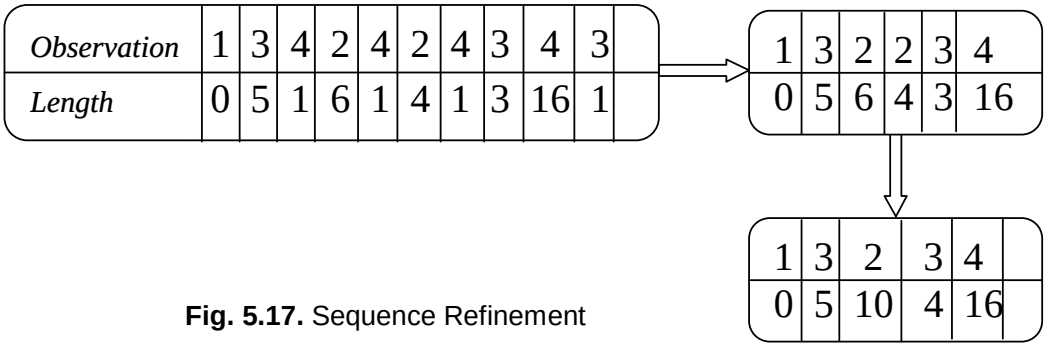

Fig. 5.17. Sequence Refinement

After this refinement step, observation code sequences may include identical consecutive observation codes (See Fig. 5.17). Thus, the sequences are further refined to remove such codes by merging them into one observation code, as they have no use and the action simplifies the sequence. This also helps to reduce the variation between the sequences of similar characters.

Even though, the length of observation codes in an observation code sequence is originally computed for the purpose of avoiding shorter codes, we have observed that it is also a useful parameter for matching sequences during classification. Basically, it measures and quantifies the degree of similarity or dissimilarity between sequences, which is an important input to decide on which stroke or character the sequence belongs to. To take this advantage, the length of each observation code is considered while observation code sequences are extracted from a given preprocessed data.

But, we have noticed that there is still an intolerable variation between sequences of the same character, which might result in misclassification of characters. It has also been observed that most variations between sequences of the same character are created by the observation code **4**. In the majority of the sequences, this code appears with relatively shorter length except for horizontal and vertical line strokes. Horizontal strokes are characterized by the x observation sequence (1,2) or (1,3) depending on the direction of writing, and by the y observation sequence (1,4) since the y values will be constant (nearly constant) among the data points along the horizontal line. Similarly, the x observation sequences extracted for vertical line strokes will be (1,4) and the y observation sequences will be (1,2) or (1,3) depending on whether the vertical line is drawn upward or downward. It is possible to conclude that the observation code **4** has a considerable part in describing such strokes. In spite of this fact, it was preferred to leave all occurrences of the observation code **4** from all sequences that are used to classify strokes and characters for the coarse classification task. These codes would be included while detailed matching is later performed in the second-stage classification. The whole process to extract observation code sequence, which is ready for training and classification, is summarized and illustrated in Fig. 5.18. Note that the length of element observation codes is processed in each step along with the observation code sequences.

Step 1:
Input: Preprocessed handwriting data for a character
Procedure: Observation Code Extraction:
(Using the algorithm in Fig. 5.13)
Output: X and Y observation code sequences along with the length of each element code (original form)
Step 2:
Input: Original X and Y observation code sequences
Procedure: First stage refinement:
Removal of shorter observation codes (codes with length 1)
Output: Refined X and Y observation code sequences
Step 3:
Input: Refined X and Y observation code sequences
Procedure: Second stage refinement:
Removal of identical consecutive codes in the sequence
Output: Further refined X and Y observation code sequences
Step 4:
Input: Further Refined X and Y observation code sequences
Procedure:
If the X and Y observation code sequences are needed for coarse classification, then third stage Refinement will be done which constitutes the removal of the observation code **4**. Otherwise, the sequence will be left as it is.
Output: X and Y observation code sequences which are ready for training or classification.

Fig. 5.18. Observation Code Sequence Extraction Process

In conclusion, this section discusses the pattern representation scheme used to represent handwriting patterns that involve in the online handwriting recognition engine developed in this work. During training, the engine stores reference handwriting patterns that are represented by observation code sequences. In the same way, unknown patterns are also represented by this scheme before they can be compared with the reference patterns. The next section explains how the training is conducted. The subsequent section explains the classification.

5.4.5. Training

The main goal of the training process is to teach the recognition system an individual's handwriting so that it could be able to classify unknown handwriting patterns written by the

same writer. Particularly, training a writer-dependent system focuses on identifying a writer's handwriting style who is supposed to use the system. The recognition engine learns the handwriting of an individual by observing and recording (persistently) how each character is written. This requires to have set of characters written by the individual and the data is usually referred as training data. The training data needs to be essentially complete in the sense that it should include samples for each character. The training should also be effective enough to bring about better recognition accuracy for the recognition engine. However, it is also a desirable feature for a training process to use small amount of training data as much as possible.

The recognition engine has a module named '*Trainer'* which manages the training activities. Observation code sequences, which are extracted from the training data by the feature extraction component of the system, are the input to the trainer (Fig. 5.4). The trainer then acts on the input observation code sequences to produce a set of *reference* observation code sequences for each stroke of a character that is supposed to be recognized by the recognition system.

The general procedure used to train the system is to identify and record unique observation code sequences for the various strokes that constitute a character. A number of samples for a character will be collected and observation code sequences for each stroke will be extracted from each sample. Then, the sequences for similar strokes in the different samples are compared to each other to identify unique observation code sequences. The trainer then keeps these sequences as reference sequences that represent the strokes in the character. It should be noted that the length of element observation codes in the sequences are also stored as they are important during classification. Keeping the length of each code that involve in the sequences is a way to know the chance of the code to appear in the sequence in that particular order. The training is conducted both for the x and y observation sequences of the strokes.

The output of the *trainer* module is a text file that stores all possible observation sequences of strokes of a character that are obtained from the given samples. This file is refereed as a *reference file*. After training, two reference files are generated for each character that correspond to the X and Y observation sequences. There is a possibility that more than one

pair of reference files would be produced for a character if the writer claims that he/she writes the same character by using different number of strokes and/or in different order.

The reference file mainly stores the possible observation sequences of strokes of a character, which also includes the length of each code in the sequences. An example of a reference file is shown in Fig. 5.19. When observation sequences are written in the reference file, an observation code and its length are placed in the same line as a pair. These pairs are listed in different lines until the last code of the sequence. At the end of each unique sequence of a stroke, the pair (10,10) is written for the purpose of marking its end.

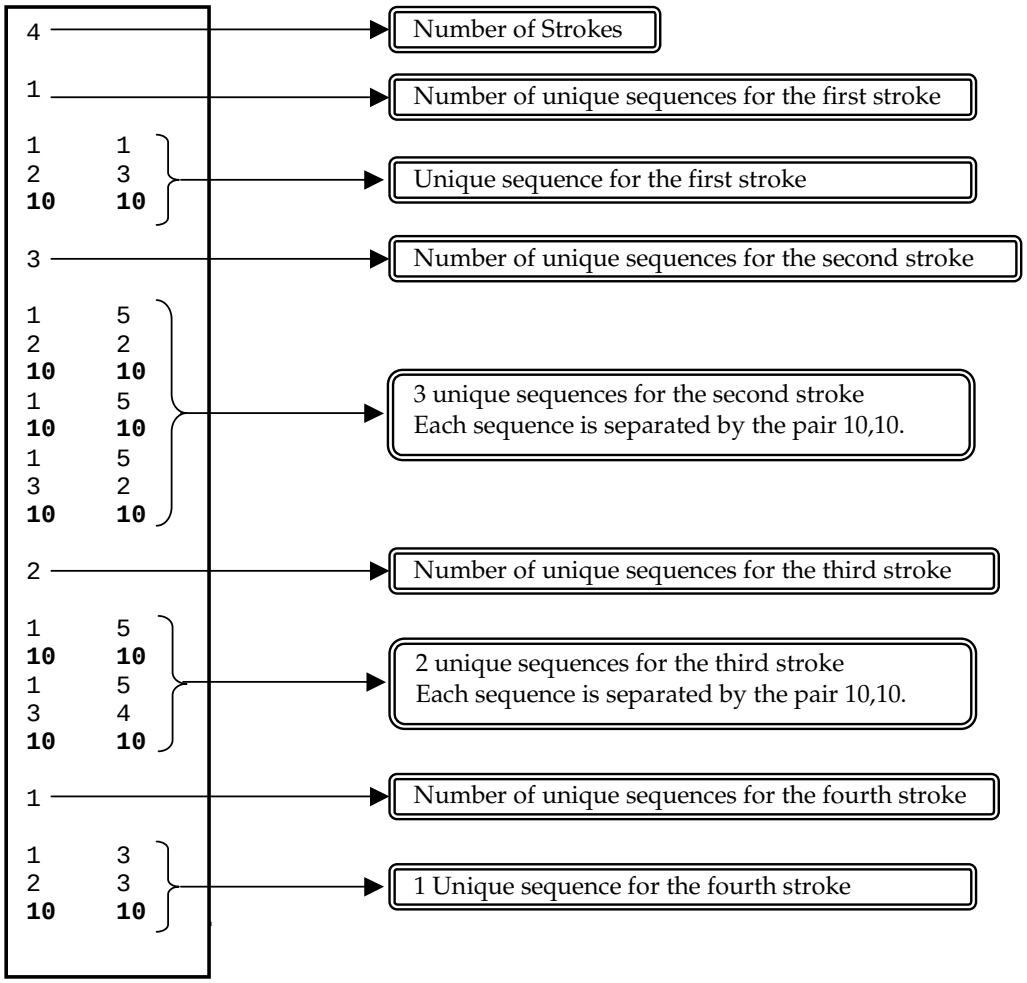

Fig. 5.19. A Reference File
[Generated for X observation sequence of the letter "ع" (Writer B)]

In addition to the possible observation sequences of strokes of a character, the reference file is made to contain other useful information such as the number of strokes. The number in the first line in all reference files corresponds to the number of strokes of the character as

written by the writer for whom handwriting the training was conducted. A number in the second line is the number of unique observation sequences for the first stroke. Then, the sequences of the particular stroke will be written. When all the unique sequences are exhaustively written for a stroke, another number is inserted before the listings of the sequences for the next stroke. This number is the number of unique sequences for the next stroke.

