Efficient Pre-processing, Feature Extraction and Post-Processing Algorithms for Recognition of Online Handwritten *Gurmukhi* Script

A thesis submitted in fulfillment
of the requirements for the award of the degree of

DOCTOR of PHILOSOPHY

Submitted by

Ravinder Kumar
(Registration No. 950903012)

under the supervision of

Dr. Rajendra Kumar Sharma
Professor,
Thapar University,
Patiala

Dr. Anuj Sharma
Assistant Professor,
Panjab University,
Chandigarh

Computer Science & Engineering Department
Thapar University, Patiala–147004 (Punjab) India
July, 2015

Dedicated

to

My family

CERTIFICATE

I, **Ravinder Kumar** hereby certify that the work which is being presented in this thesis entitled "**Efficient Pre-processing, Feature Extraction and Post-Processing Algorithms for Recognition of Online Handwritten Gurmukhi Script**", in fulfillment of requirements for the award of degree of the **DOCTOR OF PHILOSOPHY** submitted in the Computer Science & Engineering Department (CSED), Thapar University, Patiala, is an authentic record of my own work carried under the supervision of **Dr. Rajendra Kumar Sharma** (Professor, CSED, Thapar University, Patiala) and **Dr. Anuj Sharma** (Assistant Professor, DCSA, Panjab University, Chandigarh).

The matter presented in this thesis has not been submitted either in part or full to any other University or Institute for the award of any degree.

Ravinder Kumar

(Ravinder Kumar)

Date: July 20, 2015

It is certified that the above statement made by the candidate is correct to the best of our knowledge and belief.

RK Sharma 20/7/2015

(Dr. R. K. Sharma)
Professor,
Computer Science and Engineering Department,
Thapar University, Patiala
PIN –147004 (INDIA).
Supervisor

Date: July 20, 2015

Anuj Sharma

(Dr. Anuj Sharma)
Assistant Professor,
Department of Computer Science and Applications,
Panjab University, Chandigarh
PIN –160014 (INDIA).
Supervisor

ACKNOWLEDGEMENT

"I wish to thank you from the bottom of my heart but for you my heart has no bottom"

Sometimes words get limited when it comes to express deep and hearty regards for an inspirational experience of lifeand finalizing this research is one of those beautiful moments of my life. I feel myself lucky to reach the milestone which I desperately wanted and for which I was giving my committed efforts since a long time.

I am endeavoring to express my deep thanks to almighty as whatever I am today isbecause God always answered my sincere prayers and bestowed me with contentment, satisfaction and courage to solve the complexities of life. I just pray to him to ignore my mistakes and be with me throughout my life. Thanks for always standing by me whenever I need you. May your name be exalted, honored and glorified *"Jai Mata Di"*.

I warmly acknowledge assorted contributions of my revered supervisor *Dr. R. K. Sharma*, Professor, Department of Computer Science and Engineering, Thapar University, Patiala as his unflinching encouragement and thorough involvement has nourished my intellectual level which I will always relish. I will also like to acknowledge the valuable guidance of *Dr. Anuj Sharma*,Assistant Professor, Department of Computer Science & Applications, Panjab University, Chandigarh that helped me bringing refinement in this research work. I am also grateful to *Dr. Prakash Gopalan*, Director, Thapar University, Patiala for providing me required resources and stimulating learning environment

I feel myself fortunate to have truly understanding wife *Dr. Nidhi Walia* whom I owe my loving thanks for her unflagging love & support throughout this research. Wholeheartedly I wish to admit that without her encouragement this research would have never taken this shape as she is the one who realize me the joy of intellectual pursuit.My deep love and thanks to my kids*Jeenu* and *Vardaan*forhaving a lot of patience and a deep regret for some time missing their childhood.

I am indebted to my parents*Sh. Rajinder Kumar*and *Smt.JanakJulka*for being exceptionally supportive that paved the way for a privileged education as like a perfect parents they always stood beside me and encouraged me constantly since my childhood.I would also like to acknowledge help of colleagues and friends at Thapar University, Patiala as a healthy discussion with them helped me identify the broad problems concerning my research area. It is indeed a pleasure to thank one and all who participated and contributed in this research in some way.

Ravinder Kumar

ABSTRACT

Human beings start recognizing digits, letters and symbols in very early stage of their childhood. This tendency of human beings to recognize alphabets/characters develops rapidly with the aging process. Though this natural process of recognizing alphabets or objects by human beings is taken for granted but the complexity of recognition system is realized when task of teaching the same is performed on a machine. Online Handwriting Recognition (OHWR) system is one such area of research that has gained world-wide attention of researchers. One main reason attributed to this popularity is that recognition of online handwriting has a wide range of applications at the interface between man and machine which ultimately facilitates easy way of giving inputs to the computer.

Recognition of online handwritten character for any script is a difficult task due to problems posed by different handwriting styles of different writers, complexity of content and different handwriting devices used by user. Extensive research in the area of handwritten characters has made it feasible to recognize characters of English, Chinese, and Japanese language. However, less attention has been paid to recognition of Indian languages, especially, regional languages. A good number of methods have been proposed for recognition of handwritten English characters. However, the development of technologies is not at the same level for Indian scripts, especially *Gurmukhi* script which is complicated in terms of its structure and writing style.

Hence, an attempt has been made in this thesis to provide online handwriting recognition system for *Gurmukhi* script. Main objective of this research was to propose efficient pre-processing, feature extraction and post-processing algorithms by introducing new constraints/ steps/ parameters for the recognition of online handwritten *Gurmukhi* script. Further, to validate the proposed algorithms targeting about 95% accuracy at character level. For this purpose, a stroke based approach has been adopted for developing a recognition engine for online handwritten *Gurmukhi* script. This approach made use of all standardized steps, *viz.* pre-processing, feature extraction, recognition and post-processing as required for designing a handwriting recognition engine.

The preliminary examination in this area made it clear that no public online handwritten *Gurmukhi* character database was available to carry out recognition experiments. Hence, creating a reliable and authentic database became a part of the current research work. Raw data for online handwritten *Gurmukhi* words have been collected from 144 users residing in Punjab. These users were categorized on the basis of their level of proficiency in writing *Gurmukhi* script and were put in four categories. The handwritten words collected from different writers have been formulated using different combinations of 35 characters, 9 vowel modifiers, 3 nasals, and 10 numerals of *Gurmukhi* script. A total number of 39,871 sample words of *Gurmukhi* script were collected from different category of users. From these collected samples 2,28,442 strokes have been collected. Out of collected strokes, total 2,26,330 strokes have been annotated. All those strokes which are identified and annotated are further used for training of LibSVM classifier. Experiment results have been obtained by using the open source software LibSVM. For LibSVM classification, Radial Basis Function (RBF) kernel has been used.

In this work, efficient pre-processing, feature extraction and post-processing algorithms for recognition of online handwritten *Gurmukhi* script have been proposed. These algorithms shall lead to the development of an efficient WI, open-vocabulary and cursive writing recognition system.

In pre-processing, a new methodology of association of strokes has been proposed. Here, association between different strokes has been targeted to form a character based on positional coordinates and overlapping windows of strokes. A set of features that represents a stroke of handwritten *Gurmukhi* script has been defined and analyzed. Cross-validation has been used in this research work at 2-fold, 3-fold, 4-fold and 5-fold for three engines, *viz.* Engine A, Engine B, and Engine C. New algorithms for post-segmentation of stroke sets, sequencing of stroke sets and merging the stroke sets for post-processing in handwritten *Gurmukhi* script recognition have been presented. Application of SVM has achieved excellent recognition results for various pattern recognition problems. Under current study, 95.6% recognition accuracy for *Gurmukhi* characters has been achieved. An overall recognition accuracy of 96.8% has been achieved for *Gurmukhi* numerals.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BDD	Background Directional Distribution
DCT	Discrete Cosine Transform
DTW	Dynamic Time Warping
HMM	Hidden Markov Model
IME	Input Method Editor
ISR	Integrated Segmentation and Recognition
MQDF	Modified Quadratic Discriminant Function
OHWR	Online Handwritten Recognition
OOV	Out of Vocabulary
PCA	Principal Component Analysis
PDA	Personal Digital Assistants
RBF	Radial Basis Function
ROC	Receiver Operating Characteristics
SDNN	Space Displacement Neural Network
SOISR	Self Organizing Integrated Segmentation and Recognition
SVM	Support Vector Machine
WD	Writer Dependent
WI	Writer Independent
XML	Extensible Markup Language

CHAPTER 1

MOTIVATION AND OBJECTIVES

1.1 Motivation

Presence of updated and sophisticated technology has changed the working style of human beings and made them comfortable in different arena. Immense use of technology in addressing different problems demands refined technology that may ease human computer interface. Natural handwriting of user is an easy way to exchange information between machine and user. World-wide researchers and academicians have indulged in this area for more than four decades whereby they have endeavored to look for different ways where a user can directly interact with computers instead of giving input through keyboard or mouse as these devices have got their own limitations. A natural way of communicating with computers that may facilitate instant command to computer system is giving inputs through user's own speech or input in the form of online handwriting. Though recognition systems have achieved an edge yet there are some challenging issues that need refinement in existing systems such as developing a speech recognition system becomes challenging in noisy environment. Similarly, developing handwriting recognition system is complex in terms of multiple languages available world-wide and variations in handwriting style of users.

The challenges in developing Online Handwriting Recognition (OHWR) systems are generally due to user's style of holding pen, sentiments expressed by him during writing, speed of writing, surface on which script is written and variations in handwriting style. Many organizations have entered into this venture where they have introduced new products like Newton^{TM} from Apple; Personal Digital Assistants (PDA) from AT&T and HP; and Google's IME (Input Method Editor) API that facilitate online handwriting recognition technology. During the last two decades advancements in digital technology and its applications in different arena initiated a new wave of interest among researchers for improving handwriting recognition systems. In this direction many achievements have been reported where high accuracy has been achieved in presence of few constraints that

included writing style, small vocabulary size and writer dependency. Although these constraints can help in improving handwriting recognition process considerably but an efficient recognition system should involve the development of a technology that can work in an environment which is free from all kind of constraints. Recognizing the potential of online handwriting recognition technology for improving man-machine interaction, present research work was initiated. This research study has been undertaken to develop an efficient recognition system for online handwritten *Gurmukhi* script that can perform its task in real time. As such, present research work aims to take handwritten *Gurmukhi* character as input, process the input, train the Support Vector Machine (SVM) to recognize the input and get the results in the form of Unicode version of input character.

1.1.1 Applications of online handwriting recognition systems

Online handwriting recognition systems have a wide range of applications. Some of these applications are: online form filling in census data collection and land acquisition departments, dictation for official work in state government offices, teaching of a language in educational institutes, messaging and news, publication and writing, and online compliant filling for grievance handling in legal matters. This is worth mentioning here that continuous efforts to improve the recognition accuracy and usability of handwriting recognition system is required because of ever increasing use of computer systems in various public and private sectors. A sophisticated online recognition system with enhanced accuracy should give instant response with the help of system user interface.

1.2 Objectives of research work

The main objective of this research work had been to propose a writer independent online handwriting recognition system for *Gurmukhi* script. This research work was initiated with the following broad objectives:

(i) To create an annotated dataset of 30,000 words of *Gurmukhi* script (including 2000 unique words) for training and testing the OHWR engine.

(ii) To propose efficient pre-processing, feature extraction and post-processing algorithms by introducing new constraints/ steps/ parameters for the recognition of online handwritten *Gurmukhi* words and numbers.

(iii) To validate the proposed algorithms using existing recognition algorithms targeting about 95% accuracy at character level.

(iv) To develop a system for recognition of online handwritten *Gurmukhi* words and numbers for its demonstration to the potential users.

1.3 Overview of *Gurmukhi* script

Gurmukhi script is used to write Punjabi language which is a widely spoken language of northern India. This language is spoken as a native language by over 2.73% Indians. Punjabi is also the official language of the Indian state of Punjab.

Gurmukhi script has many features similar to other popular Asian scripts such as Devanagari and Bangla. *Gurmukhi* script consists of 35 basic characters, 6 special consonants with /*pairi bindí*/, 9 vowel modifiers called /*mátrás*/, two symbols for nasal sounds /*bindí* /and /*ṭipí*/, and one symbol which duplicates the sound of a long consonant /*adhak*/ and 10 numerals (Nagra, 1988; Singh, 1995; Bahri, 2011).

Some of the features of *Gurmukhi* script are:

- *Gurmukhi* script is cursive and written in left to right direction with top down approach.
- Most of the characters of *Gurmukhi* script have a horizontal line called head line at the upper part. Characters in a word are connected by this head line, so there is no vertical inter character gap in the letters of the word.
- In online handwritten *Gurmukhi* script text, stroke is considered as the smallest unit. A valid combination of strokes forms a *Gurmukhi* character/vowel. Further, combination of character and vowel form akshara or word.
- A word in the *Gurmukhi* script can be divided into three horizontal zones, *viz.* upper zone, middle zone and lower zone. The upper zone denotes the region above the head line where some of the vowels reside. The middle zone is the

eventful zone that represents the area below the head line, where consonants and some part of vowels are present. The last zone represents the area below middle zone where some vowels and certain half characters lie in the foot of consonants. A statistical analysis of Punjabi corpus has shown zone-wise percentage distribution of *Gurmukhi* symbols in printed text as: upper zone (25.39%), middle zone (70.41%), and lower zone (4.20%) (Lehal and Singh, 2001). Figure 1.1 illustrates these three zones of a *Gurmukhi* word.

Gurmukhi Writing Zones

Upper zone: It contains headline, sub part of vowels /*sihárí*/, /*bihárí*/ and complete vowels /*lãĩ*/, /*duláwãĩ*/, /*hoṛá*/, /*kanāūṛá*/, /*ṭipí*/, and /*adhak*/.

Middle zone: It contains all consonants and sub part of vowels /*sihárí*/, /*bihárí*/ and one complete vowel /*kanná*/.

Lower zone: It contains two vowels /*āūnkaṛ*/, /*dulāinkṛe*/ and three symbols called /*pairian*/ characters /*háhá*/, /*rárá*/, and /*vává*/ to form conjunct consonants.

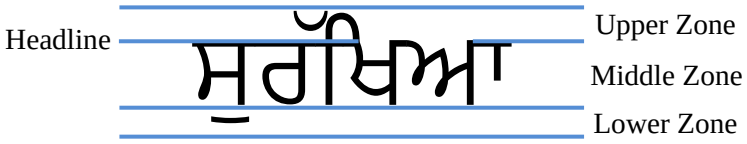


Figure 1.1: Three writing zones of *Gurmukhi* script

Gurmukhi character set

As mentioned above, *Gurmukhi* script contains 35 characters (three vowel bearers and thirty two consonants). Table 1.1 depicts these characters. The vowel bearers are expressed in bold face in this table.

Table 1.1: *Gurmukhi* Character set

ੳ	/***ooṛá***/	ਅ	/***āiṛá***/	ੲ	/***íṛí***/	ਸ	/*sassá*/	ਹ	/*háhá*/
ਕ	/*kakká*/	ਖ	/*khakkhá*/	ਗ	/*gaggá*/	ਘ	/*kaggá*/	ਙ	/*ṅaṅá*/
ਚ	/*chachchá*/	ਛ	/*chhachhá*/	ਜ	/*jajjá*/	ਝ	/*cajhá*/	ਞ	/*náñá*/
ਟ	/*ṭāīnká*/	ਠ	/*ṭhaṭhṭhá*/	ਡ	/*ḍaḍḍá*/	ਢ	/*taḍhá*/	ਣ	/*ṇáṇá*/

ਤ	/*tattá*/	ਥ	/*thaththá*/	ਦ	/*daddá*/	ਧ	/*tadhá*/	ਨ	/*nanná*/
ਪ	/*pappá*/	ਫ	/*phaphphá*/	ਬ	/*babbá*/	ਭ	/*pabhá*/	ਮ	/*mammá*/
ਯ	/*yayyá*/	ਰ	/*rárá*/	ਲ	/*lallá*/	ਵ	/*vává*/	ੜ	/*ṛáṛá*/

There are six additional consonants created by placing a dot /*bindí*/ at the foot /*pairi*/ of the consonant called /*pairi bindí*/ which are given in Table 1.2.