In our case, three samples are provided for the trainer for a single character as training data. The number of samples can however be increased as the need may be. In Fig. 5.20 and Fig. 5.21, two examples are given to illustrate how training of a single character is conducted from three given samples. The trainings shown in the figures were conducted for Y observation code sequence of the character "ա" and X observation code sequence of the letter "ճ" respectively. All the samples in the figures are written by Writer B.

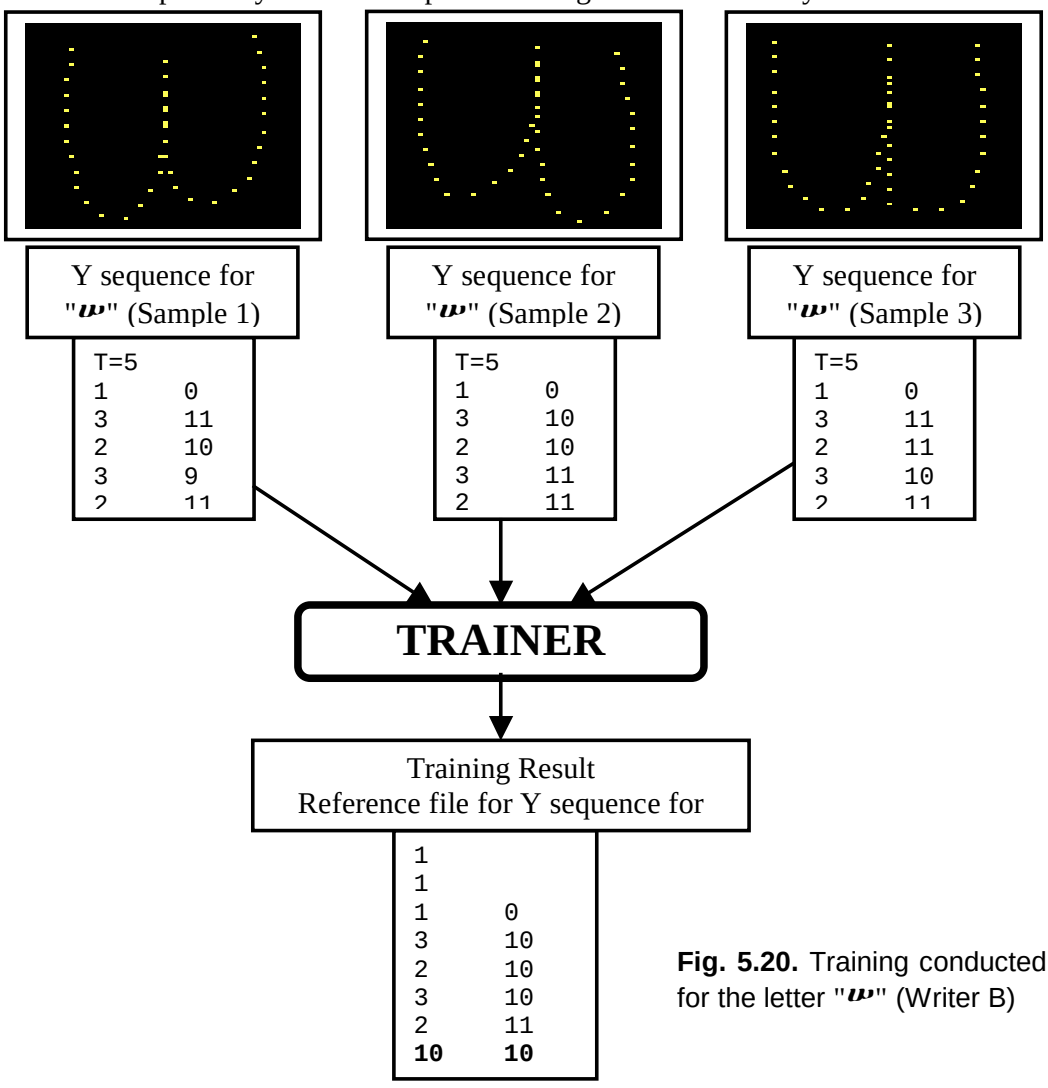

Fig. 5.20. Training conducted for the letter "ա" (Writer B)

In Fig. 5.20, three different samples of the letter "ሠ" are provided and the indicated Y observation code sequences are extracted from each sample by the feature extraction module. These sequences are input to the trainer module and the trainer generates the reference file. The trainer first determined the number of strokes in each sample and it finds out that only a single stroke is used to write all the samples of the letter (by Writer B) and **1** is recorded as the number of strokes in the reference file. Then, the trainer compared the sequences of the strokes from the three samples. The three sequences are identical in terms of the number of code elements and the order that they appear in the sequence. This leads the trainer to the decision of making the number of unique sequence for the stroke **1**, hence **1** in the second line of the reference file. Then, the only unique sequence was recorded right after this number. Another important action done while recording the sequence is that the length of each code element is computed by finding the average of the length of the corresponding code elements in the given three sequences.

The second example in Fig. 5.21 is quite different from the previous case. The writer used three strokes to write the character and this can be seen from the sequences. Each stroke sequence is separated by the pair (5,0). Moreover, the sequences of the first stroke differ in the three samples. Though the gap between these sequences seems to be big, it is not in real sense. The sequences in the first and second sample differ by the observation code **2** which is found between **1** and **3** in the sequence of the second sample. However, the length of this observation code is only 2 which is not a large number. Another interesting relation is exhibited between the first stroke of the first and third samples. The existence of the two observation codes **3** and **2** between the first code **1** and the last code **3** are the reasons to create a difference between the two sequences. But, if **2** does not exist between the second **3** and the third **3**, then it is possible to combine the two to have only a single **3** with length which is equal to the sum of the length of the two identical codes. The length becomes 5 (3+2) which is near to 6 that is the length of the code **3** in the sequence of the stroke in the first sample. The resulting sequence would be {1,3} with the corresponding length 0 and 5. In any case, the trainer records all the three unique sequences in the reference file as the possible sequences for the first stroke of the letter "ፌ". A similar analysis is done to treat the second and the third strokes (See Fig. 5.21).

The training is also done in the same way for sequences that contain the code 4, because sequences including this code are sometimes needed during classification.

The section that follows discusses the classification procedures.

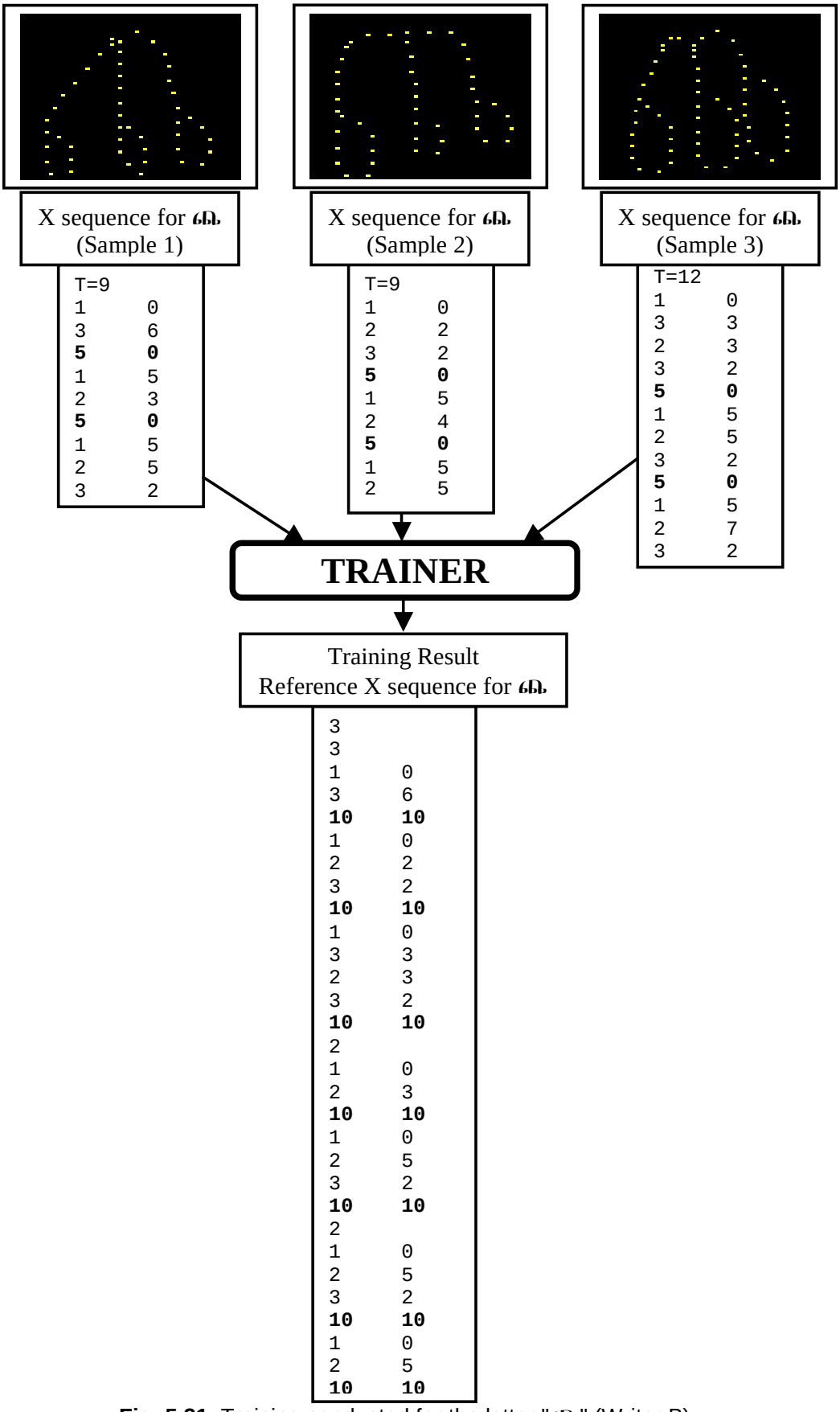

Fig. 5.21. Training conducted for the letter "ҩҧ" (Writer B)