Table 1.2: *Gurmukhi* consonants with /*pairi bindí*/

ਸ਼	ਖ਼	ਗ਼	ਜ਼	ਫ਼	ਲ਼
/*sassá*/ pairi bindi	/*khakkhá*/ pairi bindi	/*gaggá*/ pairi bindi	/*jajjá*/ pairi bindi	/*phaphphá*/ pairi bindi	/*lallá*/ pairi bindi

Gurmukhi vowel modifiers

Table 1.3 provides the list of ten *Gurmukhi* vowels. *Gurmukhi* supports two forms of vowels: independent form and dependent form. Independent vowels do not require a consonant, dependent vowels require it. All consonants use dependent form of vowel.

Table 1.3: *Gurmukhi* vowel modifiers

Independent	ਅ	ਆ	ਇ	ਈ	ਉ	ਊ	ਏ	ਐ	ਓ	ਔ
Dependent	NA	◌ਾ	◌ਿ	◌ੀ	◌ੁ	◌ੂ	◌ੇ	◌ੈ	◌ੋ	◌ੌ
	/mukta/	/kanná/	/sihárí/	/bihárí/	/āūnkar/	/dulāinkre/	/lāṁ/	/dulāwāṁ/	/horā/	/kanāūṛā/
with consonant /*lallá*/	ਲ	ਲਾ	ਲਿ	ਲੀ	ਲੁ	ਲੂ	ਲੇ	ਲੈ	ਲੋ	ਲੌ

Nasalisation: /*ṭipí*/ and /*bindí*/

/◌ੰ/ /*ṭipí*/ and /◌ਂ/ /*bindí*/ are used for producing the velar nasal. These are used as superscripts in the upper zone.

Gemination: /*adhak*/

The use of /◌ੱ/ /*adhak*/ indicates that the following consonant is geminate. While writing a particular character /*adhak*/ is introduced as superscript in the upper zone and is used between the two consonants.

***Gurmukhi* Numerals**

Ten numerals used in *Gurmukhi* script are presented in Table 1.4.

Table 1.4: Numerals in *Gurmukhi* script

੦	੧	੨	੩	੪	੫	੬	੭	੮	੯
ਸਿਫਰ	ਇੱਕ	ਦੋ	ਤਿੰਨ	ਚਾਰ	ਪੰਜ	ਛੇ	ਸੱਤ	ਅੱਠ	ਨੌਂ
/sifár/	*/ikk/*	*/do/*	*/tinn/*	*/chár/*	*/pánj/*	*/chhe/*	*/sátt/*	*/áṭh/*	*/nãũ/*
Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine

1.3.1 Standard way of writing the *Gurmukhi* script characters

A *Gurmukhi* character is written in a particular pattern. It means that the pen movement has to be in a specific order to form a character. Table 1.5 provides detail on required pen movements in order to form each *Gurmukhi* character. Dot (•) shown in each character describes the initial point for pen movement for writing that particular character.

Table 1.5: Correct way of writing the *Gurmukhi* character

1.3.2 Difficulties in recognition of online handwritten *Gurmukhi* script

In handwritten *Gurmukhi* script, one can use head line to join two or more characters. This head line needs to be segmented for associating this to the relevant characters. This problem of segmentation in *Gurmukhi* script is entirely different from other common scripts such as Latin, Phags-pa, and Arabic. In Latin script, each character composing a word does not share the pixels in horizontal direction. But, in *Gurmukhi* script, two or more characters/symbols of a word may share the pixels in horizontal direction. Following are some of the difficulties that arise in recognition of *Gurmukhi* script:

- A writer may not make the use of head line to join the characters as their writing style allows them to avoid its use, as shown in Figure 1.2.

(a) (b)

Figure 1.2: Illustration of *Gurmukhi* word /ਕਬੂਤਰ/ written (a) without head line, (b) with head line

- Handwriting style of a user affects the number of strokes one uses to form a character/ akshara/ word. Some individual uses one stroke to write a character whereas others may use multiple strokes to write the same character as shown in

Figure 1.3 for the *Gurmukhi* akshara /ਲੀ/. *Gurmukhi* script consists of a good number of such characters with a high degree of variation.

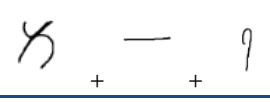
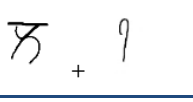
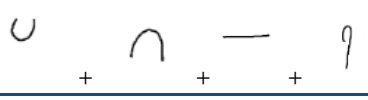
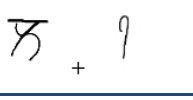
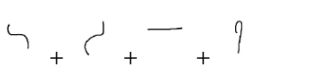
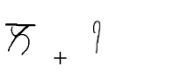
Strokes	Grouping of strokes	Akshara
		
(a)		
		
(b)		
		
(c)		

Figure 1.3: Illustration of grouping of single, and multiple strokes for formation of *Gurmukhi* character /ਲ/. (a) The akshara /*lee*/ /ਲੀ/ /*lallá + bihárí*/ composed of single character /*lallá* /, and /*bihárí*/. (b) and (c) The akshara /lee/ /ਲੀ/ /*lallá + bihárí*/ formed with multiple strokes for character /*lallá* / and single stroke for /*bihárí*/

- One other difficulty associated with recognition of *Gurmukhi* script is that there exists characters in this script that are similar in their shape. Table 1.6 portrays the combinations of such *Gurmukhi* characters. In *Gurmukhi* script, there are six consonants which are just distinguished by a dot /*pairi bindi*/ in the lower zone. A minor variation or noise in writing this dot /*pairi bindi*/ will make it difficult to identify the correct character written (Singh *et. al.*, 2013).

Table 1.6: Similar characters in *Gurmukhi* script

1.	ਖ and ਖ਼	2.	ਗ and ਗ਼
3.	ਸ and ਸ਼	4.	ਫ and ਫ਼
5.	ਲ and ਲ਼	6.	ੳ and ੳ
7.	ਨ and ਠ	8.	ਅ and ਘ
9.	ਤ, ੜ, ਡ and ਭ	10.	ਢ, ਚ, ਦ and ਟ
11.	ਪ, ਖ, ਥ and ਧ	12.	ਜ and ਜ਼
13.	ਮ and ਸ	14.	ਵ and ੲ
15.	ਹ and ਰ	16.	ਥ and ਬ
17.	ਟ and ੲ		

- One more problem associated with *Gurmukhi* handwriting recognition is slanted writing style of a user. Figure 1.4 describes how a particular word in *Gurmukhi* script is written in slanted way by a user which poses the problem in recognition process.

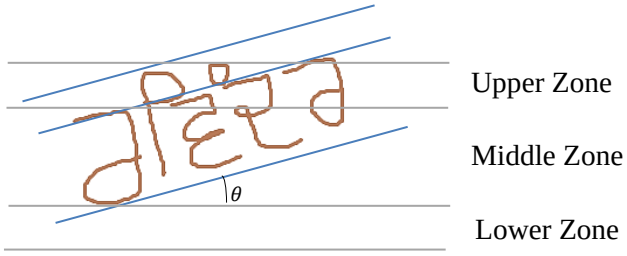


Figure 1.4: Illustration of slant handwriting

- The problem of sequencing and merging also arises in *Gurmukhi* script recognition. It is possible that all input strokes belong to same character. However, sequence of input may be different which creates difficulty in merging them to form the desired character. The *Gurmukhi* word /ਵਿਸਾਨ/, as shown in Figure 1.5, can be written in seven strokes. One can write the vowel modifier (ਿ) as the last stroke. This, however, should be merged with first stroke for correct recognition.

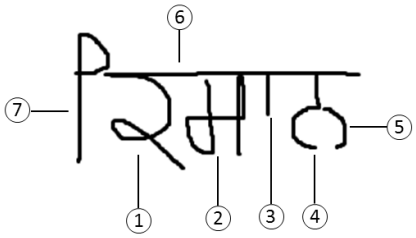


Figure 1.5: Illustration of sequencing problem in handwriting

- Some writers are habitual of writing *Gurmukhi* words in a pattern that they write the first character of the word significantly large and as they proceed further the size of next characters written keep on reducing. This problem of writing *Gurmukhi* script is presented in Figure 1.6.

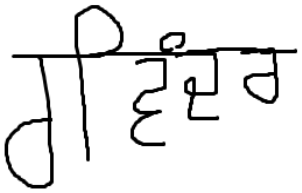
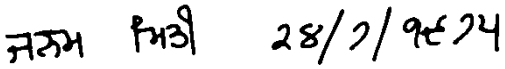


Figure 1.6: Gradual size reduction problem

- Yet another problem in *Gurmukhi* script recognition is that a given set of strokes may yield different characters. For example, /ਪ/ and /–/ will give /ਖ/ or /ਧ/.
- Handwriting is the consequence of a complex control process that is activated by muscle contraction. Different users have different styles of writing; even there are left hand users. These variations are geometrical in nature. The geometrical properties including position, size, aspect ratio, retraces, slant, and number of strokes can be associated with a character. Variations may also occur in writing due to material written at different times, different documents and even using different instruments. The difference between these variations is generally not very high and this mostly depends upon circumstances.

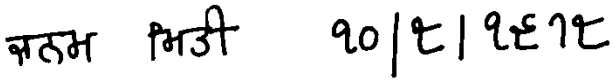
Figure 1.7 exhibits a few samples of alphanumeric characters of *Gurmukhi* script collected from different writers. It provides an overview of how an individual writes the same characters and numerals with variations and thus results into different handwriting styles.

Writer 1



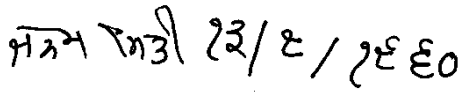
ਜਨਮ ਮਿਤੀ ੨੪ / ੭ / ੧੯੭੫

Writer 2



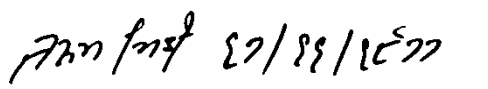
ਜਨਮ ਮਿਤੀ ੧੦ / ੮ / ੧੯੭੮

Writer 3



ਜਨਮ ਮਿਤੀ ੧੩ / ੮ / ੧੯੬੦

Writer 4



ਜਨਮ ਮਿਤੀ ੧੭ / ੧੧ / ੧੯੭੭

Writer 5

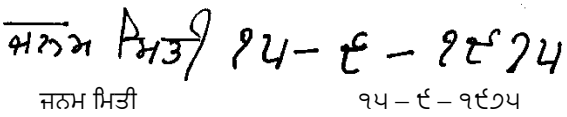


Figure 1.7: Samples of alphanumeric characters of *Gurmukhi* script taken from different writers

1.3.3 Classification of OHWR systems

Based on the training methodology, OHWR systems can be classified into two categories: Writer Dependent (WD) systems, and Writer Independent (WI) systems. The systems that are able to recognize handwritten text of the user(s) who were also used to train the system, are WD systems, and the systems that are able to recognize handwritten text of the user(s) who were not involved in the training process of the system are WI systems. Apparently, WD recognition systems have achieved higher recognition accuracy than WI recognition systems (Subrahmonia and Zimmerman, 2000).

Vocabulary is an important aspect that determines how accurately a handwriting recognition task can be completed. Closed-vocabulary recognition system refers to recognition of words which are available in a pre-determined dictionary. The dictionary size is arbitrary. Open-vocabulary recognition refers to recognition of words without the constraint of being in a dictionary which are also known as Out of Vocabulary (OOV) words (Vertanen, 2008). Recognition process is smoother and easier in case of closed-vocabulary systems when compared with open-vocabulary systems.

Handwriting process can further be constrained and we can restrict the space on which user writes. Constrained handwriting is boxed discrete and spaced discrete in nature. Unconstrained handwriting is cursive or mixed cursive in nature. In boxed discrete handwriting, each character is written inside a special box whereby, each character is written separately with space and no character touches other character. In cursive handwriting, characters in one word are connected and strokes are used more than once in individual character. It is a difficult task to recognize cursive handwriting due to large amount of variability. Each writer is having one's own speed of writing and uses different shapes to represent characters. Also, in cursive handwriting no clear boundaries are specified between characters to distinguish between them.

1.4 Contribution of the present research work

Present research work includes design and implementation of an online handwritten *Gurmukhi* script recognition system. The main contributions of this research work are:

- In this work, efficient pre-processing, feature extraction and post-processing algorithms for recognition of online handwritten *Gurmukhi* script have been proposed. These algorithms shall lead to the development of an efficient WI, open-vocabulary and cursive writing recognition system.
- In pre-processing, a new methodology of association of strokes has been proposed. Here, association between different strokes has been targeted to form a character based on positional coordinates and overlapping windows of strokes. Further, algorithms for normalization and centering have also been improved for strokes with low slope using scaling factor.
- A set of features that represents a stroke of handwritten *Gurmukhi* script has been defined and analyzed. These features are: positional features of a stroke, pre-processed coordinates, reversed pre-processed coordinates.
- In classification phase, SVM classifier has been used. A comparative analysis has been conducted for stroke recognition for 96 classes.
- New algorithms for post-segmentation, sequencing, and merging for post-processing in handwritten *Gurmukhi* script have been presented.
- Slanted strokes have been managed with the help of association rule.
- A methodology has been proposed for removal of noise and partial strokes present in online handwriting.
- Designed an efficient algorithm for online handwritten *Gurmukhi* character recognition that achieves a recognition accuracy of 95.6% for *Gurmukhi* characters.
- Real time post-processing has been implemented. For every new stroke, post-processor forms the new character instead of forming the character at the end of all input strokes.

1.5 Outline of thesis

The thesis on this work is divided into six chapters. A brief outline of these chapters is given below.

This chapter provides discussion on motivation behind the proposed research that worked as a driving force to pursue the study on recognition of online handwritten *Gurmukhi* script. Further, this chapter comprises of an overview of broad features of *Gurmukhi* script; major issues in handwriting *Gurmukhi* script; and the complexities in recognition process due to these issues. Finally, contribution of the present research work has also been presented.

In **second chapter**, a comprehensive review of literature on various stages involved in online handwriting recognition process has been presented. This chapter is divided into six sections. Section 2.1 contains the literature on the recent developments in data collection activity. Techniques involved in pre-processing are discussed in Section 2.2 of this chapter. Literature on feature extraction techniques from handwriting data is reviewed in Section 2.3. Different strategies for recognition process are reviewed in Section 2.4. The literature related to post-processing is presented in Section 2.5. Last section of this chapter, Section 2.6 provides a brief review on recognition accuracy results obtained in the field of handwriting recognition for various Indian scripts.

Third chapter explains methodology used and system designed for carrying out this research work. This chapter also explains handwritten data acquisition from 144 Punjabi writers. A total number of 39,871 words have been collected in this work. From these words, 2,26,330 number of strokes were identified and annotated. This chapter has been divided into four sections. Section 3.1 presents proposed framework, and also the novelty of this framework. Section 3.2 describes the process of collection of handwritten *Gurmukhi* words written by different users. The annotation process provides a list of strokes in *Gurmukhi* script along with their class, zone and characters which use that particular stroke. Section 3.3 introduces the SVM classifier, in brief. Section 3.4 describes the test data for recognition of characters, numerals, aksharas and words for *Gurmukhi* script.

In **Chapter 4**, a modified algorithm for normalization of horizontal and vertical bars has been proposed. This chapter also explains the association rules proposed in this work. An algorithm for finding the zone for a stroke based upon raw points has also been included in this chapter. This chapter is divided into five sections. Section 4.1 provides preliminary analysis of a stroke. Extraction of coordinates for input stroke and their overlapping with adjacent strokes has been discussed here. Section 4.2 explains proposed algorithms for association of strokes which is an important prerequisite before pre-processing phase. Three steps, *viz*. schema for stroke association, set of associated strokes and reductions of associated sets have been discussed in this section. Section 4.3 gives an overview of pre-processing phase. This section explains the phases of pre-processing, *viz.* removal of duplicate points, size normalization and centering, interpolating the missing points and resampling. Section 4.4 explains the zone identification technique for online handwritten *Gurmukhi* script. Finally, details of feature extraction techniques from online handwritten data have been discussed in Section 4.5.