5.4.6. Classification

A trained character handwriting recognition system is ready to receive character handwriting data and attempt to classify it as one of the characters in the reference data set by which it was trained. In the previous section, the training procedures are discussed. The training activity aims at making the system familiar with one's handwriting and capable of recognizing characters written by the same writer.

A module named '*Recognizer'* is available in the recognition system for the purpose of classification. Similar to the trainer, the recognizer gets its input from the feature extraction module, which produces the X and Y observation code sequences from the online handwriting data of an unknown character. Thus, the job of the recognizer is to assign the given observation code sequences to one of the characters in the reference set. A three-stage classification method is designed and developed to implement the recognizer. This layered structure of the recognizer is necessary to enhance the accuracy of the result.

Each classification stage is given a name. The first-stage classification is referred as *coarse classification.* The second and the third stage classification steps are named as *detailed matching* and *superimposing matching* respectively. The layered structure of the recognizer imposes that it should be passed through an upper layer before getting to a lower layer. This means that the top most layer of the recognizer is the coarse classification, the detailed matching is the middle layer and the superimposing matching is the bottom layer. Output of upper layers is an input for lower ones. Fig. 5.21a and Fig. 5.24b show the structure of the recognizer and the control flow in the recognizer respectively.

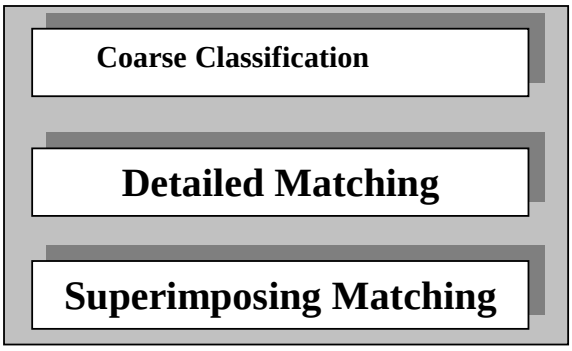

Fig. 5.24a. The Structure of the recognizer

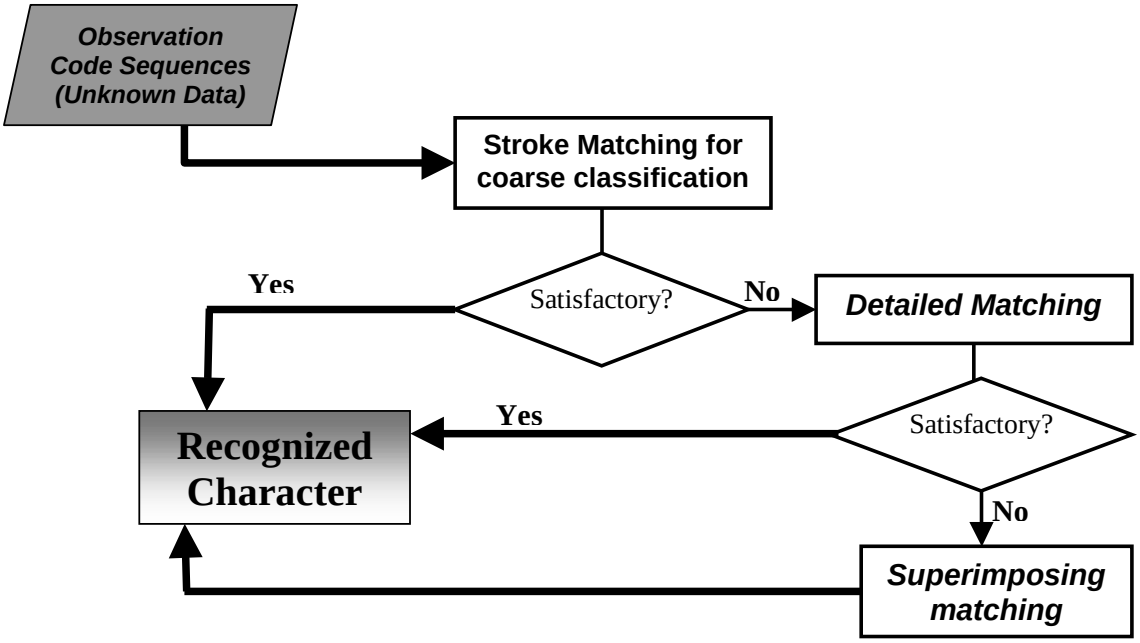

Fig. 5.24b. Flow in the recognizer

As illustrated in Fig. 5.24b, the X and Y observation sequences coming from the feature extraction module are first passed through the coarse classification layer. The layer in its part examines the sequences and the characters in the reference set to produce the five most likely characters in order. Of the five characters, the character in the first order is suggested to be the character that belongs to the given sequences by the coarse classification layer. However, the recognizer has a mechanism to judge on what the first layer claims. If the recognizer finds that the result is not acceptably certain, it will pass the five characters to the second layer so that the second layer does its best to choose one of the five characters by conducting detailed matching. In some cases, it is still possible that the detailed matching layer hesitates on choosing one of the five characters and the third layer will be allowed to resolve this situation. Such steps help to guarantee the correctness of the output. This implies that a character might not necessarily pass through the second and/or the third layers of the recognizer.

Each layer utilizes different techniques to classify unknown handwriting patterns. The simplest method is used by the first layer and its simplicity is expressed by the resources needed to accomplish it. During detailed matching, more resources could be used. The

superimposing matching technique is computationally complex as compared to the methods applied in the other two layers. The layered structure of the recognizer has an advantage in minimizing the number of reference characters that are to be compared with the unknown character when moving from one layer to another. This is useful to avoid unnecessary efforts that would be made in the second and third stage classification. Hence, the speed of the recognizer will be improved.

The following subsections explain the three classification stages in detail.

5.4.6.1. Coarse Classification

The X observation and Y observation code sequences of an unknown character are always initially presented to the coarse classification layer. The coarse classification layer utilizes the sequences after they are refined to remove shorter codes and the observation code **4** as explained in section 5.6.2.1. Thus, all sequences presented for this layer do not include the observation code **4** and consequently are less detailed.

In general, the coarse classification layer is expected to find closest matches for the unknown sequences from the reference set. To realize this, it should compare the input unknown sequences with the available sequences from the reference set. The reference sequences are already prepared by the trainer and stored in the system persistently. The comparison can possibly be carried out in different ways. In this system, the comparison is aided by computing distance between sequences (*inter-sequence distance*). Distance in this sense is a measure that indicates the degree of similarity of the sequences and it leads to a conclusion of choosing one of the reference sequences as a classification result. Reference sequences with smaller inter-stroke distances are generally accepted as the most likely match of the unknown sequences.

As stated earlier, the coarse classification layer processes both the X and Y observation code sequences for a given character. Thus, two types of inter-sequence distances are calculated and a total *inter-character distance* is computed by adding up the two X and Y inter-sequence distance values. From now on, we refer to these distances as the X inter-sequence distance and the Y inter-sequence distance. Both types of inter-sequence distances are computed by first finding distance between strokes and adding them to get the

total distance between the sequences. Note that we are dealing with three types of distances where one is derived from the other. These are *inter-stroke distance*, *inter-sequence distance* and *inter-character distance*. Inter-stroke distance is illustrated in Fig. 5.25.

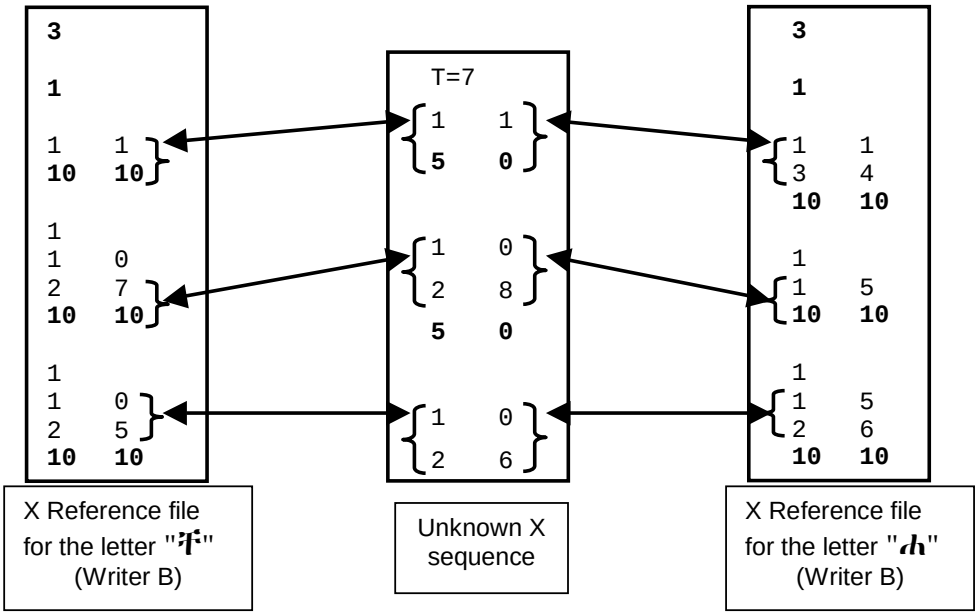

Fig. 5.25. Inter-stroke distance

In Fig. 5.25, an unknown X sequence with three strokes is compared with two reference file which are products of the trainer after the system has been trained for the letters ह and ध. Note that the distance is calculated between corresponding strokes as shown by the arrows in the figure. The inter-sequence distance is the sum of the distance between the three strokes. The relationship between the abovementioned three distances is outlined below:

$$dist_{sequence} = \sum_{1}^{n} dist_{stroke}$$

$$dist_{character} = Xdist_{sequence} + Ydist_{sequence}$$

where $dist_{stroke}$ is inter-stroke distance,

$dist_{sequence}$ is inter-sequence distance,

$dist_{character}$ is inter-character distance,

$Xdist_{sequence}$ is X inter-sequence distance,

$Ydist_{sequence}$ is Y inter-sequence distance, and

n: is the number of strokes in the given sequences.

5.4.6.1.1. The Computation in the Coarse Classification Layer

The set of the general procedures for classifying unknown characters coarsely is shown in Fig. 5.26. Given the unknown sequence (say the X sequence) and the reference files (X reference files in this case), the distance between the unknown sequence and the sequences in the reference files (***inter-sequence distance***) is computed.

The process is started by first determining the number of strokes in the unknown sequence. It is also possible to get the number of strokes in the reference sequence in charge by reading the number in the first line in the reference text file (See Fig. 5.17 to see in what format the information is structured in any reference file which is the product of the *trainer*). Acquiring this information is quite important in order to avoid reference sequences with different number of strokes than that of the unknown pattern. This is to say that the distance will be computed if and only if the number of strokes is equal in the unknown and reference sequences. If the opposite happens, the algorithm simply assigns the value 500 as an inter-sequence distance. This number is large as compared to the expected computed distances. Thus, it is an indication to show the large gap between the unknown and reference sequence. As a result, this sequence would be one of the last choices to be selected as a match.

Input: X Observation code Sequence OR Y Observation code sequence

Get *NoOfStrokesFromInputSeq*, no of strokes in the input sequence
n ← 1
Do
    Get (open) the nth reference file.
    Read *NoOfStrokesFromRefFile*, number of strokes in the reference file
    If (*NoOfStrokesFromInputSeq* = *NoOfStrokesFromRefFile*)
    Begin
        i ← 1
        Do
            Get $StrokeSeqFromInput_i$, ith stroke of the unknown input sequence
            Read $NoOfUniqueSeq_i$, number of unique sequences for the ith stroke in the reference file
            j ← 1
            Do
                Get $StrokeSeqFromRef_{i,j}$, jth unique stroke sequence of the ith stroke in the reference file
                Compute $InterStrokeDistance_{i,j}$, inter-stroke distance between $StrokeSeqFromInput_i$ and $StrokeSeqFromRef_{i,j}$ ***(Algorithm in Fig 5.25)***
            Until (j <= $NoOfUniqueSeq_i$)
            Compute *MinOfInterStrokeDist*, minimum of inter-stroke distances
            $StrokeDistance_i$ = *MinOfInterStrokeDist*
            Increment i
        Until (i <= *NoOfStrokesFromInputSeq*)
        Compute *TotalSeqDistance*, the total distance between the input sequence and the reference sequence by adding up all inter-stroke distances.
    End
    Else
        TotalSeqDistance = 500 //This is to make this reference out of choice
    Increment n
Until (n <= 34)
Get *TopFive*, five sequences with the smallest distance.
MinDist ← $DistOfTopCand_1$, distance computed for the top most character
k ← 2
TopCnt ← 1
Do
        If (| $DistOfTopCand_k$ - $DistOfTopCand_1$| <=5)
                TopCnt ← TopCnt + 1
        k ← k + 1
Until (k > 5)
If (TopCnt > 1)
        Pass to the next layer taking the nearest candidates to the first one.
else
        Report that $DistOfTopCand_1$, top most character as the recognized character

Fig. 5.26. Algorithm for Coarse Classification Procedures

If the number of strokes in the unknown and reference sequences is equal, the distance computation is started by retrieving the sub-sequence for the first (it could be the only one if the number of strokes is 1) stroke from the unknown sequence.

This is followed by reading the number of unique sequences for the first stroke from the reference file. This is done to make sure that the unknown sequence is compared with all possible sequences of the corresponding stroke in the reference file. Accordingly, distance between the first stroke in the input sequence and the unique sequence(s) listed for the first stroke in the reference file are calculated. If the number of unique sequences for the strokes is greater than one, then the stroke sequence with the minimum of the distances will be associated to the input stroke. The Inter-stroke distance calculation is discussed in the next section.

Similarly, all strokes are analyzed exhaustively in this way. At the end of this process, we have the inter-stroke distances in hand and the total inter-sequence distance is equal to the sum of all the inter-stroke distances. The coarse classification layer finally picks five sequences with the smallest distances as the top five candidates ordered by their closeness to the reference character. The top most from the five candidates is the recognized character as proposed by the layer. Though, it seems that it is uncertain to take this proposal for granted, we have proved through experimentation that 93.4% of the suggested top most characters are the correct characters. To decide on whether accepting the top most character as a final output or to further compare the five candidates, the recognizer attempts to examine the distance difference between the top most character and each of the remaining four characters. If the distance difference between the top most character and the remaining four characters is greater than **3**, the recognizer will accept it as the recognized character. Otherwise, the top most character and its nearest candidates are passed to the detailed matching layer. The distance difference value **3** is chosen based on experiments. That means, we have seen that the possibility of one of the remaining four characters (other than the top most) to be the correct one is much lesser in cases in which a distance difference of 3 exists.

Next, the most important issue is the technique to compute the distance between stroke sequences. We designed and implemented a simple but clever algorithm for this purpose. It has been proven that the algorithm produces a good inter-stroke distance measure that

determines the degree of similarity between stroke sequences. This means that smaller distance measures are obtained for sequences with higher level of similarity. An explanation of this algorithm is presented in the subsection that follows.

5.4.6.1.2. Inter-Stroke Distance Computation

The algorithm for inter-stroke distance computation is presented in Fig. 5.27. In this algorithm, the *length* of a stroke sequence is the number of elements in the observation codes sequence. If the stroke sequences are of the same length, the algorithm compares each corresponding observation code in the two sequences to see if they are identical. If they are identical, then the codes might differ only in their length. Thus, the difference between the lengths of the two code values is added to the distance (Fig. 5.27, line 9). However, if the codes are different, the difference is in observation codes which is a much bigger difference than the previous case. Therefore, the lengths of both observation codes are added to the distance (Fig. 5.27, line11).

***Input*: Two stroke sequences**

// *UnknownStrokeSeq*: Sub-Sequence for a stroke in the unknown sequence
// *RefStrokeSeq*: Sub-Sequence for a stroke in the reference sequence
//*UnknownStrokeLen*: set of lengths for the codes in *InputStrokeSeq*
// *RefStrokeLen*: set of lengths for the codes in *RefStrokeSeq*

1: Compute *LengthOfUnknownStrokeSeq,* no of element observation codes in - *UnknownStrokeSeq*
2: Compute *LengthOfRefStrokeSeq,* no of element observation codes in - *RefStrokeSeq*

3: *dist* ← 0 //*dist* is inter-stroke sequence
4: if (*LengthOfUnknownStrokeSeq* ≠ *LengthOfRefStrokeSeq)*
5: Expand the shorter sequence ***(Algorithm in Fig. 5.26)***

6: i ← 0
7: Do
8: if (*UnknownStrokeSeq* $_i$ = *RefStrokeSeq* $_i$*)*
9: dist ← dist + | *UnknownStrokeLen* $_i$ - *RefStrokeLen* $_i$ |
10: else
11: dist ← dist + *UnknownStrokeLen* $_i$ + *RefStrokeLen* $_i$
12: i ← i + 1
13: Until (i > *LengthOfUnknownStrokeSeq*)

***Output*: distance between the two sequences**

Fig. 5.27. Algorithm to compute inter-stroke distance

The difficult case is when the sequences are of different length. In such cases, it is impossible to create a one-to-one correspondence between the elements observation code sequences. To solve this problem, the sequence with the shorter length is expanded to be of equal length with the longer one. The expansion is not straightforward and it needs careful examination of the sequences. The reason is because there is a need to align observation codes to their likely match in the longer sequence.

In Fig. 5.28, the expansion algorithm is shown. The algorithm, in general, finds a suitable place of each observation codes in the shorter sequences as to align them to their match in the longer sequences. The rest of the code places and their length in the shorter sequence are filled with the value **0**.