In **Chapter Five**, recognition and post-processing phases for *Gurmukhi* script have been discussed. This chapter briefly explains the LibSVM used for recognition and proposed algorithms for post-processing. This chapter has been divided into six sections. Section 5.1 introduces and compares different LibSVM recognition engines. In this study, on the basis of writing zones of *Gurmukhi* script, strokes classes were divided into three zones, *viz.* upper, middle and lower zones. Three recognition engines have been designed and an accuracy of 98.7%, 96.7% and 100.0%, respectively, for these zones has been achieved. An accuracy of 95.3% could be achieved when we combine the classes in three zones using 5-fold cross validation. Section 5.2 describes the rule book pertaining to *Gurmukhi* characters, nasals, vowels and numerals formation with their possible combination of classes and corresponding weight. Section 5.3 includes proposed algorithms for post-segmentation of stroke sets, sequencing of stroke sets, and merging the stroke sets that help in final recognition process. Section 5.4 addresses the problem of ambiguity during the recognition process and how these issues have been resolved. A brief demonstration of recognition process about characters, numerals, aksharas and words formation has been provided in Section 5.5. An accuracy of 95.6% has been achieved in this work for *Gurmukhi* characters formation and accuracy of 96.8% has been achieved for numerals discussed in Section 5.6.

Chapter six contains the concluding remarks from the results obtained in this research work. Future scope and limitations of proposed research have also been discussed briefly in this chapter.

CHAPTER 2

REVIEW OF LITERATURE

With the advent of upcoming technology and increased demand for user friendly interfaces, handwriting recognition process has attracted the world wide researchers to propose new and efficient methods in this area. Most of the research work in this field has been done for English, Chinese and Japanese languages (Plamondon and Srihari, 2000; Nataneli and Faloutsos, 2007; Wang *et al.*, 2012). Designing a recognition system for handwritten characters of various indic and non-indic scripts has been discussed by various researchers that included Arabic, Persian, Bangla, Devanagari, Oriya, Gujrati and Kannada characters (Almuallim and Yamaguchi, 1987; Roy *et al.*, 2004; Pal *et al.*, 2007a; Rajashekararadhya and Ranjan, 2008; Bhattacharya and Chaudhuri, 2009; Alaei *et al.*, 2010; Rampalli and Ramakrishnan, 2011; Bhattacharya *et al.*, 2012). Among Indian scripts, a good amount of research work has been done for Devanagari script as it is the national language of India and spoken by a large population in India (Pal *et al*., 2007b). Some research work on handwritten *Gurmukhi* characters is also available in literature (Lehal and Singh, 2000; Sharma *et al.*, 2008; Jhajj and Sharma, 2010; Sachan *et al.*, 2011). This chapter provides a brief on existing research studies available in this field and the approaches used in these studies at different stages of handwriting recognition process.

The widely used process of recognizing online handwritten characters includes the sequential steps: data collection, pre-processing, feature extraction, segmentation, recognition and post-processing (Impedovo *et al.*, 1991; Chan and Yueng, 1999; Jaeger *et al.*, 2001; Zanibbi *et al.*, 2002; Koerich *et al.*, 2003; Sharma and Jain, 2010; Pardeep *et al.*, 2011). These standard phases of recognition process are carried out in such a way that output obtained from the previous phase becomes the input for the next phase. Figure 2.1 describes the architecture of handwriting recognition process which shows the standard phases of the handwriting recognition process. Literature on the procedure of data collection and annotation is discussed in Section 2.1. Methods involved in pre-processing are reviewed in Section 2.2 of

this chapter. Pre-processing includes the steps that are necessary to bring the input data into an acceptable form for feature extraction. Hence pre-processing of data provides the base for feature extraction. Literature on feature extraction for representation of online handwriting data are reviewed in Section 2.3. Approaches for recognition are reviewed in Section 2.4. Steps involved in post-processing are reviewed in Section 2.5. Last section of this chapter, Section 2.6, presents recognition accuracy results obtained in the field of handwriting recognition by various researchers.

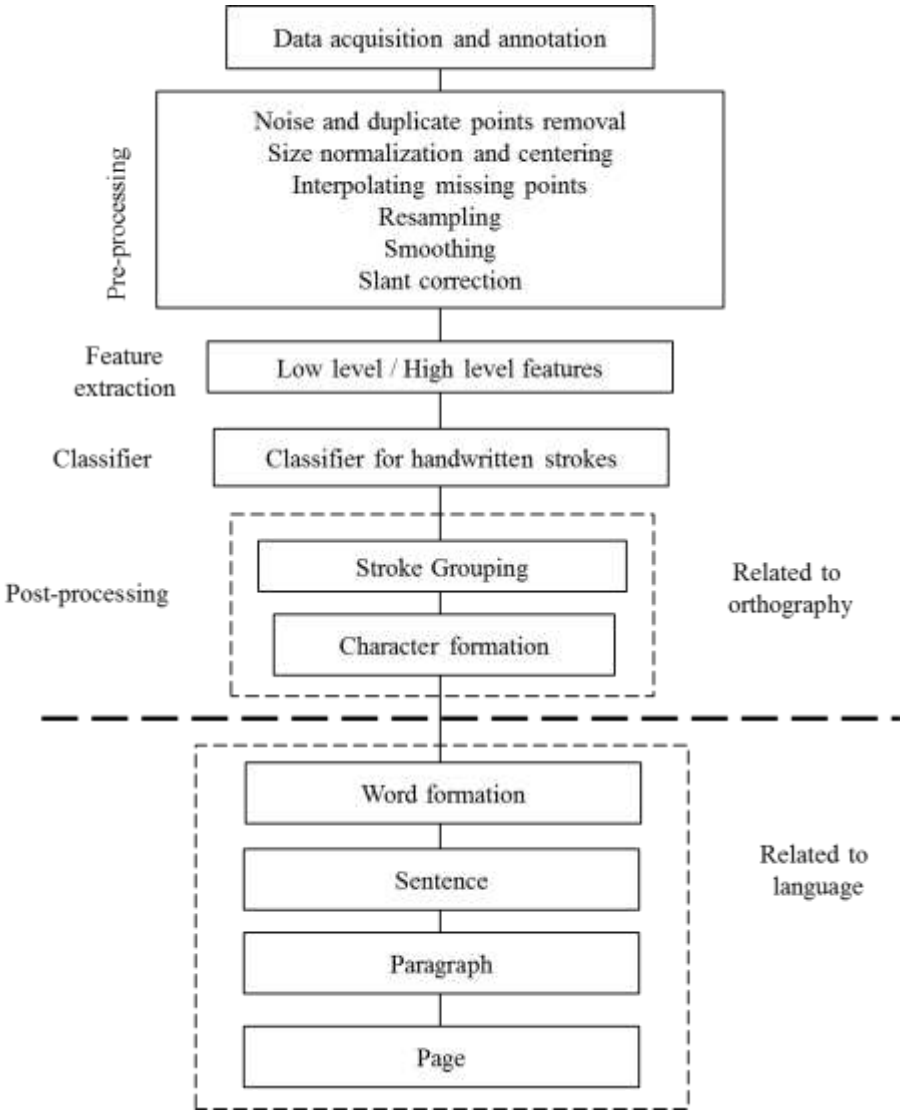


Figure 2.1: Architecture of handwriting recognition process

2.1 Data acquisition and annotation

First step in the direction of handwriting recognition process is to collect data from diverse users. This step requires a transducer that will capture the writing. Electronic tablet or digitizer is the most commonly used device that uses a digital pen (stylus) to collect handwriting data. When we use a digital pen to write data, two movements are recorded, *viz.* Pen-Down and Pen-Up. These pen traces are sampled at constant rate, therefore these pen traces are evenly distributed in time and not in space. When a user moves digital pen on the writing pad, the time sequential signal captures the dynamic positions of the pen, *i.e.*, sequence of consecutive points on *x-y* plane in the form of coordinates with the help of sensors attached to the pad. However, in recent time *DigiMemos* are also used to collect data for handwriting samples which do not require any special kind of pen. Examples of electronic tablet or digitizer include Personal Digital Assistant (PDA), cross pad (or pen tablet), tablet PC and *DigiMemo*.

For successful completion of this phase, an attempt should be made to capture a good amount of variation with respect to category of writers and handwriting styles of writers. Jaegar *et al.* (2003) and Nakagawa and Matsumoto (2004) have described the process of data collection for Japanese language where input data of handwritten Japanese characters are captured in boxes. As such, segmentation was not required in their work. Handwriting data collection for French language is discussed by Viard-Gaudin *et al.* (1999).

Bhaskarabhatla *et al.* (2004) and Agrawal *et al.* (2004) have discussed few issues in data collection. They have suggested that for data collection, it is desirable to categorize different styles of writing a character and obtain a minimal set of characters and words in the writing of a particular script. After this initial identification, data collection of handwritten samples should start.

Following data collection, it is necessary to accurately assign labels to the collected handwritten data. Decision on selection of the kind of input device we want to use for recognition process is mainly based on compatibility with operating system in use, active

area dimensions and report rate of pen movements. A good number of tools are available that assist in the collection and annotation of handwritten data (Bhaskarabhatla *et al.*, 2004; Vuurpijl *et al.*, 2004; Madhvanath *et al.*, 2006; Kumar *et al.*, 2006; Balasubramanian, 2006). The first PDA, Organizer was released in 1984 by Psion. Common personal digital assistants available in the market are Acer N Series, Apple Newton, Dell Axim, HP iPAQ, HP Jornada Pocket PC, iPAQ, Philips Nino, Sony Magic link with the Magic Cap operating system, Toshiba e310, Palm Taxi aka Palm-Pilot, Handsrping Visor, Tungsten E2, PalmPilot Personal and PalmPilot Professional.

An interesting area explored by researchers in this direction is automatic generation of handwritten data. Hollerbach (1981) discussed oscillation theory for measurement of human writing and for this purpose developed an acceleration and position measuring apparatus. Helmers and Bunke (2003) introduced three different methods for synthetic generation of handwritten text. They also discussed models for handwritten characters in five Indian languages for which they collected 12000 words from 5 writers. Other similar studies on developing models for handwriting data collection are discussed by Hollerbach (1981), Morasso and Ivaldi (1982), Plamondon and Maarse (1989), and Plamondon and Guerfali (1998).

2.2 Pre-processing

When input of data is done using a writing pen on digitizer tablet, few limitations like noise and distortion in input data makes the recognition process difficult. Pre-processing serves many purposes that make the recognition process smooth and easy. It is used for removing noisy artifacts from handwriting and also helps in correcting imperfections (Guerfali and Plamondon, 1993). Pre-processing stage intends to provide a uniform representation that aim at enhancement of improving the classification accuracy of recognition process. During online handwriting recognition process, pre-processing phase is helpful to remove noise or distortions present in input data due to hardware and software limitations (Govindaraju *et al.*, 1997; Subrahmonia and Zimmerman, 2000). Such limitations of noise or distortions include abnormal size of text, missing points due to fast writing with the pen, jitter present in text,

slant handwriting and uneven distance of points from adjoining positions. In addition to this, pre-processing is also useful to reduce the information content of the online handwritten signal (Guerfali and Plamondon, 1993).

Variation in handwriting style, way of holding the pen and writer's sentiments affect his writing style. These issues ultimately cause the problem of noise and make the recognition process complicated. Various forms that witness the presence of noise in the input text include sharp edges, irregular sizes of text, non-centered text and missing points in text trajectories (Beigi *et al.*, 1994a; Jaeger *et al.*, 2003). Pre-processing stage has been admitted to be an important phase of recognition process that improves overall recognition rate (Jaeger *et al.*, 2001; Slavik and Govindaraju, 2001; Huang *et al.*, 2007). Recognizing the importance of this phase many researchers have endeavored to describe its challenges from time to time for various scripts (Guerfali and Plamondon, 1993; Beigi *et al.*, 1994a; Beigi, 1996; Nagy, 2000, Wink and Roerdink, 2004; Malik and Khan, 2005; Javed and Hussain, 2009). In the forthcoming sub sections important steps of pre-processing have been discussed which include:

- Noise removal
- Size normalization and centering
- Interpolating missing points
- Smoothing
- Slant correction
- Resampling of points

2.2.1 Pre-processing for noise removal

The presence of noise is a common problem faced in handwriting recognition process. This problem surfaces due to hardware limitations, limitations of the digitizing process, crooked hand motion and imprecisions of pen-down indication. This section reviews the literature on noise removal steps including de-hooking, smoothing and wild point correction.

The problem of hooks results from skidding of pen in a particular direction. This problem is found at the start or end of handwritten character. This problem usually emerges with the habitual writing style where the writer has tendency to write character and end it up by skidding the pen. This problem is characterized by large angular variations and poses a major hurdle for feature extraction due to erratic hand motion in placing the stylus on or lifting it off the tablet accompanied by inaccuracies in pen-paper contact detection (Guerfali and Plamondon, 1993). De-hooking algorithms eliminate the problem of hooks identified at the beginning, but more frequently at the end of strokes (Mandler, 1987; Daifallah *et al.*, 2009). Smoothing of data is required to remove jitters in the handwritten character which is generally the result of roughness in the pen tip or surface used for writing handwritten characters (Kavallieratou *et al.*, 2002). Daifallah *et al.* (2009) performed smoothing with the proposed low pass filter algorithm to reduce noise for online Arabic handwriting recognition process. Finally, wild point correction removes an occasional spurious point which is generally caused by the hardware problem. Speed of hand motion is restricted by the forces of muscular contraction and the masses of hand and pen. Hence, high speed can detect wild points as spurious lines in the input pattern (Tappert *et al.*, 1990). Malaviya *et al.* (1993) have explained use of fuzzy features to identify wild points.

2.2.2 Size normalization and centering

The handwritten characters involve a great deal of variability. Normalization aims at the elimination of intra-class variability. In order to obtain uniformity in representation, process of normalization involves regularization of position, size and shape. Size normalization step adjusts the character size to the required standard (Brown and Ganpathy, 1983). Normalization is required as the size of handwritten character will vary from person to person and even with same person from time to time. Reason for this variation in the size of the input stroke is because output of handwritten character depends on the movement of the pen, the writing pad and also on the writing style. Size normalization normalizes every stroke to the same size and centered to a constant frame and places the text at a fixed distance from the origin. Centering is required when pen is moved along the border of the writing pad. The

use of size normalization techniques in online handwriting recognition has been discussed by Beigi *et al.* (1994a) and Daifallah *et al.* (2009). Kim (1983) explained the baseline orientation or baseline drift and used this to implement normalization. Bengio and Cun (1994), Brakensiek *et al.* (2002) and Pastor *et al.* (2004) discussed the approach of size normalization. Jhajj and Sharma (2010) also adopted size normalization where they used 48×48 size normalized image of *Gurmukhi* script characters and created 64 (8×8) zones. Yamada *et al.* (1990) discussed entirely different approach for normalization. They introduced non-linear normalization that aims to equalize the line density in order to exploit the area within the bounding box of the character.

2.2.3 Slant correction and resampling

Hollerbach (1981) discussed the issue of slanted handwriting, especially in the case of writers with cursive handwriting style. Researchers have focused on slant correction with an aim to reduce variations in a script and enhance the quality of segmentation (Madhvanath *et al.*, 1999; Yimei *et al.*, 2000; Slavik and Govindaraju, 2001). Cheriet *et al*. (2007) argued that in order to solve slanting problem, slant angle θ is to be measured and horizontal shear transform is to be applied on all the pixels so that handwritten character may be shifted to right or left. Slant correction is applied for size normalization of the input words that usually occur due to variation in handwriting style (Uchida *et al.*, 2005). Existing literature provides deskewing algorithms for slant correction (Brown and Ganpathy, 1983; Burr, 1983). Deskewing restricts the projection of non-diacritic characters on the *x*-axis to be spatially separable and involves the absolute segmentation of characters especially in case of cursive word (Tappert, 1982). Histogram of directions of pen movement while writing a character gives the basic estimation of slant of handwriting. However, the main techniques for slant estimation and correction are projection method, run length based technique, generalized chain code estimator and extreme method (Bozinovic and Srihari, 1989; Guillevic and Suen, 1994; Negi *et al.*, 1995; Simoncini and Kovacs, 1995).

Resampling of points involves the points in the list to be equidistant from neighboring points where data points are considered from the original points in the list. Equidistant resampling is

done to bring each stroke at equal intervals in space along with its trajectory. Aparna *et al.* (2004) described one dimensional linear interpolation technique that can be used as a procedure for uniform resampling. Existing literature provides detail on resampling techniques with emphasis on retaining information about corner points (Brault and Plamondon, 1993; Pavlidis *et al.*, 1997; Huang *et al.*, 2000; Kobayashi *et al.*, 2001).