***Input*: Two stroke sequences (one shorter, the other longer)**

// *ShortSeq*: The shorter sequence
// *ShortSeqLen*: Set of Lengths for the code in the shorter sequence
// *LongtSeq*: The longer sequence
// *LongSeqLen*: Set of Lengths for the code in the longer sequence

```
1:      Compute LengthOfShortSeq, length of the ShortSeq
2:      Compute LengthOfLongSeq, length of the LongSeq
3:      i ← 1
4:      min ← 50            //a large number is assigned
5:      Do
6:          j ← 1
7:          Do
8:              if (ShortSeq i = LongSeq j)
9:                  l ← | ShortSeqLen i – LongSeqLen j |
10:                 if (l < min)
11:                     min = l
12:                     index = j
13:                 j ← j + 1
14:         Until ( j > LengthOfLongSeq
15:             if (i ≠ index)
16:                 ShortSeq index = ShortSeq i
17:                 ShortSeqLen index = ShortSeqLen i
18:                 ShortSeq i = 0
19:                 ShortSeqLen i = 0
20:         i ← i + 1
21:     Until (i > LengthOfShortSeq)
22:     Fill the rest of the places in the shorter sequence by 0.
```

Fig. 5.28. The Sequence Expansion Algorithm

An example is provided to illustrate the inter-stroke computation in Fig. 5.29. Two sequences for the different instances of the same letter "***ʊ***" are compared to the X reference file that was produced during training conducted for letter "***ʊ***" of Writer A. It can be seen that the writer used only one stroke to draw the characters. In the second instance of the character, the length of the observation code sequence and the length of the reference code sequence are equal. Moreover, the sequences are identical in their element observation codes. Thus we need to compute only the length difference of each code. In the first instance, the lengths of the sequences are different and consequently an additional step, *expansion* is needed before computing the distance. Accordingly, the X reference sequence (i.e. the shorter sequence) is expanded to have the same length with the first instance sequence. Then, the two sequences are compared to calculate the distance between them. Following the steps of the algorithm, each corresponding code is compared. The first and the second element codes are identical in both sequences. But, the third element is **0** in the reference sequence and **3** in the other sequence. This difference implies that the length of both will be added to the distance value (Fig. 5.27, Line 11).

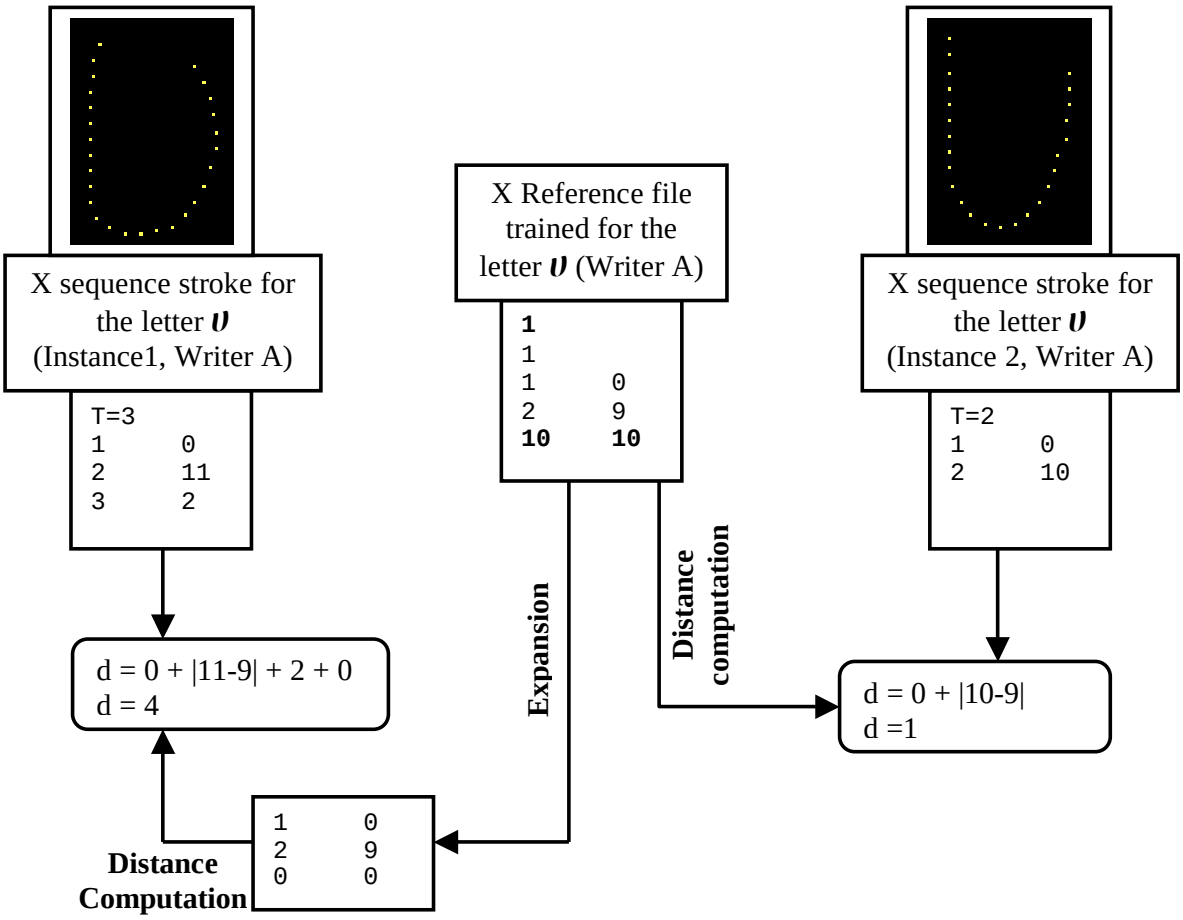

Fig. 5.29. An example Inter-stroke distance computation

Note that the ***X distance*** is larger for the first instance. This is due to the slightly deformed shape of the character in its right side. The existence of the bend results in having the last observation code **3** (decrease in x values) in its sequence. While computing the distance, the existence of this code in the sequence has a major part in making the distance larger. Therefore, the inter-stroke distance computation algorithm is effective in modeling the level of similarity between strokes. This is important, as the classification result highly depends on it.

Table 5.2 shows the coarse classification results for three sample characters (written by Writer A). The table lists the X observation code distances and Y observation code distances of the sample test characters against each of the reference characters. It also includes the total distance. In all the three cases, the smallest distance belongs to the closest characters. Despite this, the first layer's proposal in the second case is not taken as a final classification result since the first two ("**ለ**" and "**ረ**") from the group are competing candidates as the distance of "**ለ**" is less than only by **1** from that of "**ረ**". This situation forces the recognizer to take a measure before reporting the classification result, which is passing the two characters to the next layer. This implies that the next layer is supposed to identify the correct one, if not pass them to the third layer.

No.	Reference character	Character ሀ			Character ለ			Character ዘ		
		X	Y	Tot	X	Y	Tot	X	Y	Tot
1	ሀ	4	3	7	500	500	1000	500	500	1000
2	ለ	500	500	1000	5	2	7	500	500	1000
3	ሐ	500	500	1000	19	33	52	500	500	1000
4	መ	12	25	37	500	500	1000	500	500	1000
5	ሠ	4	23	27	500	500	1000	500	500	1000
6	ረ	6	13	19	500	500	1000	500	500	1000
7	ሰ	500	500	1000	17	30	47	500	500	1000
8	ሸ	500	500	1000	500	500	1000	15	26	41
9	ቀ	15	35	50	500	500	1000	500	500	1000
10	በ	4	16	20	500	500	1000	500	500	1000
11	ተ	500	500	1000	7	1	8	500	500	1000
12	ቸ	500	500	1000	500	500	1000	13	22	35
13	ኀ	5	18	23	500	500	1000	500	500	1000
14	ነ	6	12	18	500	500	1000	500	500	1000
15	ኘ	500	500	1000	11	3	14	500	500	1000
16	አ	12	11	23	500	500	1000	500	500	1000
17	ከ	4	12	16	500	500	1000	500	500	1000
18	ኸ	500	500	1000	14	25	39	500	500	1000
19	ወ	500	500	1000	500	500	1000	19	19	38
20	ዐ	17	2	19	500	500	1000	500	500	1000
21	ዘ	500	500	1000	500	500	1000	1	1	2
22	ዠ	500	500	1000	500	500	1000	500	500	1000
23	የ	500	500	1000	13	7	20	500	500	1000
24	ደ	14	15	29	500	500	1000	500	500	1000
25	ጀ	500	500	1000	18	23	41	500	500	1000
26	ገ	6	13	19	500	500	1000	500	500	1000
27	ጠ	4	32	36	500	500	1000	500	500	1000
28	ፈ	14	41	55	500	500	1000	500	500	1000
29	ጸ	500	500	1000	19	32	51	500	500	1000
30	ፀ	19	17	36	500	500	1000	500	500	1000
31	ፀ	500	500	1000	17	14	31	500	500	1000
32	ፈ	500	500	1000	12	7	19	500	500	1000
33	ፐ	500	500	1000	13	14	27	500	500	1000
34	ሸ	500	500	1000				500	500	1000
Top Five characters		**ሀ**, distance **7** **ከ**, distance **16** **ነ**, distance **18** **ረ**, distance **19** **ዐ**, distance **19**			**ለ**, distance **7** **ተ**, distance **8** **ኘ**, distance **14** **ፈ**, distance **19** **የ**, distance **20**			**ዘ**, distance **2** **ቸ**, distance **35** **ወ**, distance **38** **ሸ**, distance **41** **ሀ**, distance **1000**		
Recognized Character		**ሀ**			**ለ and ተ are passed to the middle layer.**			**ዘ**		

Table. 5.2. Coarse Classification Results for three sample characters from the testing data

5.4.6.2. Detailed Matching

Characters with equal or nearly equal (with difference less than 3) distance are passed to the second layer of the recognizer so that detailed matching could be done to identify the most likely character that would be considered as the classification result. This requires using a more detailed sequence to compare the characters. We, purposely, took out all occurrences of the observation code **4** from all sequences before they were provided to the coarse classification layer. Now, it is time to include this code in the sequences, in order to make them more detailed than the previous sequences that have been used in the first layer.

The main difference between the coarse classification layer and the detailed matching layer is the type of the sequences they use to classify. Moreover, the competition would be among fewer numbers of characters in the detailed matching layer than the coarse classification. As we did in the coarse classification layer, distances are computed between the detailed sequences in a similar manner by applying the algorithms discussed in the preceding section.

An example is provided below (Fig. 5.30a and Fig. 5.30b) to show how the inclusion of the observation code 4 makes a difference. In the first layer, the distance between the character "ረ" (writer A) and the reference files trained for the characters "ረ" and "ገ" is **0**. This is because the reference files are identical. This means that the first layer often confuses the two characters. To resolve this, the detailed matching (Fig. 5.30b) compares the original form of the sequences that include the observation code **4**. Distance values that can clearly distinguish the two characters are obtained. Thus, the detailed matching layer produced the character "ረ" as the final classification result.

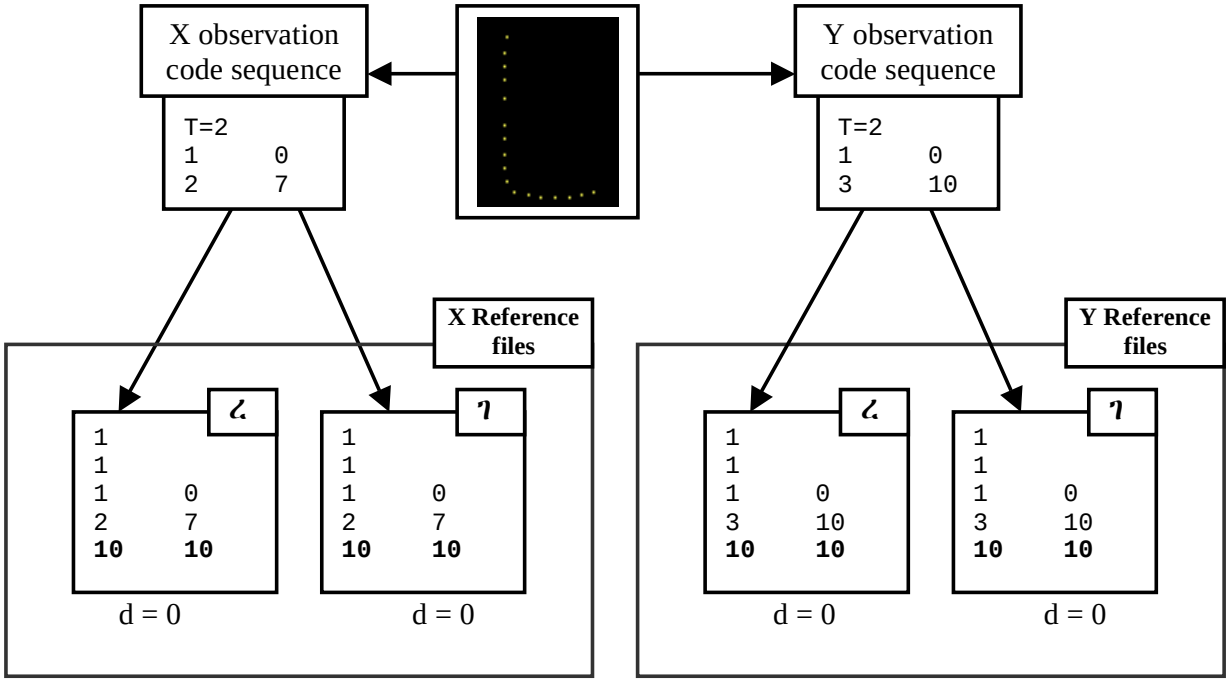

Fig. 5.30a. Coarse Classification for the character "ረ"

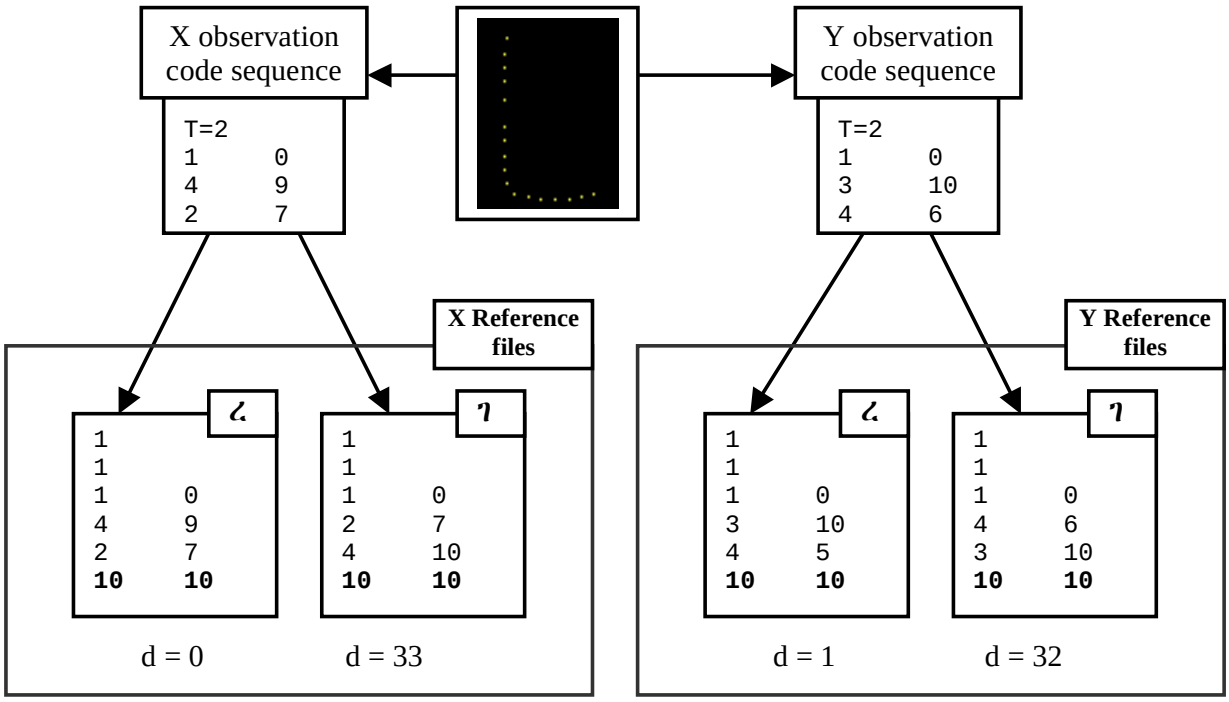

Fig. 5.30b. Detailed Classification for the character "ረ"

Yet, this is not always the case. Some character are also confused by the middle layer which in that case passes them to the last layer, namely the superimposing matching.

5.4.6.3. Matching by Superimposing

As stated earlier, characters that have been passed to this layer are those that cannot be identified by the first as well as the second layers of the recognizer. For this reason, the classification here must be handled in a special way. This means that observation code sequences are no more useful in distinguishing these characters. This necessitates employing a different approach to distinguish one character from the other.

The methodology employed in the third layer classification scheme constitutes superimposing the unknown character on top of the reference characters and measure the degree of matching, hence superimposing matching. To measure the matching, the distance between the corresponding data points (*inter-point distance*) that form the reference characters and the superimposed character is computed. After that, the inter-point distances are added to get the matching distance. A reference character that results in smaller matching distance is considered to have high degree of matching and it will be the result of the recognizer.

The superimposing matching process incorporates two major steps namely overlying the unknown character on top of each of a reference character and computing the matching distance.

5.4.6.1.3. Superimposing

Superimposing is the action of taking the unknown character and putting it on top of the reference character. An algorithm was designed to perform this task and it is presented in Fig.5.31. The algorithm first identifies the boxes in which the reference and the unknown character are drawn turn by turn. This is achieved by computing the maximum and minimum values of the x and y coordinate for both characters. To put the unknown character on top of the reference character, it is necessary to translate all the data points that make up the unknown character. The translation is done by adding values to the x and y coordinate values of the points. Thus, the difference between the maximum x value of the reference character and that of the unknown character, and the difference between the minimum x value of the reference character and that of the unknown character are averaged to get the change in x. The same thing will be done for obtaining the change in Y. Then,

***Input*: Preprocessed data of a reference character and an unknown character**
//Each data point of the unknown character is a triple (x, y, z)

1: Get *UnknownMaxX, UnknownMinX, UnknownMaxY, UnknownMinY*
//to get the box in which the unknown character is drawn
2: Get *RefMaxX, RefMinX, RefMaxY, RefMinY*
//to get the box in which the reference character is drawn
3: ChangeInX ← [(*RefMaxX – UnknownMaxX)+(RefMinX- UnknownMinX*)]/2
4: ChangeInY ← [(*RefMaxY – UnknownMaxY)+(RefMinY- UnknownMinY*)]/2
5: i ← 1
6: Do
7: if ($x_i \neq 0$)
8: $newx_i \leftarrow x_i$ + ChangeInX
9: $newy_i \leftarrow y_i$ + ChangeInY
10: $newz_i \leftarrow 0$
11: else
12: $newx_i \leftarrow 0$
13: $newy_i \leftarrow 0$
14: $newz_i \leftarrow 0$
15: i ← i + 1
16: Until (i > n) //n is the number of data points that make up the unknown character

each data point of the unknown character is translated to a relative data point inside the box in which the reference character has been drawn.