2.2.4 Segmentation

Segmentation task can be performed before or after pre-processing. Segmentation can be classified into two categories, *viz.* external segmentation and internal segmentation. If segmentation is performed before recognition phase it is known as external segmentation (Maier, 1986) whereas segmentation when performed concurrent to the process of recognition, it is known as internal or implicit segmentation (Tappert, 1982; Caillault and Gaudin, 2006). External segmentation offers greater interactivity, reduces computation and provides simplicity in recognition process (Tappert *et al.*, 1990). Casey and Lecolinet (1996) reviewed the research on segmentation and suggested that segmentation methods can be classified into four categories, *viz.* classical, dissection, hybrid, and holistic. They have inferred that segmentation is dependent on local topological as well as global contextual information. In addition to this, many researchers have classified segmentation methods based on their findings and experiences. Dunn and Wang (1992) presented their research findings on segmentation and divided segmentation techniques as: straight segmentation and segmentation-recognition. Lu and Shridhar (1996) discussed segmentation for hand-printed words, handwritten numerals and cursive handwritten words. Casey and Lecolinet (1996) divided their survey on recognition into four categories as dissection technique, recognition-based segmentation, over segmentation and holistic segmentation.

Gaillit (1974) discussed this concept and provided that segmentation can be obtained as horizontal difference between the characters. Further, Fox and Tappert (1987) discussed the use of both spatial and temporal information for the purpose of character segmentation. Many researchers have argued that segmentation task has been more tough in offline handwriting recognition process compared to online handwriting recognition (Casey and Lecolinet, 1996;

Lu and Sridhar, 1996; Elnagar and Alhajj, 2003). Plamondon and Srihari (2000) conducted a survey on handwriting recognition systems where segmentation task has been discussed and compared for offline and online handwriting recognition systems. Generally, study on segmentation for offline handwriting recognition system is valuable to comprehend segmentation in online handwriting recognition system because a common task of word level segmentation exists in both offline and online handwriting recognition systems. Ha *et al.* (1995) studied mathematical expressions and proposed the use of *x-y* cuts as bounding boxes to separate the symbols. These *x-y* cuts proposed by them were used as horizontal projection cuts and vertical projection cuts, as illustrated in Figure 2.2.

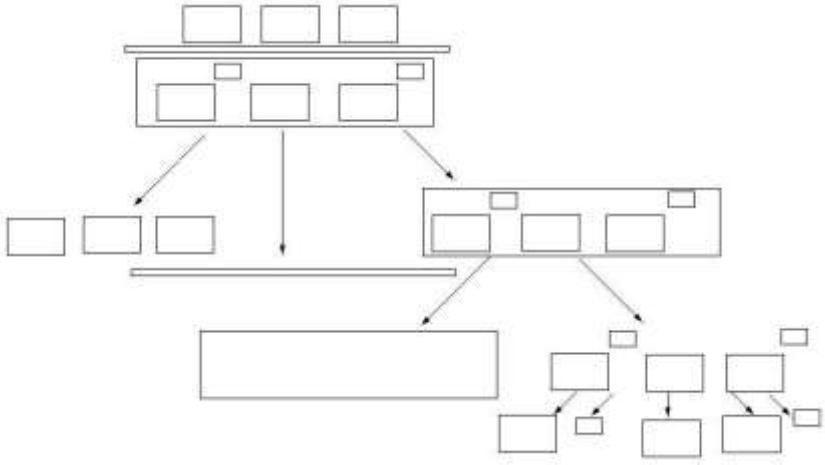


Figure 2.2: Projection cut for segmentation [Ha *et al*., 2005]

During 1990s researchers conducted many studies on Integrated Segmentation and Recognition (ISR) based approaches which used neural classifiers to decide on the proper segmentations of numeral strings (Keeler *et al.*, 1991; Matan *et al.*, 1992; Martin *et al.*, 1993). An important method of segmentation was proposed by Keeler *et al.* (1991). They named it as integrated segmentation and recognition method, which utilizes neural network architecture. It is also called as Self Organizing Integrated Segmentation and Recognition (SOISR) method. Matan *et al.* (1992) provided a segmentation technique based on a Space Displacement Neural Network (SDNN). They proposed a new method that removes the problems of excessive or repetitive use of the recognizer across the input data as was

common to traditional techniques of ISR. Yanikoglu *et al.* (1998) provided a novel algorithm for segmentation that involved the global characteristics of handwriting. Under this technique they obtained sequential segmentation points by assessing a cost function at each point along the baseline. Blumenstein and Verma (1999) proposed a new algorithm for segmentation that initially over-segments handwritten input data by using heuristics and feature detection. Nicchiotti and Scagliola (2000) discussed a method of segmentation for cursive handwriting which assumes the base of handwritten profiles and extraction of white holes for the purpose of segmentation.

Lehal and Singh (2001) discussed text segmentation of machine printed *Gurmukhi* script and provided solutions for the difficulties related to connectivity of the characters on the headline, crossing between two or more characters in a word, multi-component characters and touching characters. Elnagar and Alhajj (2003) discussed a new approach of segmentation based on separating single touching handwritten digit strings. Important segmentation points are estimated based on decision line that is assumed from the deepest or highest valley in the image. The partitioning track is determined exactly and then the numerals are separated before restoration is applied. Shin (2004) provided a structural analysis algorithm that searches globally for important points of segmentation on a character unit level. Most important benefit offered by his approach is that it can cope with large variations in stroke shape and position. The search for key segmentation points is performed in a very systematic and logical manner with the help of two-level dynamic programming matching algorithm in conjunction with syntax control of composition characteristics. Jindal *et al.* (2007) proposed some new algorithms for segmentation of horizontally overlapping lines and applied this technique on Indian scripts. For this purpose they divided the whole document into strips and then proposed an algorithm for segmentation of horizontally overlapping lines and associating small strips to their respective lines. Their algorithms achieved 96.5% to 99.8% accuracy for different scripts.

2.3 Computation of features

Identification of appropriate features is an important step in handwriting recognition process (Suen *et al.*, 1980; Impedovo *et al.*, 1991; Gader and Khabou, 1996; Jinhai and Liu, 1999). Devijaver (1982) argued that for successful recognition of characters, the features extracted should be beneficial for minimizing within-class pattern variability and at the same time maximizing between-class pattern variability. Research on feature extraction for different scripts is being carried out since more than three decades. However, special emphasis has been put on this phase during last decade. Joshi *et al.* (2005) proposed and discussed effectiveness of structural features for online handwritten Devanagari characters recognition process. Al-Taani (2005) introduced feature extraction steps in his study on recognition of online handwritten digits in Arabic language. Vamvakas *et al.* (2007) discussed statistical and structural features for Greek handwritten character recognition. They used zoning, projections and crossings among statistical features whereas end point, loop, radial histogram and crossing points have been discussed under structural features. Many pieces of research work on feature extraction have concluded that high accuracy results of recognition can be achieved with appropriate features selected (Tier *et al.*, 1996). Sharma *et al.* (2009) introduced recognition algorithms for four high-level features of *Gurmukhi* strokes that included loop, headline, straight line and loop. They conducted experiments across 60 writers and 5%, 3.3%, 6.7% and 8.3% improvements have been observed for recognition of loop, headline, straight line and dot features, respectively, after using preprocessing algorithms.

Researchers have proposed wide range of features for character recognition process that include geometrical features, statistical features, topological features, and global features. (Okamoto, 1999; Jose *et al.*, 2002; Dongre *et al.*, 2010). Broadly, features are classified as low level and high level features. Low level features focus on estimation of neighboring points whereas high level features focus on large scale and provide information regarding loop, headline, dots, *etc*. Hu *et al.* (1997) proposed combined use of both low level features and high level features at each point that ensured covering large input pattern and possessed invariance property. Verma *et al.* (2004) provided a new technique for feature extraction of

online handwriting recognition from raw data based on structural, directional and zoning information that has been used to create a single feature vector. Siddharth *et al*. (2011) considered three types of statistical features for recognition of *Gurmukhi* handwritten characters: zonal density, projection histogram and distance profile. They achieved an accuracy of 99.2% for numeral recognition using projection histogram and an accuracy of 99.1% while using zonal density features. Along with the combination of these three categories of features, they also used Background Directional Distribution (BDD) features. In case of BDD, they computed eight directional features for each of foreground pixel.

Although different set of features are identified for different types of script, yet Rocha and Pavlidis (1994); Lee (1996); Kim and Govindaraju (1997); Hu *et al.* (2000) suggested that no unique set of features can be applied in generic form on all kinds of scripts. Jaeger *et al.* (2001) also supported this phenomenon in his research and concluded that there is no standard method for computing features of a script. Lehal and Singh (2000) identified features for offline *Gurmukhi* script and classified them into two sets of structural features as primary features and secondary features. Features identified and used for online handwritten characters are called spatiotemporal features because the sequence of feature here reveals the temporal variation of the positional information which helps in locating the attributes that convey positional, directional, structural or statistical features derived for the handwritten character. Both local and global features can be used for online handwriting data (Bahlmann *et al.*, 2002). Local features are used to capture important information about part of the stroke that reveals the spatial or temporal proximity of subsequent points around the data point whereas global features combine long range pattern information. Few features that are included in the list of global features are Fourier coefficients (Man and Poon, 1992), wavelet transform (Sundaresan, 1999), linear prediction coefficients (Alahari *et al.*, 2005), and graph based representations (Liu *et al*., 1996).

In feature extraction, problem of dimensionality has been noticed by researchers in the situation when we use all points of a character. To resolve this issue feature extraction based on statistical distribution of point was proposed (Impedovo *et al.*, 1991). Existing literature

provides discussion on five methods based on distribution of points: *a)* Moments, *b) n*-tuples, *c)* Zoning, *d)* Characteristic loci, and *e)* Crossings and distances (Hussain *et al.*, 1972; Suen *et al.*, 1980). However, recently fuzzy features have also been proposed in some research studies on online handwriting recognition process (Malaviya *et.al.*, 1996). Lajish and Kopparapu (2010) also used fuzzy features for recognition of online handwritten Devanagari characters. In their research study, directional features were obtained with the help of angle between two adjacent points and an accuracy of 96.9% has been achieved by them.

2.4 Recognition

Intense ongoing research during last four decades proves to be the testimonial of popularity gained by handwriting recognition process. Many developments have already taken place in online handwriting recognition process. Researchers are still engaged in designing and developing new and accurate dissection systems that may be used to advance succeeding word classification (Xiao and Leedham, 2000). Recognition phase involves sequence of encoded values corresponding to the handwritten characters in the input text. Broadly, recognition paradigms are classified and differentiated as holistic and analytic. Among these, holistic approach considers handwritten word as indivisible unit. This approach is used for both online and offline handwritten recognition but this system is considered better for cursive writing systems where words written are connected. Govindaraju *et al.* (1997) and Koerich *et al.* (2005) proposed and discussed approaches for classification of handwritten words. In case of analytic methods, classification is performed for sub words that will involve graphemes (Lecolinet and Crettez, 1991), character (Favata and Srihari, 1992) and stroke (Swethalakshmi *et al.*, 2006).

In addition to this, generative and discriminative approaches are also used for the purpose of classification. Generative approach of classification provides developing of a model for a particular class by considering common features of that class. The best example of generative classifier is Hidden Markov Model (HMM). Discriminative approach does not restrict its concentration on designing a model for a class using examples belonging to that class but also takes in its purview those belonging to other classes. However, few studies have

suggested using combination of both the approaches (Sanguansat *et al.*, 2004 and Bahlmann *et at.*, 2002). Suen *et al.* (1992) also supported it and highlighted that each classifier has its own strengths and weaknesses so no single scheme will serve the purpose of achieving high accuracy in recognition of handwritten numerals or characters. In this light, authors emphasized that a new direction that can offer promising results of accuracy in handwriting recognition is the combination of some classifiers into a multiple expert system.

Recognizing the importance of different classifiers, many researchers have proposed multi-stage classification methods and multi expert combination methods in their study. Duerr *et al.* (1980) made use of both statistical methods and structural methods for classification of handwritten characters. Gader *et al.* (1991) proposed multi-stage system that used Yule template matcher for identification of easy digits and model based classifier was used for more complex digits. Hull *et al.* (1990) proposed a combination of template matching, structural analysis and syntactic classifiers for recognition of handwritten alphanumeric characters. For this purpose, they used the output from each classifier and a common consensus was arrived considering the output of each classifier. Forthcoming sub sections provide detail of existing literature available on different recognition methods.

2.4.1 Statistical methods

In statistical methods, each pattern is represented in terms of a particular measurement unit *d* and is observed as a point in that particular *d* dimensional space. The main purpose is to select those features where all pattern vectors belonging to different classes occupy disjoint area in a *d* dimensional space (Jain *et al.*, 2000). Separation of patterns from different categories will help in determining the success rate of representation space. The best use of statistical methods that has made it popular is that this technique takes into consideration prior probabilities of classes and easily adopts the natural variations in handwriting as stochastic in nature. For this purpose a training set is provided for each class and emphasis is made to decide boundaries based on the probability distribution in the feature space so as to produce demarcation between the patterns belonging to different set of classes. In statistical methods, for the purpose of character recognition we first select the most probable character

class that poses minimum expected classification error. Thus, this process includes the estimation of the discriminant function specific to the statistical approach (Duda *et al.*, 2001). Among earlier studies, Beigi *et al.* (1994b) used statistical methods for recognition of handwritten Persian and Arabian digits where few difficulties associated with this method were identified. They collected 600 samples of each digit from 20 users and system was trained on this input data. The system was tested by 14 writers who wrote each digit five times and achieved an accuracy of 93.1%.

Statistical methods are classified into two broad categories: parametric methods and non-parametric methods. In parametric methods, handwriting samples are statistical variables from distribution that is characterized by a set of parameters and each class includes its own distribution parameters. Fu *et al.* (2000) provided that if the estimated parameters are continuously updated the system itself adjusts to different handwriting styles. Hidden Markov Model (HMM) is the most widely accepted method in the category of parametric methods (Ait-Mohand *et al.*, 2014). Non-parametric methods are directly valued from training data like nearest neighbor is one of the famous non-parametric methods. In general, parametric methods are preferred over non-parametric methods because parametric methods are considered to be computationally efficient than non-parametric methods. HMM has also been used for online handwriting recognition systems (Sabourin *et al.*, 1999; Britto *et al.*, 2003). Earlier HMM was applied for speech recognition tasks only. It was adopted for online handwriting recognition systems in 1990s. A study by Plamondon and Srihari (2000) provided use of HMM for cursive handwriting data. However, recently during last decade supervised learning model Support Vector Machine (SVM) has popularly been used by world-wide researchers as pattern classifier which is based on statistical learning technique (Marwala *et al.* 2007; Kim *et al.*, 2001; Byan *et al.*, 2003). Sanguansat *et al.* (2004) proposed a combination of HMM and SVM for recognition of Thai handwritten characters. Besides this, nearest neighbor approach (Jaeger *et al.*, 2003), clustering approach (Vuori and Laaksonen, 2002), Bayesian classifiers (Cho and Kim, 2001) and Modified Quadratic Discriminant Function (MQDF) (Liu *et al.*, 2004) are few other statistical approaches that have been used for handwriting recognition. Sharma *et al.* (2010) presented HMM based

online handwritten character recognition for *Gurmukhi* script. For developing this system, they constructed a database of 5,330 *Gurmukhi* characters and achieved 91.9% recognition accuracy.