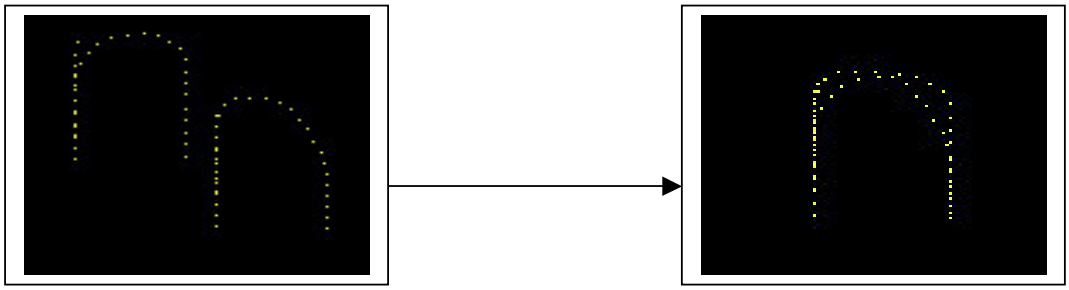

Fig. 5.32a. Superimposing Algorithm

Fig. 5.32b. Characters after superimposing

The character "ለ" and "ሀ" (Writer B) are often confused by the second layer and they are sent to the third layer for further examination. Fig. 5.32a and Fig. 5.32b show how superimposing is done on them. It can be clearly seen that the first set of characters (Fig. 5.32a) are well matched than the second set (Fig. 5.32b). This difference could be quantified by calculating the distance between each corresponding data point of the two overlapped characters.

5.4.6.1.4. Matching Distance Computation

The matching distance is equal to the sum of the inter-point distances. The inter-point distance is calculated as:

$d_i = |x - newx| + |y - newy|$ where (x,y) is a data point on the reference character and (newx, newy) is the corresponding data point on the character that is superimposed on the reference character.

The matching distance is:

$D = \sum_{1}^{n} d_i$ where d_i is the inter-point distance

In the example given in Fig. 5.32a and Fig. 5.32b, after superimposing the characters, the matching distance between each pair is computed. It was obtained that the matching distance for the characters in Fig. 5.32a is ***465*** and it is ***875*** for the characters in Fig. 5.32b. This led to the conclusion that the tested character is recognized to be "ለ".

5.5. Summary

In this chapter, the online handwriting recognition engine designed and developed for the basic Ethiopic characters has been discussed. The design of the recognition engine with the detailed explanation of its components has been presented. We have also described the *Movalyzer* digitizer software that was used for data collection. Algorithms that were designed for accomplishing the preprocessing activities have also been presented. We dealt with the feature extraction process widely. Under this, we covered the observation code sequence extraction process, which is the proposed handwriting pattern representation technique. We have claimed that we developed a recognition engine based on this pattern representation scheme. Along this, the new algorithms designed for the purpose of training and classification have been presented.

CHAPTER SIX

EXPERIMENTATION AND DISCUSSION

Experiments have been conducted to test the system for its accuracy. In this chapter, we present the results of these experiments to evaluate the performance of the recognizer and the performance of each of the layers in the recognizer.

6.1. Experimentation

Two major experiments were conducted to test the recognition engine for its accuracy and ability to handle variations. It was already mentioned that two writers (writer A and writer B) involved in the data collection process and the experiments were carried out on the handwriting of each writer. Nine instances of each of the 33+1 character written by these writers were collected. This constitutes the whole available data. The first three instances of each character from the collected data were used to train the system. Now, the data was divided into two groups namely the training data and the testing data. The training data contained the first three instances of each of the 33+1 characters and the rest of the instances are made to be part of the testing data.

Two types of experiments were conducted for each writer. The first experiment type was performed by utilizing the training data itself as testing data. The second experiment was conducted by using the real testing data. Note that the two experiments are different in the type of data used. In the first experiment, the engine was provided data that it has seen previously. The results of the experiments are presented in Tables 6.1 and 6.2.

Experiment No.	Writer	Data Used	Accuracy
1	Writer A	Training data	99.7%
2	Writer A	Testing Data	99.4%
		Average	99.55%

Table 6.1. Experiment Results for Writer A

Experiment No.	Writer	Data Used	Accuracy
1	Writer B	Training data	99.5%
2	Writer B	Testing Data	99%
		Average	99.25%

Table 6.2. Experiment Results for Writer B

6.2. Performance of the Classifier's Layers

The final outputs of the recognizer are tested by the experiments discussed in the last section. From the same experiments, the performance of the three layers of the recognizer is also analyzed as follows.

On the average, only about 16% of the characters are passed to the second layer. However, this doesn't necessarily mean that all of these characters were not recognized correctly by the first layer. But, it means that the recognizer is not sure about the results proposed by the first layer. For writer A, from the characters that are not passed to the second layer (the result is reported based on the proposal of only the first layer), 99.8% of them are correctly recognized. Conversely, from the characters that are passed to the second layer, 61.3% of them were correctly recognized. In summary, the recognition accuracy of the first layer (coarse classification layer) is up to 93.9% for writer A. A similar analysis has shown that the recognition accuracy of the same layer for writer B is 91.9%.

To determine the accuracy of the second layer (detailed matching), it suffices to see how many of the characters passed to it are correctly recognized. We have noticed that 99% of them are correctly recognized for both writers.

Characters are passed to the third layer only in rare cases. For example, only 1% of the total characters are made to pass to the third layer for writer A. Actually, the number of characters that has been given to the third layer are greater in number for writer B. This is because writer B writes different characters with the same number of strokes in the same order. This situation increases the probability of passing the characters written by Writer B to the third layer. Whatsoever, characters passing to the third layer are fewer in number as compared to the total number of characters that are tested. After the computationally complex transformation and distance computation, the third layer is able to identify 99.8% of the characters correctly (see Table 6.3).

Layer	Writer A	Writer B	Average Accuracy
Coarse Classification	93.9%	91.9%	93.4%
Detailed Matching	99%	99%	99%
Superimposing Matching	99.8%	99.7%	99.8%

Table 6.3. Performance of the layers of the recognizer.

6.3. Discussion

It is reminded that this is the first attempt to solve the problem of online handwriting recognition for Ethiopic characters. This necessitates conducting a study on the various approaches used to solve this problem for other character sets in relation to the nature of the Ethiopic characters. To this end, we have also studied the shape of Ethiopic characters.

Online handwriting data collection is the primary step and it is accomplished by digitizer software named *MovAlyzer* using the mouse as an input device. This software collects pen down and pen up data points along the path of the mouth. A stroke is the sequence of data points between a pen down and a pen up. Thus, the character can be defined as group of strokes in the order they appear while the character is written where the strokes in the are separated by set of pen up points which are sampled by *MovAlyzer* while the user lift up the mouse to draw another stroke. This data is presented to the recognition engine where part of it is utilized for training and the rest for testing purpose.

Regardless of the usage of data (for training or testing), it is first presented to the preprocessing module in which different preprocessing activities that help in reduction of data and reduction of variations are performed. The feature extraction module takes the preprocessed data to represent each stroke is in terms of X and Y observation code sequences. Then, the set of strokes in a character represented in terms of X and Y observation code sequences are stored in a text file in which their order is maintained.

While training, the training algorithm collects X and Y observation code sequences of strokes in a character from the sample data and creates a reference file. The major content of this file is the collected X and Y observation code sequences from the sample data.

Classification is accomplished by using the three-layered recognizer module. The first layer is the coarse classification layer in which inter-strokes distances are used to match characters. In the second layer, detailed matching is performed by using detailed observation code sequences. The last layer, namely the superimposing layer, uses a mechanism of comparing two characters by superimposing one on another and finding inter-point distances to measure the matching

The experimental results show that the pattern representation scheme, the algorithms that we have designed for the various steps in the recognition process and the three-layered structured recognizer together make most of the recognition tasks successful. Another great discovery in this research is only small amount of data is needed to train the system and this is an advantage to users. On top of this, the layered structure of the recognizer facilitates use of resources efficiently by allowing processing computationally complex algorithms implemented in lower layers for only ambiguous results of upper layers.

According to an analysis that has been done on Ethiopic character set, it is concluded that the majority of non-basic characters are formed by adding secondary strokes on the basic characters. Some others are formed by modifying the shapes of the basic characters based on different rules except some exceptional cases. Thus, we proposed a strategy to approach the problem by first solving the online character handwriting problem for the basic characters. Accordingly, we narrowed down our work to consider only the 33+1 basic characters. We strongly believe that the system can be extended to include the non-basic characters though we didn't do that due to time constraint. Though it seems that this is a series limitation, we justified that the non-basic characters could be recognized based on the basic characters as this strategy has been applied for the construction of other systems such as the Ethiopic/Amharic typewriter keyboard layout and the phonetic Ethiopic text entry system on computer keyboards.

In its easy form, extending the system will not be different from adding model characters and train the system with them. This means that the system could be trained with instances of the 231+7 Ethiopic alphabetic letters to incorporate the remaining non-basic characters. However, this approach should not be directly pursued as it may make the system inefficient both in memory requirement and speed. Thus, additional techniques to avoid such consequences must be considered. Some of these are consideration of the structural relationship between strokes, formation of an efficient structure to store the available strokes and their order to form characters (so as to minimize the number of set of reference sequences stored for similar strokes that occur in different characters).

Another limitation of the whole approach is that the approach is stroke number and stroke order dependent. For writer-dependent systems, this is tolerable. In the case of writer-independent systems, this causes to have many more reference sequences of characters

which might widen the search space and consequently reduce the efficiency of the recognition system. This situation might also increase the amount of training data needed to train the system. But for our consideration, we believe that it is not a big problem.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORKS

In broader sense, online handwriting character recognition is the process of finding out what character is represented by a given online handwritten character data. Online handwriting character data is, in general, a set of the coordinate values of data points that are sampled along the trajectory of the character as it is written. The difficulty of identifying what character is actually represented by the data set start from the fact that identical set of data points will not be obtained for different occurrences of the same character. Thus, pattern representation scheme that produces similar representations for different occurrences of the same character solve the problem of character classification greatly.

We have introduced a new online handwriting character data representation scheme, which we have proved that it models the pattern in a manner that creates similarity between different occurrences of the same character. Major contributions of this work are outlined below:

- The nature of the shapes of Ethiopic characters is analyzed and a strategy to approach the problem of online handwriting recognition for Ethiopic characters is proposed. This strategy is to first concentrate only on the basic characters since it would be reasonably possible to extend the system to incorporate the non-basic characters. This is evidently logical, as we have shown that the shapes of most of the non-basic characters are derived by applying different modifications on the shapes of the basic characters.
- An online handwriting recognition engine for Ethiopic basic characters is designed.
- Algorithms for preprocessing activities are designed and implemented. These include extra point noise elimination, size normalization, filtering and resampling algorithms.
- A new method of online handwriting pattern representation scheme, which makes use of X and Y observation code sequences derived from the data is introduced and algorithms for generating these sequences are designed and implemented. This

constitutes the feature extraction process, which is considered as the most important factor in determining the classification result.

- Training algorithm by which the engine is trained is also designed and implemented. This algorithm takes the observation code sequences extracted from the sample characters in the training data and produces a reference file that systematically stores the reference observation code sequences. The training algorithms needs relatively small amount of data to train the writer-dependent recognition system.
- Most importantly, a three-layered recognizer is designed and implemented. The recognizer has three layers namely coarse classification, detailed matching and superimposing matching, through which unknown character passed if needed. The logical structure employed by the recognizer attempts to pass a to be recognized character to a lower layer if and only if the character is not surely identified by the current layer. We have done this to minimize the inaccuracy of the engine.

As it is only the beginning, future works are expected. We suggest the following:

- Improve the approach to make it stroke number and order independent.
- Extend the recognition engine to incorporate the non-basic Ethiopic characters.
- Extend the system to incorporate word-level and/or sentence-level recognition.
- Extend the recognition engine to be writer-independent.

References

1. Anil K., Jain, Robert P.W., Duin, Jianchang Mao: "***Statistical Pattern Recognition: A Review***", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 1, January 2000.

2. Jorma Laaksonen, "***Subspace Classifiers in Recognition of Handwritten Digits***", (Thesis for the degree of Doctor of Technology), Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory of Computer and Information Science, May 1997.

3. Palmondon, S.N. Srihari: "***Online and Offline Handwriting Recognition: A Comprehensive Survey***", IEEE Transactions on pattern Analysis and Machine Intelligence, Vol. 22, No-1, 2000.

4. A. Drissman, "***Handwriting Recognition Systems: An Overview***", available from www.drissman.com/avi/school/HandwritingRecognition.pdf

5. "***Pen-based Solutions and Handwriting Recognition***", HP Labs, Hewlett- Packard Development Company, L.P., 2004. Available in site: www.**hp**l.**hp**.com/india/research/lt-penhw-interfaces.html

6. Nigussie Tadesse: "***Handwritten Amharic Text Recognition Applied to the Processing of Bank checks***", (Masters Thesis). Addis Ababa University, School of Information Studies for Africa, Addis Ababa University, 2000.

7. Messay Hailemariam: "***Handwritten Amharic Character Recognition:The Case of Postal Addresses*** ", (Masters Thesis). Addis Ababa University, School of Information Studies for Africa, Addis Ababa University, 2003.

8. Wondwossen Mulugeta, ""***OCR for Special Type of Handwritten Amharic Text ("Yekum Tsifet"), Neural Network Approach*** (Masters Thesis). Addis Ababa University, School of Information Studies for Africa, Addis Ababa University, 2003.

9. Scott D.Connell, "***Online Handwritten Recognition Using Multiple Pattern Class Models***", (Doctor of Philosophy Dissertation), Department of Computer Science and Engineering, Michigan State University, 2000.

10. H. Shu, "***On-line Handwriting Recognition Using Hidden Markov Models***", (Master Thesis), Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1997.

11. G. Loudon, O. Pellijeff, Li zhong-wei, "***A Method for Handwriting Input and Correction on Smartphones***", , Ericsson Research, Cyberlab Singapore, 2000.

12. S. Jaeger, C.-L. Liu, M. Nakagawa, "***The state of the art in Japanese Online Handwriting Recognition Compared to Techniques in Western Handwriting Recognition***", International Journal on Document Analysis and Recognition, 2003.

13. C.L.Liu, S. Jaeger, M.Nakagawa, "***Online Recognition of Chinese Characters, The State-of-the-Art***", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 26, No. 26, 2004.

14. T.Klassen, "***Towards Neural Network Recognition of Handwritten Arabic Letters***"(Masters Thesis). DalHousie University, Faculty of Computer Science, 2001.

15. E. Gómez Sánchez, Y.A. Dimitriadis, M. Sánchez-Reyes Más, Sánchez García, J.M. Cano Izquierdo, J. López Coronado, "***Online Character Analysis and Recognition With Fuzzy Neural Networ***k", Intelligent Automation and Soft Computing, Vol. 7, No. 3, pp. 161-162, 1998.

16. Kam-Fai Chan, Dit-Yan Yeung "***Recognizing online handwritten alphanumeric characters through flexible Structural Matching***", The Journal of Pattern Recognition Society, October, 1998

17. W. Thimbleby, "***A Better Calculator: Processing Handwritten Mathematical Expressions to Solve Problems***", May, 2004

18. N. Joshi, "***Tutorial On Handwriting Character Recognition***", Biomedical Lab, Department of Electrical Engineering, Nov.2003

19. M.Aksella, "***Handwritten Character Recognition: A Palmtop Implementation and Adaptive Commmitte Experiments***"(Masters Thesis). Helsinki University of Technology, Department of Engineering Physics and Mathematics, May 2000.

20. Ignace Jay Gelb, R. M. Whiting, "***Writing***", Microsoft Encarta Reference Library 2004, Microsoft Corporation.

21. Robert A. Fradkin, "***Alphabets***", Microsoft Encarta Reference Library 2004, Microsoft Corporation.

22. Alwiya S. Omar, "***African Languages***", Microsoft Encarta Reference Library 2004, Microsoft Corporation.

23. "***Ethiopic Script and Language***", Microsoft Encarta Reference Library 2004, Microsoft Corporation.

24. C.C. Tappert, C.Y. Suen, T.Wakahara, "***The State of the Art in Online Handwriting Recognition***", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, August, 1990.

25. A.Lenaghan, R. Malyan and G.A Jones***, "Matching Structural Descriptions of Handwritten Characters Using Heuristic Graph Search",*** Kingston University, School of Computer Science and Electronic Systems.

26. "***Pen Technologies***", IBM Pen Technologies, Available from "www.research.ibm.com/electricInk/"

27. K. Chan, D. Yeung, "***PenCalc: A Novel Application of Online Mathematical Expression Recognition Technology***", Hong Kong University of Science and Technology, Department of Computer Science.

28. S.S.Tang, I.Methaste, "***Thai Online Handwritten Character Recognition Using Windowing Backpropagation Neural Networks***", Information Research and Development Division, National Electronics and Computer Technology Center, National Science an Technology Department Agency, Rachathewi, Bangkok, Thailand, available from www.links.nectec.or.th/ uploads/upload/040336/onlineBNN.pdf

29. P. Burrow, "***Arabic Handwriitng Recognition***", (Masters Thesis), School of Informatics, University of Edinburgh, 2004.

30. D. Perry, ***"Handheld Computers (PDAs) in Schools"***, Becta ICT Reseach, March 2003.

31. R. Kassel, "***A comparison of approaches to on-line handwritten character recognition***". Ph.D. Thesis, MIT Department of Electrical Engineering and Computer Science, June 1995.

32. T. Starner, J. Makhoul, R. Schwartz, and G. Chou, "***On-Line Cursive Handwriting Recognition Using Speech Recognition Methods***," presented at International Conference on Acoustics, Speech, and Signal Processing, 1994.

33. F. Kubala, A. Anastasakos, J. Makhoul, L. Nguyen, R. Schwartz, and G.Zavaliagkos, "***Comparative Experiments on Large Vocabulary Speech Recognition***," presented at International Conference on Acoustics, Speech, and Signal Processing, 1994.

34. Jaeger S, Manke S, Reichert J, Waibel A, "***Online Handwriting Recognition: the Npen++ Recognizer***", International Journal on Document Analysis and Recognition, 2001.

35. D. Paul, "***The Design for the Wall Street Journal-based CSR Corpus***," presented at Proceedings of the DARPA Speech and Natural language Workshop, 1992.