2.4.2 Structural and syntactical methods

Structural and syntactical methods are related to handwritten patterns where patterns identified are complex in nature. To make this process convenient, structures and grammar base is considered and hierarchical perspective is adopted where each pattern is composed of few simple sub patterns which are further made up of simpler sub patterns (Fu, 1982). Structural recognition describes the process of how a given pattern is constructed from the primitives. Recognition of these primitives enables representation of complex patterns based on the interrelationship between the primitives. This paradigm is mainly adopted in positions where the patterns have a fixed structure that can be studied in terms of a set of rules such as textured images, waveforms and shape analysis of contours. Duneau and Dorizzi (1994) presented a system for recognition of English cursive words input on a digitizing tablet. This system uses an analytical approach that tries to localize the letters of the word to be recognized due to a set of letter prototypes. Chan and Yueng (1999) also discussed structural approach for online handwritten alphanumeric characters that achieved reasonable speed and fairly high accuracy. Structural recognition approach has also been used for online handwritten Korean characters (Jung and Kim, 2000). Kim and Kim (2001) proposed structural methods for online handwritten Hangul characters in which interrelationship between the strokes has been explained with the use of hierarchical characteristics of target characters. Al-Taani (2005) provided use of structural approach for online Arabian handwritten digits. His system has been tested on an online dataset containing the digits 0-9 collected from 100 users and primitives were identified. This system achieved an average accuracy of 95.0%. The chain codes are most popular structural representations used for recognition of online handwriting. In case of chain code, a stroke is temporarily divided into segments and finally this coding is done on these segments. These segments are small

straight lines of equal lengths and consider information as directions, angles, and geometric information in segments.

However, in case of syntactic pattern recognition, a formal analogy is drawn between the structure of patterns and the syntax of a language. The patterns are viewed as sentences belonging to a language, alphabet of the language are considered as primitives and based on the grammar sentences are generated. Jain *et al.* (2000) provided that a huge collection of complex patterns can be explained by small number of primitives and defined grammatical rules where these grammar rules are inferred from the available training samples. Sinha and Mahabala (1979) proposed a syntactic analysis system for recognition of handwritten Devanagari characters where the system stored the structural description of each symbol in terms of primitives and their interrelationship. The implementation of syntactic methods is considered to be more difficult as it has to deal with segmentation of noisy patterns and deriving grammar rules from the training data. Perlovsky (1998) also explained that syntactic methods become difficult to adopt as it offers combinatorial explosion of possibilities that need to be studied thoroughly. It will require large training sets and computational efforts. Suen *et al.* (1992) provided that many researchers adopted syntactic classifiers when features are identified based on character shape. For using syntactic approach, particular structures are parsed into descriptive grammar and further its comparison is made with regular expressions. Tai *et al.* (1993) discussed syntactic approach for recognition of online handwritten character of Chinese language where they described a multilayer representation of attributed semantic network. Many pieces of research have also introduced combination of these approaches to get better results (Duerr *et al*., 1980).

Ahmed and Suen (1987) proposed a two-stage classifier to recognize unconstrained handwritten zip codes. They used statistical approach at the front end and structural method for computing fuzzy membership between more complex patterns.

2.5 Post-processing

System results might be conflicting and possess some errors due to the problems in classification and segmentation. To remove these problems and improve accuracy of results, post-processing techniques are applied for recognition of online handwritten characters. The most common way to remove these problems is to apply linguistic knowledge. Statistical methods and tracing through dictionary are two commonly used methods for error correction in the recognition process. Among earlier studies, Sinha (1987) provided that generally, statistical approach is preferred over dictionary based methods because they perform better in terms of computational time and memory utilization. However, for minor mistakes dictionary approach is useful. Pitrelli and Perrone (2002) evaluated variety of functions to estimate posterior probabilities of correctness in a post-processing mode. The study provided that using post-processor along with Receiver Operating Characteristics (ROC) curves, the system was able to correctly reject 90.0% of recognizer errors and falsely reject 33.0% of correctly-recognized words. In isolated-digit recognition, they achieved a correct rejection rate of 90.0% and false rejection rate of 13.0% with a dataset of 45,800 character tokens written by 100 writers.

For handwriting input, some shape recognizers yield a single string of characters, while others yield a number of alternatives for each character, often with a measure of confidence for each alternative. A post-processor can operate on this information to obtain estimates for larger linguistic units, such as words. When the shape recognizer yields a single choice for each character, string correction algorithms are applicable (Tappert *et al.*, 1990). Alternate choices provide more information for post-processing. In post-processing, a dictionary can be used to restrict the character combinations. This can be implemented as a grammar that specifies all possible combinations of characters. Chan and Yueng (1999) proposed using automatically discovered word classes in Chinese class-gram models for contextual post-processing of handwriting recognition results. Three other language models have also been constructed by them for comparison. Pitrelli and Perrone (2002) evaluated a variety of scoring functions, including likelihood ratios and estimated posterior probabilities of

correctness, in a post-processing mode to generate confidence scores at the character or word level. Carbonnel and Anquetil (2004) presented an optimized lexical post-processor designed for handwritten word recognition. They corrected recognition and segmentation errors using lexical information from a lexicon. They presented lexical post-processing based on two phases. In the first phase a lexicon organization is made to reduce the search space into sub-lexicons during the recognition process. The second phase develops a specific edit distance to identify the handwritten word using a selection of the sub-lexicons. Namboodiri *et al.* (2007) presented a framework for incorporating the metric information in classical Indian poetry to enhance the results of recognition. Their algorithm can be used in conjunction with other post-processing approaches and is effective in correcting modifier symbols.

2.6 Earlier recognition results for online handwritten character recognition of Indian scripts

India having diversified cultures possesses numerous scripts. Each script has its own set of icons which are known as characters/ letters. For all different languages, numerous rules are there for combining letters that make a message understandable. One peculiar feature common in all major Indian scripts is that they comprise mixture of syllabic and alphabetic scripts. Although recent technology has contributed largely in the area of handwriting recognition of a particular language yet some more fine tuning is required for Indian regional languages. Limited literature is available for recognition results achieved for Indian regional languages in comparison to some world-wide popular languages like, Chinese and English. In this section we have compiled research work done on Indian scripts. Some widely acceptable recognition techniques adopted for Indian regional languages include HMM (Babu *et al.*, 2007), SVM (Swethalakshmi *et al.*, 2006) and Elastic matching (Prasanth *et al.*, 2007).

Devanagari is among oldest Indian script used to write Hindi, Marathi, Bihari and Sanskrit. Hindi is the national language of India and used by more than 300 million people for documentation. A good number of works are available for recognition of online handwritten Devanagari script. Malaviya *et al.* (1996) proposed fuzzy logic based approach for online

handwritten Devanagari script. Connell *et al.* (2000) conducted experiment with 20 different writers, with each writer writing 5 sample words of unconstrained Devanagari script. They achieved an accuracy of 86.5% with the combination of multiple classifiers. Namboodiri (2004) also worked on recognition of Devanagari script by using database of 13,379 words. They identified 11 spatial and temporal features from the tested strokes and obtained 87.1% overall accuracy. A study by Joshi *et al.* (2005) focused on developing a recognition engine for online handwritten Devanagari script by testing samples of 1,487 Devanagari characters written by 9 writers. They bifurcated these samples into two classes, namely, FREQ and RAND whereby in the first category they collected 100 most common characters and 250 characters were drawn randomly from the complete set and obtained 93.1% and 85.5% recognition accuracy, respectively. Bharath *et al.* (2005) have presented an approach whereby Devanagari characters can be recognized by the order and number of strokes used to write it. Kubatur *et al.* (2012) used neural networks to recognize online handwritten Devanagari characters and provided that a simple feature set yielded by Discrete Cosine Transform (DCT) can be highly promising. They conducted testing on 2,760 Devanagari characters and achieved 97.2% recognition accuracy.

Bangla script is used for writing Bengali and Assamees languages and used by more than 200 million people. Existing studies that support research on recognition of online handwritten Bangla characters include study by Garain and Chaudhuri (2002) and Bhattacharya *et al.* (2007). Parui *et al.* (2008) have discussed a novel method for recognition of online handwritten Bangla characters by collecting sample of 24,500 Bangla characters written by 70 individuals. They constructed an HMM with the help of training set of each stroke and achieved an accuracy of 84.6%. Biswas *et al.* (2012) designed HMM based classifier using each stroke class by collecting data set of 29,951 character samples of Bangla script. They obtained 91.9% character level accuracy with the proposed methodology. Bhattacharya *et al.* (2008) proposed analytic recognition approach for Bangla characters by using modified quadratic discriminant function. They collected sample of 10,000 online handwritten Bangla words written by 50 writers and obtained overall word level recognition of 82.3%.

In addition to these studies, few pieces of research work are also available for Tamil, Kannada, Malayalam, *Gurmukhi* and other Indian languages. Shankar *et al.* (2003) used string matching technique and proposed an independent Malayalam character recognition system for stroke recognition. They collected samples of 216 strokes from 3 writers and reported an accuracy of around 90.0%. Among *Gurmukhi* script much of the existing literature available provides recognition engine for offline handwritten *Gurmukhi* characters (Lehal and Singh, 2000; Jindal *et al.*, 2005). However, Sharma *et al.* (2008) proposed recognition engine for online handwritten *Gurmukhi* script using elastic matching approach and obtained a recognition accuracy of 90.1% with system trained on 40 classes. Prasad *et al.* (2012) used two methods, *viz.* Principal Component Analysis (PCA) and Dynamic Time Warping (DTW) for recognition of online handwritten Kannada script. They concluded that results obtained using PCA are highly promising with an accuracy of 88.0%.

2.7 Chapter summary

This chapter has compiled the studies conducted in the area of recognition of handwritten characters. Review of research studies has been conducted for recognition of handwritten characters of various scripts based on their novelty. Hence, a step wise literature review has been presented here, whereby the related existing studies have been accumulated under a particular phase as per their relative contribution and approach adopted. Data capturing and annotation is the starting step in designing a handwriting recognition system. Various techniques involved in the preprocessing of raw data have been presented. Different strategies adopted by researchers for recognition of online handwritten characters have been discussed in this chapter. Studies on post processing that aim to increase the recognition accuracy have also been studied. Under last section of this chapter, we have focused on methodology adopted by existing studies on Indian regional languages and their achievement in terms of recognition accuracy. In the next chapter, we have given a brief overview about system design and procedure used in the proposed study.

CHAPTER 3

METHODOLOGY AND SYSTEM DESIGN

Existing framework for handwriting recognition clearly defines active areas for the development of such a system. From existing literature, it is evident that research in this area has been alive for more than four decades. Nevertheless, it is still admitted to be a challenging area which is encouraging researchers world-wide to refine existing methods. Research in this area demands developing new techniques for segmentation, identifying new features, and developing post-processors that can achieve higher recognition accuracy. This study presents new sub-systems for pre-processing and post-processing in online handwritten *Gurmukhi* script recognition. This chapter describes the methodology and system design as adopted in the current research work. The chapter has been divided into five sections. Section 3.1 explains the novelty of proposed framework. Section 3.2 describes the collection process of *Gurmukhi* words written by different users. The words collected here have been used for testing the results. Section 3.3 introduces SVM classifier that has been used for recognition of handwritten *Gurmukhi* script strokes in this research work. Section 3.4 discusses the testing strategies adopted for recognition of characters, numerals, aksharas and words of *Gurmukhi* script. Last section summarizes the chapter by presenting an overview of the framework and techniques followed for recognizing online handwritten *Gurmukhi* characters.

3.1 Proposed framework for the recognition of handwritten *Gurmukhi* script

Handwriting recognition systems use standardized sequential steps to give output. Various steps involved in handwriting recognition process include data collection, pre-processing, feature extraction, segmentation, recognition, and post-processing (Impedovo *et al.*, 1991; Chan and Yeung, 1999; Koerich *et al.*, 2003; Pardeep *et al.*, 2011).

In a traditional handwriting system, data collection is an important step. Pre-processing step provides a uniform representation that aims to improve the classification accuracy. Further, identification of appropriate features leads to help in the recognition process.

This section describes, in detail, the novelty of proposed framework used for *Gurmukhi* script recognition.

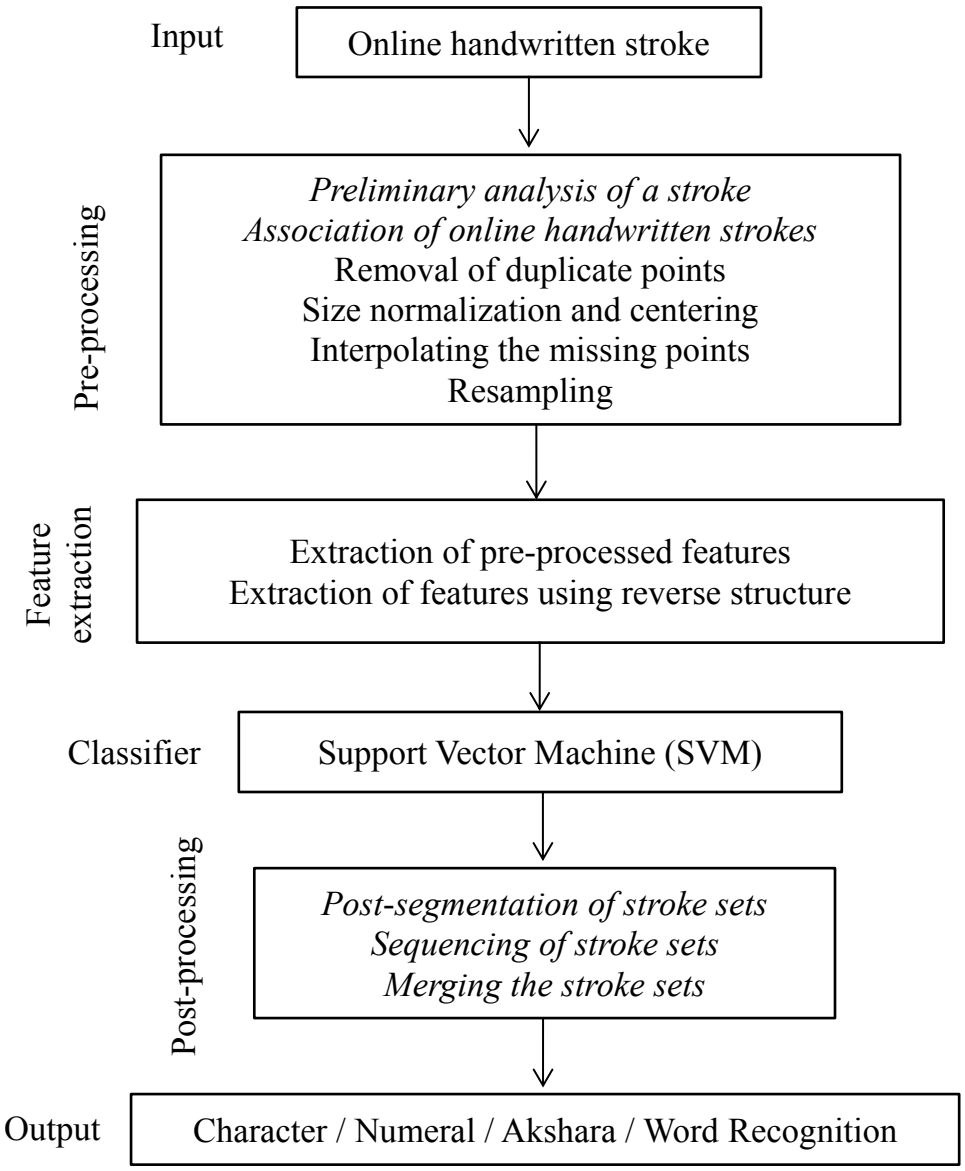


Figure 3.1: Proposed framework for online handwritten *Gurmukhi* script recognition system

Some of the steps of this framework are similar to the sequential steps used in traditional architecture of handwriting recognition process. However, in the proposed framework, some novel steps have been introduced. Figure 3.1 displays a diagrammatic representation of the proposed framework for online handwritten *Gurmukhi* script recognition. In this framework, the recognition process starts with the collection of raw data in the form of handwritten samples of *Gurmukhi* script from users of diverse

background in writing *Gurmukhi* script. Pre-processing step includes a novel way of associating online handwritten strokes, whereby association of each stroke is checked with other strokes. In the existing methods, if one stroke is overlapped with adjacent stroke, then the segmentation process is required. However, in the proposed work, a new methodology has been developed for finding association between the strokes which is divided into three steps, *viz.* schema for stroke association, sets of associated strokes, and reduction of associated sets. Further, pre-processing also includes removal of duplicate points, size normalization, centering, interpolating the missing points and resampling. The features such as positional features, pre-processed features, and features using reverse structure have been used in this work. An important contribution of this work is the methodology for implementing post-processing. Post-segmentation of stroke sets, sequencing of stroke sets, and merging the stroke sets are included under post-processing which contributes towards achieving high accuracy.

3.2 Data collection and annotation

Data collection is one of the most important tasks to be performed in online handwritten character recognition. The preliminary examination in this area made it clear that no public online handwritten *Gurmukhi* character database is available to carry out recognition experiments. Hence, creating a reliable and authentic database is also a part of the current research work. The first objective of this study was to create an annotated set of 30,000 *Gurmukhi* words (including 2000 unique words) to be used for training and testing of OHWR engine. Raw data was obtained with the help of digital pen (stylus) and the tablet PC (Dell Latitude XT3). The time sequential signal from the movement of the pen on the writing device captured the dynamic position of the pen with the help of sensors attached to the device. Thus, it was able to understand the pen pressure, pen down/up events and sequence of consecutive points on the *x*-*y* plane in the form of coordinates. In this process, following attributes of a handwritten character were recorded:

(i) Total number of strokes in a character,

(ii) Digitized (x, y) coordinates of each stroke, and

(iii) Index of each stroke.

Figure 3.2 depicts screen shots of data collection tool used for collecting handwritten samples of *Gurmukhi* script words. A dictionary of 2,200 different *Gurmukhi* words has been used for data collection in this work.

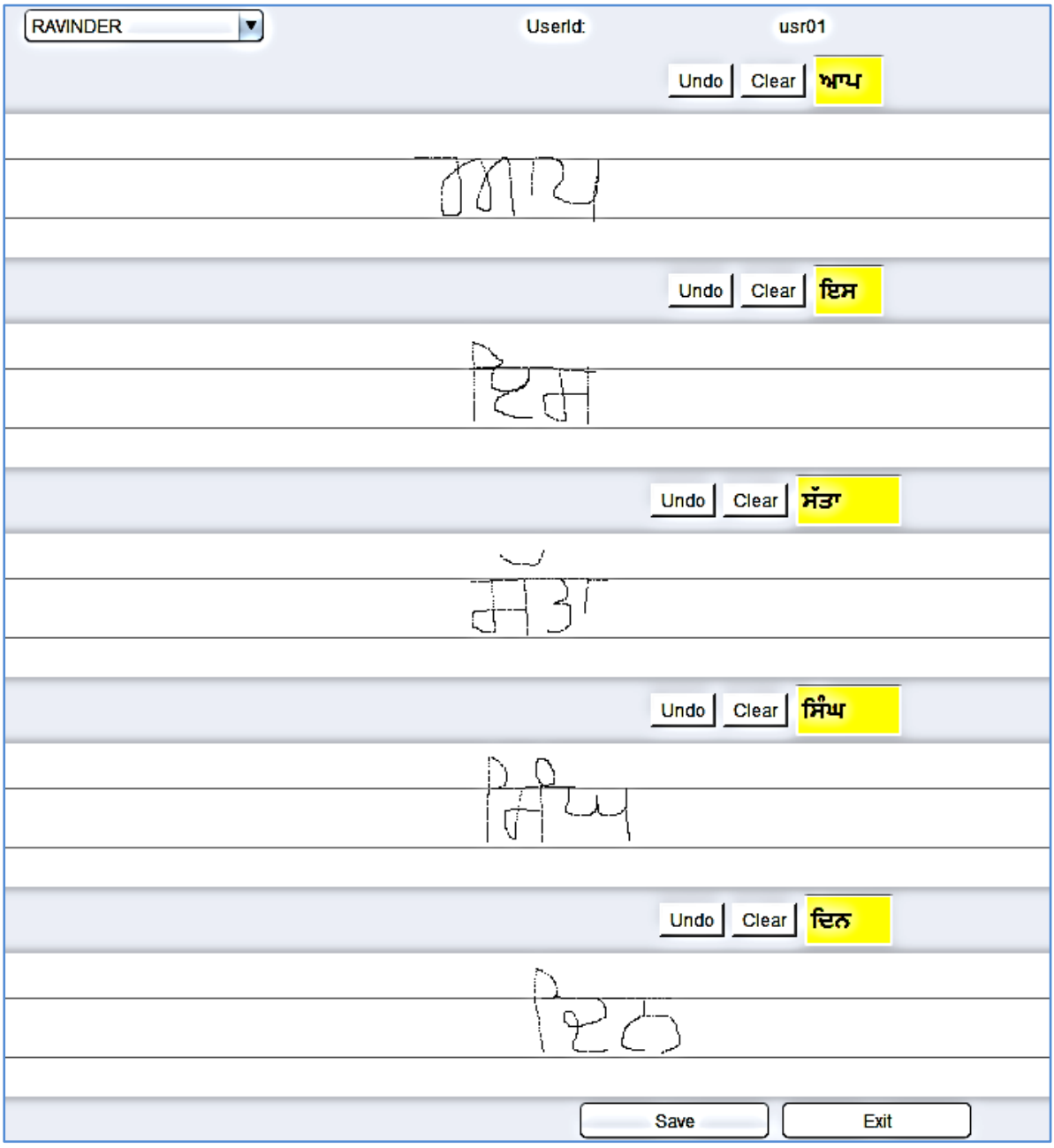


Figure 3.2: Screen shots of data collection tool

This data dictionary has been stored in .XML (extensible markup language) format and the pseudo of this file is presented in Table 3.1(a). Table 3.1(b) contains the pseudo of the file that has been used to store user's information.

Table 3.1: Extensible markup language (.XML) template of (a) The first five records of data dictionary of *Gurmukhi* words (b) Writer information

(a)

```
<?xml version="1.0" encoding="UTF-8"?>
▼<OHWRSchema>
 ▼<dataSetDef>
  <SrNo>101</SrNo>
  <WordNo>101</WordNo>
  <WordDesc>ਆਪ</WordDesc>
 </dataSetDef>
 ▼<dataSetDef>
  <SrNo>102</SrNo>
  <WordNo>102</WordNo>
  <WordDesc>ਇਸ</WordDesc>
  </dataSetDef>
 ▼<dataSetDef>
  <SrNo>103</SrNo>
  <WordNo>103</WordNo>
  <WordDesc>ਸੱਤਾ</WordDesc>
  </dataSetDef>
 ▼<dataSetDef>
  <SrNo>104</SrNo>
  <WordNo>104</WordNo>
  <WordDesc>ਸਿੰਘ</WordDesc>
 </dataSetDef>
 ▼<dataSetDef>
  <SrNo>105</SrNo>
  <WordNo>105</WordNo>
  <WordDesc>ਦਿਨ</WordDesc>
 </dataSetDef>
</OHWRSchema>
```

(b)

```
<?xml version="1.0" encoding="UTF-8"?>
▼<OHWRSchema>
 ▼<dataSetDef>
  <templateID>GurmukhiVer1</templateID>
  <language>Punjabi</language>
  <templateSource>IISc</templateSource>
 <contactName>SMCA CC Lab</contactName>
 <contactEmail>ravinder@thapar.edu </contactEmail>
 <contentDesc>Tablet PC & Desktop</contentDesc>
 </dataSetDef>
 ▼<writeDef>
  <school_id>TU_PTA</school_id>
  <name>RAVINDER KUMAR</name>
  <DOB>22/08/1987</DOB>
  <uName>usr01</uName>
  <Password>test</Password>
  <age>34</age>
  <gender>Male</gender>
  <region>PATIALA</region>
  <educationLevel>PG</educationLevel>
  <profession>Student/Professor - B</profession>
  <hand>Right</hand>
  <deviceType>Tablet PC</deviceType>
  <index>10</index>
  <wordCount>100</wordCount>
 </writeDef>
</OHWRSchema>
```

Samples of online handwritten *Gurmukhi* words have been collected from 144 users residing in Punjab. The details of these users are given in Table 3.2. These users were categorized on the basis of their level of proficiency in writing *Gurmukhi* script and were put in four categories, namely, Category A, Category B, Category C, and Category D, as described in Table 3.2.

Table 3.2: Categories of *Gurmukhi* script writers

Category	Category Name (Number of Persons)	Educational Background
Category A	Highly Proficient (54)	Punjabi users such as Govt. employees / Language experts. These users follow prescribed way of writing.
Category B	Moderately Proficient (38)	Users who have acquired formal Punjabi education, up to high school; but they do not write Punjabi frequently.
Category C	Low Proficient (34)	Users who have acquired formal Punjabi education, up to primary; but they write Punjabi frequently.
Category D	Ignorant (18)	Users who have no formal education in Punjabi; but can write Punjabi on the basis of structure.

The handwritten words collected from different writers included combinations of 35 characters, 9 vowel modifiers, 3 nasals, and 10 numerals of *Gurmukhi* script. A total number of 39,871 sample words of *Gurmukhi* script were collected from different category of users. Table 3.3 provides a statistics on collected raw words, raw strokes, annotated strokes, and noise in collected words. From Category A writers, 24,090 handwritten *Gurmukhi* words have been collected and then passed through annotation process. From these words, 1,38,994 strokes have been identified and 1,38,296 strokes were annotated. From Category B writers, 9,460 handwritten *Gurmukhi* words have been collected. Out of these collected words, 53,676 strokes have been identified and 53,329 strokes were annotated. Further, 4,203 *Gurmukhi* words have been collected from category C users. Out of these collected words, 21,802 strokes have been identified and 21,218 strokes were annotated. From Category D writers, 2,118 words have been collected. Out of these words, 13,970 strokes have been identified and 13,487 strokes were annotated.

Table 3.3: Statistics on collected data

Name of Category (Number of Users)	Collected Raw Words	Collected Raw Strokes	Annotated Strokes	Noise in Data Collection (%)
Category A (54)	24090	138994	138296	0.50
Category B (38)	9460	53676	53329	0.65
Category C (34)	4203	21802	21218	2.68
Category D (18)	2118	13970	13487	3.46
Total 144	**39871**	**228442**	**226330**	**0.92**

Figure 3.3 depicts the procedure adopted for annotation of collected handwritten *Gurmukhi* words. Under this process, the input word as written by a user has been recorded along with its strokes by stroke sequence. Out of this, the strokes included in the defined set for formation of *Gurmukhi* character are allotted a particular class ID. Rest of the unidentified strokes are deleted and considered as noise. All those strokes which are identified and annotated are further used for training of LibSVM classifier. Figure 3.3 demonstrates the procedure of annotation for the *Gurmukhi* word /ਉੜੇ/ where, user has written this word with sequence of four strokes /ਉ/, /ੁ/, /ੜ/ and /ੇ/. Here, all the identified strokes are allotted unique class IDs as 142, 101, 191, and 122 respectively. The information about this sample handwritten word has been recorded in a complete sequence as adopted by the user to form this *Gurmukhi* word. Table 3.4(a) presents the number of strokes used to input *Gurmukhi* word /ਉੜੇ/ with its raw points stored in .XML format, while Table 3.4(b) presents the annotation <classId>142</classId> of the same in .XML format.

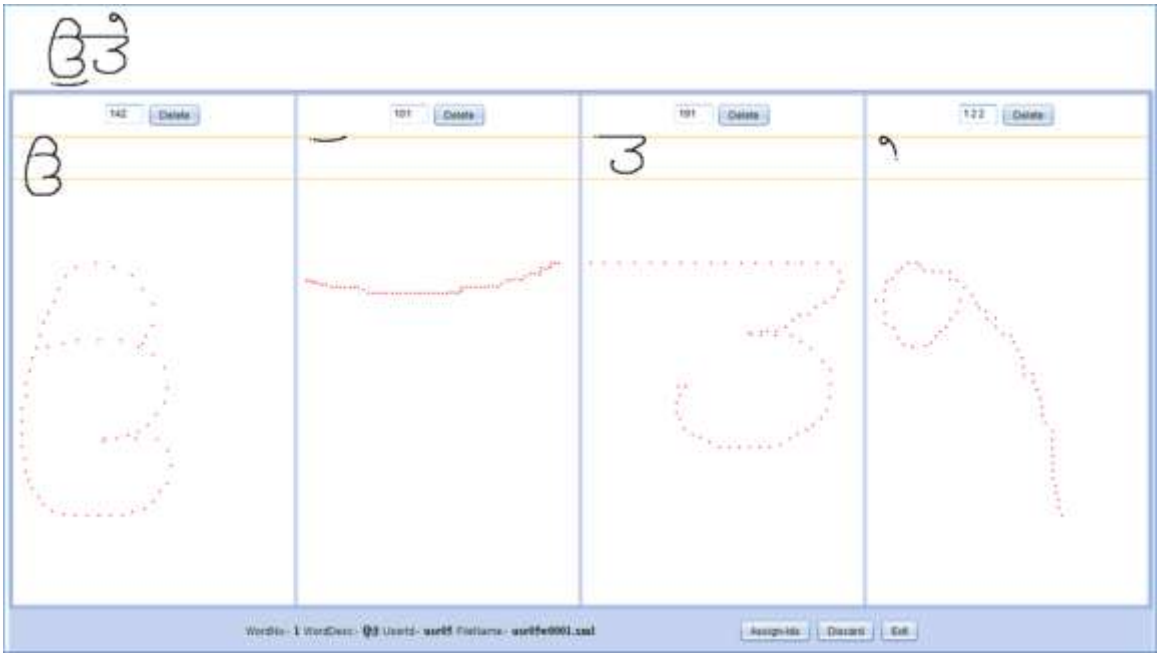


Figure 3.3: Screen shots of annotation of input data

Table 3.4: Extensible markup language (.XML) template of (a) input word with its raw points (b) annotation of input word

(a)

```
<?xml version="1.0" encoding="UTF-8"?>
▼<wordSetDef>
    <wordNo>203</wordNo>
    <wordDesc>ਉੜ</wordDesc>
      <totalStrokes>4</totalStrokes>
      ▼<stroke>
            <strokeNo>1</strokeNo>
            ►<point> . .</point>
      </stroke>
      ▼<stroke>
            <strokeNo>2</strokeNo>
            ►<point>. .  </point>
      </stroke>
      ▼<stroke>
            <strokeNo>3</strokeNo>
            ►<point>. . </point>
      </stroke>
      ▼<stroke>
            <strokeNo>4</strokeNo>
            ►<point> . . </point>
      </stroke>
</wordSetDef>
```

(b)

```
<?xml version="1.0" encoding="UTF-8"?>
▼<wordSetDef>
    <wordNo>203</wordNo>
    <wordDesc>ਉੜ</wordDesc>
      <totalStrokes>4</totalStrokes>
      ▼<stroke>
            <classId>142</classId>
            <strokeNo>1</strokeNo>
            ►<point> . .</point>
      </stroke>
      ▼<stroke>
            <classId>101</classId>
            <strokeNo>2</strokeNo>
            ►<point>. .  </point>
      </stroke>
      ▼<stroke>
            <classId>191</classId>
            <strokeNo>3</strokeNo>
            ►<point>. . </point>
      </stroke>
      ▼<stroke>
            <classId>122</classId>
            <strokeNo>4</strokeNo>
            ►<point> . . </point>
      </stroke>
</wordSetDef>
```

From these collected samples, 96 unique strokes have been identified (including *Gurmukhi* numerals) and categorized into three zones, *viz.* upper, middle, and lower. These unique strokes are presented in Table 3.5. Considering the variant styles of users' handwriting, these strokes lead to the character formation. Table 3.5 also provides a complete detail about the shape of stroke identified from raw *Gurmukhi* words as written by different categories of users. Further, a description of class ID allotted to these strokes has been provided. This table also contains the detail on stroke writing zone, *i.e.*, where the stroke lies in writing a *Gurmukhi* character. This table also carries the list of characters that will make use of that stroke to form a particular *Gurmukhi* character. For example, the stroke () has been identified from the common words written by different users. As per methodology of the study, the strokes with this particular shape are given

with the class ID 151 and listed in middle zone of *Gurmukhi* writing. The possible characters that use this particular stroke in *Gurmukhi* script are ਮ, and ਸ.

Table 3.5: List of *Gurmukhi* strokes used in this work, class IDs, writing zone and list of character(s) using that stroke

Shape of Stroke	Class ID	Stroke Writing Zone	List of character(s) using the stroke
	101	Lower	ੁ, ੂ
	106	Lower	ੂ
	121	Upper	ੳ, ਏ, ਸ, ਹ, ਕ, ਙ, ਚ, ਛ, ਜ, ਝ, ਞ, ਟ, ਠ, ਡ, ਢ, ਣ, ਤ, ਥ, ਦ, ਧ, ਨ, ਫ, ਬ, ਭ, ਯ, ਰ, ਲ, ਵ, ੜ
	122	Upper	ੇ, ੈ, ੰ
	123	Upper	ੈ
	124	Upper	ੱ, ੰ
	126	Upper	ੱ
	128	Upper	ੰ
	133	Upper	ੰ
	141	Middle	ੳ
	142	Middle	ੳ
	144	Middle	ਅ
	145	Middle	ਅ

	146	Middle	ੲ, ਟ, ਣ, ਨ
	147	Middle	ੲ, ਵ, ਣ, ਲ, ੜ
	147	Lower	ੜ
	148	Middle	ੲ, ਝ, ਵ, ਲ
	149	Middle	ੲ, ਲ
	150	Middle	ੲ, ਲ
	151	Middle	ਮ, ਸ
	152	Middle	ਮ, ਸ
	153	Middle	ਸ
	154	Middle	ਸ, ਚ, ਜ, ਥ, ਧ, ਬ, ਯ, ਾ
	155	Middle	ਹ
	156	Middle	ਹ
	157	Middle	ਕ, ਝ
	158	Middle	ਕ, ਝ
	159	Middle	ਧ, ਪ, ਥ, ਖ
	160	Middle	ਥ, ਖ
	161	Middle	ਧ, ਪ, ਥ, ਖ
	162	Middle	ਅ, ਸ, ਖ, ਗ, ਘ, ਚ, ਜ, ਥ, ਧ, ਪ,

			ਬ, ਮ, ਯ, ਾ
	163	Middle	ੳ, ਥ, ਖ
	164	Middle	ਰ, ਗ
	165	Middle	ਰ, ਗ
	166	Middle	ਘ
	167	Middle	ਘ
	168	Middle	ਝ
	169	Middle	ਝ
	170	Middle	ਚ
	171	Middle	ਚ
	172	Middle	ਛ
	173	Middle	ਛ
	174	Middle	ਜ
	175	Middle	ਜ
	176	Middle	ਝ
	177	Middle	ਏ, ਝ, ਲ
	179	Middle	ਵ, ਫ
	180	Middle	ਵ

	181	Middle	ਵ, ਵ
	182	Middle	ਏ, ਟ, ਨ
	183	Middle	ਠ
	184	Middle	ਠ
	185	Middle	ਡ
	186	Middle	ਡ
	187	Middle	ਢ
	188	Middle	ਢ
	190	Middle	ਣ
	191	Middle	ਤ
	192	Middle	ਤ
	193	Middle	ਦ
	194	Middle	ਦ
	195	Middle	ਧ, ਥ
	196	Middle	ਨ
	197	Middle	ਨ, ਲ
	198	Middle	ਨ, ਲ
	200	Middle	ਫ

	201	Middle	ਫ
	202	Middle	ਵ, ਬ
	203	Middle	ਬ
	204	Middle	ਭ
	205	Middle	ਭ
	207	Middle	ਯ
	208	Middle	ਯ
	209	Middle	ਯ
	210	Middle	ਲ
	212	Middle	ੜ
	215	Middle	ਿ
	216	Middle	ੀ
	218	Middle	ਝ
	222	Middle	ਜ
	223	Middle	ਜ
	224	Middle	ਲ
	225	Middle	ਏ, ਲ, ਨ
	226	Middle	ਚ

	351	Middle	೧
	352	Middle	೨
	353	Middle	೩
	354	Middle	೪
	355	Middle	೫
	356	Middle	೬
	357	Middle	೭
	358	Middle	೮, ೯
	359	Middle	೯
	360	Middle	೦
	361	Middle	೬
	362	Middle	೬, ೯

3.3 Support Vector Machine classifier

Support Vector Machine (SVM) is the state of the art classification technique that has been used for recognition of various scripts (Liu *et al.*, 2002; Bahlmann *et al.*, 2002; Shrivastava and Gharde, 2010). SVMs are a set of related supervised learning methods used for classification and regression. Application of SVM has achieved excellent recognition results for numerous pattern recognition problems. SVMs are discriminative classifiers that provide flexible decision boundaries in high dimensional feature spaces. Some of the properties of SVM that form the reason of SVMs' success are optimality of training results and fast training algorithms. Classifiers based on SVMs have some pre-requisites. The feature vector representing a stroke should be of a constant length; and it should ignore the class or instance it represents. Training of a SVM-based classifier involves solving a constrained optimization problem that identifies support vectors.

The standard SVM classifier takes the set of input data and predicts to classify them into one of the two distinct classes. To train an SVM classifier, a particular data set is given and a model is prepared for classification process. The base of this classification is normally the defined decision boundaries where, two or more distinct classes are defined and a boundary is framed to distinct them. Thus, SVMs are binary classifiers that separate linearly any two classes by finding a hyper plane to the closest data points. The outcome of SVMs is based only on the data points that are at the margin and called support vectors.

A classification task usually involves separating the data into training, and testing sets. Each instance in the training set contains one "target value" (*i.e.* the class labels) and few "attributes" (*i.e.* the features or observed variables). The goal of SVMs is to produce a model (based on the training data) which predicts the target values of the test data.

The testing phase involves the evaluation of the following discriminant function of the SVM for every test example:

$$y = sign(\sum_{i=1}^{Nsv} d_i \alpha_i K(x, x_i) + b)$$

where, y = {±1} is the class assigned to the test example, N_{sv} denotes the number of support vectors identified during training, x denotes the feature vector for the test example, x_i represents a support vector, d_i denotes its desired class label, α_i its corresponding Lagrange coefficient, K (•) indicates the chosen kernel function and b denotes the bias value of the hyperplane. One can use one of the following kernel functions.

Linear: $K(x_i, x_j) = (x_i^T x_j)$

Polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$

Gaussian Radial Basis Function (RBF): $K(x_i, x_j) = \exp(-\gamma || x_i - x_j ||^2),\ \gamma > 0$

Sigmoid: $K(x_i, x_j) = tanh(\gamma x_i^T x_j + r)$

Here, γ, r, and d, are kernel parameters.

For the purpose of present study, Radial Basis Function (RBF) kernel has been used for SVM classification. This kernel non-linearly maps samples into a higher dimensional space. So, unlike the linear kernel, it can handle the case when the relation between class

labels and attributes is non-linear. The RBF kernel has fewer numerical difficulties. Experiment results have been obtained by using the open source software LibSVM. The feature values have been scaled from 1 to 9 before training the LibSVM. By default, LibSVM selects the parameter gamma (γ) as 1/*d*, where, *d* stands for the number of features. The main advantage of scaling is that it helps to avoid attributes in greater numeric ranges in smaller numeric ranges which dominate. Another advantage is that it helps to avoid numerical difficulties during the calculations. The classifier can accurately predict the unknown data (*i.e.*, testing data). The cross-validation procedure can also be used here. Two parameters for an RBF kernel, *viz.* C (*penalty factor*) and γ (*learning rate*) are used in cross-validation. Various values of C and γ have been tried and the one with the best cross-validation accuracy is picked. It has been found that exponentially growing sequences of C and γ provide a practical method to identify good parameters (for example, $C = 2^{-5}, 2^{-3}..., 2^{15}$; $\gamma = 2^{-15}, 2^{-13}..., 2^{3}$).

3.4 Testing strategies for recognition

Gurmukhi script consists of a large number of strokes with a high degree of variation that occurs due to different handwriting styles of writers. While collecting the data sample from different categories of users, samples of both characters and words were collected. The reason behind collecting these samples separately from the same user was that he may write a single character in a different style from the one used to write a complete word. In addition to this, other problem related to sequencing and merging during post-processing phase arises, where it is possible that all input strokes may belong to same character, but their sequence of input may be different. Thus, it becomes difficult to merge them. The problem of ambiguity while recognizing the strokes is also of a potential concern.

A novel approach for recognition of *Gurmukhi* script has been proposed in this research work. This approach includes association of strokes in pre-processing; post-segmentation, merging the stroke sets, and sequencing of stroke sets in post processing. Each stoke in a character is assigned a weight based on three components: (*i*) number of strokes in the character, (*ii*) number of input strokes, and (*iii*) number of matched strokes.

In the present work, each stroke is re-sampled into 64 approximately equidistant points. The extracted feature, *i.e.*, the *x* and *y* co-ordinates of each stroke were stored in LibSVM format as $x_1, y_1, x_2, y_2, \ldots, x_{64}, y_{64}$ for recognition in order to obtain the desired class to which a stroke belongs.

Following testing strategies have been implemented in the proposed work.

- Finding the optimal values of parameters to maximize the accuracy of LibSVM classifier using various combinations of γ (*learning rate*) and C (*penalty factor*) in kernel function.
- Recognition of *Gurmukhi* characters and numerals has been carried on the sample of new users.
- A combination of *Gurmukhi* character with vowel modifier has been used to form *Gurmukhi* aksharas.
- Testing has been performed for proposed methodology on complete *Gurmukhi* word formation.

3.5 Chapter summary

This chapter presents an overview of the framework adopted for this research work. The basic methodology used to recognize online handwritten *Gurmukhi* characters has been explained. Although the steps involved in recognition of *Gurmukhi* script are standard ones, yet novel techniques have been used to resolve the issues involved in pre-processing and post-processing. The methodology developed especially for the pre-processing phase; through association of strokes and newly designed post-processing algorithm, differentiates the proposed research work from the earlier ones. A complete detail of the sample data on online handwritten words collected from different categories of users is given in this chapter. In addition to this, the methodology adopted to systematically store and classify the sample data has also been explained. This research study has made use of LibSVM classifier for recognition of online handwritten *Gurmukhi* script. Next chapter provides a detailed discussion on proposed pre-processing methodology and features extracted for recognition of online handwritten *Gurmukhi* script.

CHAPTER 4

PROPOSED PRE-PROCESSING AND ZONE IDENTIFICATION TECHNIQUE FOR ONLINE HANDWRITTEN *GURMUKHI* SCRIPT

Pre-processing, being a preliminary step, serves various purposes that make the recognition process smooth and easy. It is used for removing noisy artifacts from handwriting and also helps in correcting imperfections (Guerfali and Plamondon, 1993). The pre-processing phase is helpful to remove noise and distortions present in input data due to hardware and software limitations during the online handwriting recognition process (Govindaraju and Srihari, 1997; Subrahmonia and Zimmerman, 2000). Such limitations of noise or distortions include abnormal size of text, missing points due to fast writing with the pen, jitter present in text, slant handwriting and uneven distances of points from adjoining positions. These complexities may arise when sometimes digital tablets have low pass hardware filters and as a result stroke shape is present in a jagged form. Similarly, some handwritten strokes contain hooks and duplicate sampled points due to a hesitant writing style. Presence of such noise information is capable of influencing the exploration of stroke profile that affects further processing such as feature extraction and classification. The researchers have applied different pre-processing techniques and their experiments have proved that a significant increase in recognition accuracy is achieved, if systematic pre-processing techniques are adopted (Tappert *et al.*, 1990; Guerfali and Plamondon, 1993; Plamondon and Srihari, 2000; Wink and Roerdink, 2004).

This research work introduces a novel way of pre-processing phase to remove noisy artifacts from online handwritten data. Under the proposed technique, two important steps have been introduced which are necessary to bring the data in required form. These steps form the basis for further pre-processing of data. A brief description of the pre-processing technique is provided in Figure 4.1.

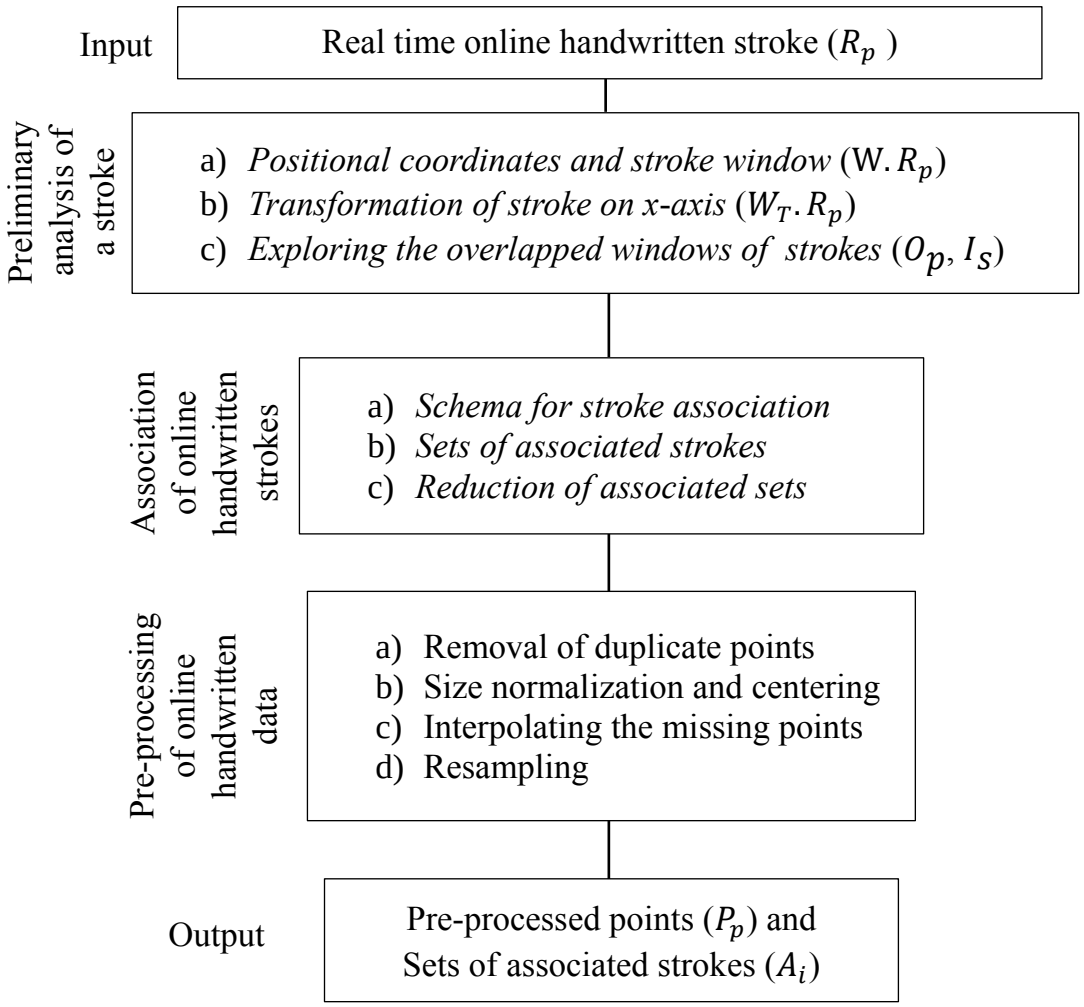


Figure 4.1: Proposed pre-processing phase

4.1 Preliminary analysis of a stroke

In the context of handwriting, a user frames complete textual message in the form of a *sentence*. These *sentences* are governed by syntactic rules whereby different combinations of words of a particular script are used. Words used in a message are formed by sequence of defined vocabulary as mentioned by a particular language. Hence, characters are the smallest unit used to form a recognizable sentence. In the case of handwritten character recognition, this study aims to identify the text from data taken and further process it as a sequence of encoded values for basic characters. The most significant sub unit of a character used in the analysis of online handwriting is called a *stroke*. Sequence of data points drawn by the position of pen tip captured at equal intervals of time leads to the formation of a stroke. Mathematically, a stroke is expressed as:

$$R_p = \{r_{p_1}, r_{p_2}, \dots, r_{p_n}\}$$

Here, R_p denotes the p^{th} input stroke; and n represents the number of raw points captured in p^{th} input stroke. Here, i^{th} point is represented as $r_{p_i} = (x_{p_i}, y_{p_i})$. A stroke window $W.R_p$ can now be formed using points x_{min}, x_{max}, y_{min} and y_{max} corresponding to each R_p.

For a particular character/word under consideration, the generated stroke windows obtained from the user's input strokes are translated such that the bottom of stroke windows is mapped to x-axis. This transformation will result into the formation of $W_T.R_p$ for all p.

Now, the proposed algorithm proceeds to determine the overlapping regions of the identified stroke windows as mapped on the x-axis. Based upon the percentage overlap between various stroke windows the disjoint stroke windows are identified.

For the purpose of recognition of stroke in *Gurmukhi* script, pre-processing of a stroke has been done to obtain the sequence of data points as:

$$P_p = \{p_{p_1}, p_{p_2}, \dots, p_{p_m}\}$$

Here, P_p represents pre-processed points of p^{th} input stroke; and m denotes the number of points in a stroke after pre-processing. In the next sub sections, the steps used in forming stroke window $W.R_p$ from positional coordinates; formation of $W_T.R_p$ from $W.R_p$ and finding the set of pre-processed points P_p are describe in detail. Information extracted in this phase is helpful in the next phases, *i.e.*, feature extraction phase and post-processing phase.

4.1.1 Positional coordinates and stroke window

Characters of *Gurmukhi* script including consonants, vowels, nasals, and numerals have been considered in this research work. For writing each of these characters, the common styles followed by users and the corresponding strokes used to write it have been considered. Shape of a stroke is determined by the sequence of positional coordinates used by the writer. However, a writer uses these sequential strokes as per his handwriting

style. As explained in the previous chapter, a set of unique strokes has been identified and each stroke in this set has been provided with a unique class ID. Figure 4.2 provides example $x - y$ coordinates of an input stroke. In this example, a user writes a stroke of *Gurmukhi* script by starting it from raw point $r_{p_1} = (x_{p_1}, y_{p_1})$ and ends it at $r_{p_n} = (x_{p_n}, y_{p_n})$. The arrows shown on stroke describe the direction of the movement of pen to write this stroke. This stroke can thus be represented as the set of R_p.

$$R_p = \{(x_{p_1}, y_{p_1}), (x_{p_2}, y_{p_2}), \dots, (x_{p_{n-1}}, y_{p_{n-1}}), (x_{p_n}, y_{p_n})\}$$

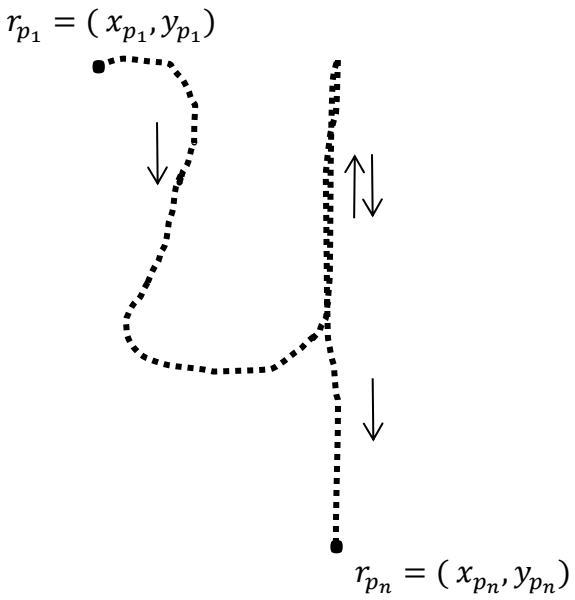
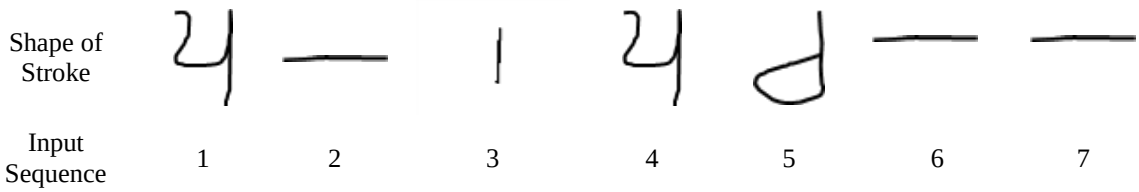


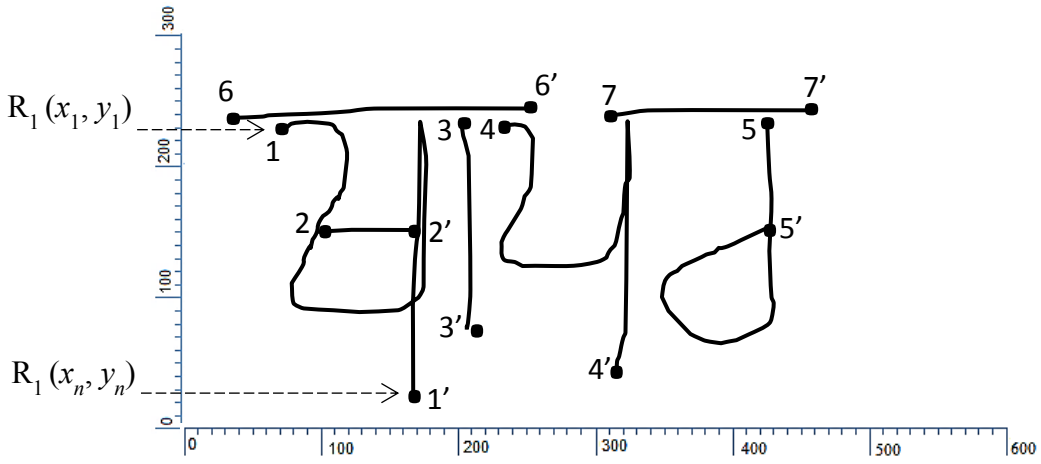
Figure 4.2: Example raw $x - y$ points of an input stroke

We have now calculated x_{min}, x_{max}, y_{min} and y_{max} of this stroke that have further been used to form the overlapping window $W.R_p$ of a stroke.

The process of formation of stroke window $W.R_p$ is now described with the help of the *Gurmukhi* word /ਥਾਪਰ/ (Thapar). This word has been written using seven strokes: $R_1, R_2, R_3, R_4, R_5, R_6$ and R_7 as given in Figure 4.3 (a). As mentioned earlier, these sets contain the raw points used by a writer to form these strokes. Figure 4.3 (b) contain the word actually written by the writer. Here (first, last) points of a stroke are represented by $(i, i'), i = 1, 2, 3, \dots, 7$.



(a)



(b)

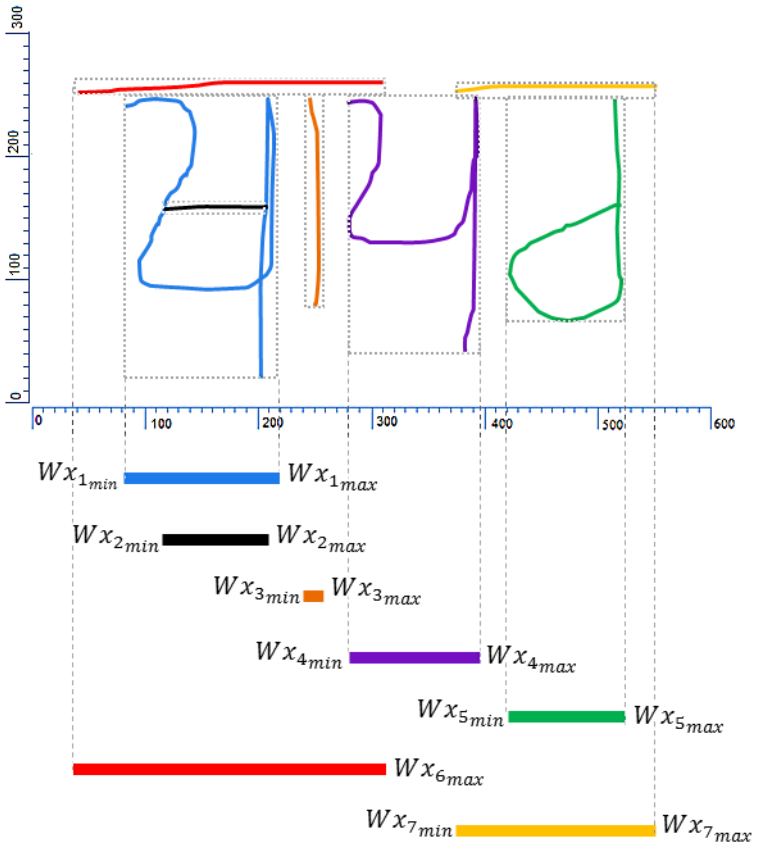
Figure 4.3: Illustration of (a) Sequence of input seven strokes and (b) marked with their initial and final points

Figure 4.4 (a) presents window points on x-axis, whereby $Wx_{1_{min}}$ and $Wx_{1_{max}}$ have been captured for R_1 input stroke on the basis of x_{min} and x_{max}. Similarly, for all the next six input strokes, the respective $Wx_{p_{min}}$ and $Wx_{p_{max}}$ have been extracted. In Figure 4.4 (b), computation of window point's on y-axis has been shown. Here, $Wy_{1_{min}}$ and $Wy_{1_{max}}$ have been computed for R_1 input stroke on the basis of y_{min} and y_{max}. Similarly, for all the next input strokes, their respective $W.R_p$ has been computed.

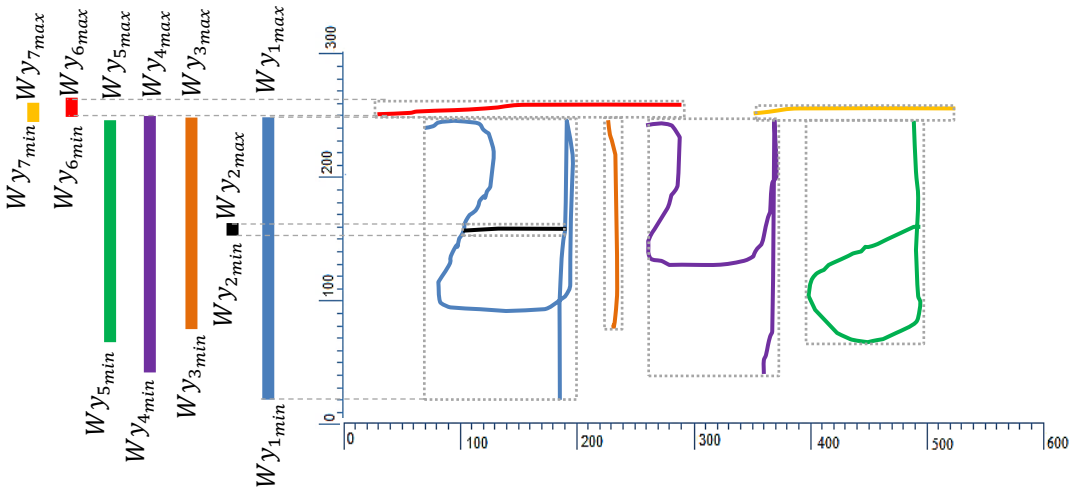
General form of a stroke window $W.R_p$:

$$W.R_p = \{Wx_{p_{min}}, Wx_{p_{max}}, Wy_{p_{min}}, Wy_{p_{max}}\}$$

Here, p represented by $1, 2, 3, \ldots, 7$.



(a)



(b)

Figure 4.4: Illustration of (a) Stroke windows points on *x*-axis (b) stroke windows points on *y*-axis

4.1.2 Transformation of stroke on *x*-axis

A stroke window can be transformed from one domain to another domain with the help of rotation, scaling and translation. The translation has been applied on a stroke in such a way that translated stroke touches *x*-axis. The original stroke and its translation have same shape and size. Algorithm 4.1 contains the steps that have been used in translation process.

A**lgorithm 4.1:** Translation of stroke window on *x*-axis

Input: Extracted window points $Wx_{p_{min}}, Wx_{p_{max}}, Wy_{p_{min}}, Wy_{p_{max}}$ and raw points R_p of p^{th} stroke.

Output: Translated window points $W_T x_{p_{min}}, W_T x_{p_{max}}, W_T y_{p_{min}}, W_T y_{p_{max}}$ and translated raw points R_p^T of p^{th} stroke.

Step 1: *Set* $W_T y_{p_{min}} = Wy_{p_{min}} - y_{p_{min}}$; // set as 0

$W_T y_{p_{max}} = Wy_{p_{max}} - y_{p_{min}}$;

$W_T x_{p_{min}} = Wx_{p_{min}}$;

$W_T x_{p_{max}} = Wx_{p_{max}}$;

Step 2: $\boldsymbol{for}\ all\ (x_{p_i}, y_{p_i})\ of\ R_p\ i = 1\ to\ n\ \boldsymbol{do}$

// Here, *n* represents number of raw points in p^{th} stroke.

$R_p^T(y_{p_i}) = y_{p_i} - y_{p_{min}}$;

$R_p^T(x_{p_i}) = x_{p_i}$;

Step 3: $return\ (W_T x_{p_{min}}, W_T x_{p_{max}}, W_T y_{p_{min}}, W_T y_{p_{max}}, R_p^T)$;

This Algorithm when applied on the stroke windows in Figure 4.4 (a) will result into the strokes as given in Figure 4.5.

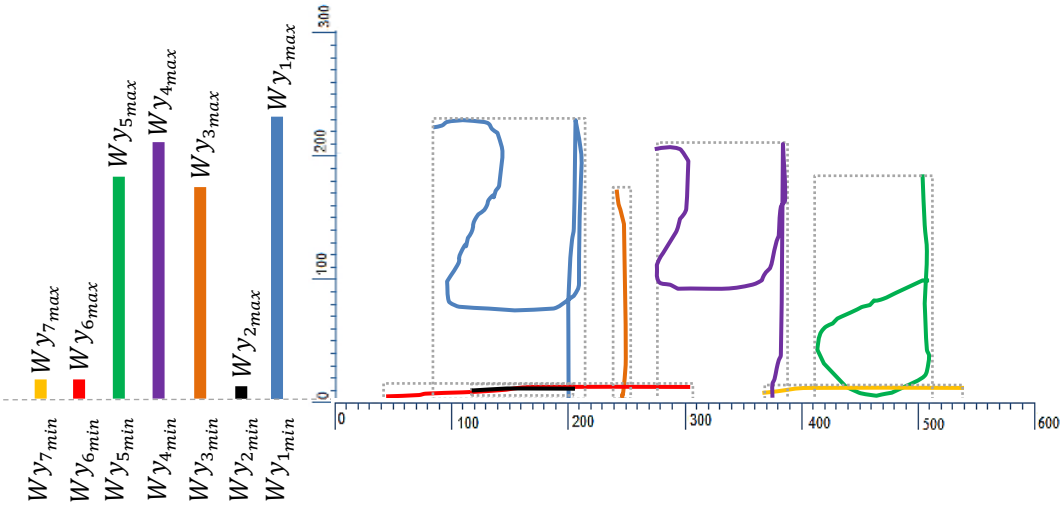


Figure 4.5: Stroke window translation on *x*-axis

4.1.3 Exploring the overlapped windows of strokes

In this section, we have explored the overlapping of the stroke windows in order to extract meaningful information from the same. We have here found the percentage of overlapping between windows and have also found whether a window is completely inside other window or not. Algorithm 4.2(a) finds the percentage of overlap between windows and Algorithm 4.2(b) finds the whether a window is completely within the other window or not. The range of overlapping percentage (O_p) is [0, 100]. A 0 value of O_p indicates that the two windows do not overlap and a value 100 of O_p indicates that two windows completely overlap.

Algorithm 4.2(a): Overlapping window

Input: Extract translated stroke windows points of $W_T.R_i$ of i^{th} stroke and $W_T.R_j$ of j^{th} stroke.

Output: Overlapping percentage (O_p), this ranges from 0 to 100.

Step 1: *Get* $W_T.R_i = \{W_Tx_{i_{min}}, W_Tx_{i_{max}}, W_Ty_{i_{min}}, W_Ty_{i_{max}}\}$

$W_T.R_j = \{W_Tx_{j_{min}}, W_Tx_{j_{max}}, W_Ty_{j_{min}}, W_Ty_{j_{max}}\}$

Step 2: **Compute area**

a) $Overlap = MAX\left(0, MIN\left(W_Tx_{i_{max}}, W_Tx_{j_{max}}\right) - MAX\left(W_Tx_{i_{min}}, W_Tx_{j_{min}}\right)\right);$

b) $Space.R_i = \left(W_Tx_{i_{max}} - W_Tx_{i_{min}}\right);$

c) $Space.R_j = \left(W_T x_{j_{max}} - W_T x_{j_{min}}\right);$

Step 3: $O_p = Overlap/(Space.R_i + Space.R_j - Overlap) * 100;$

Step 4: $return\ (O_p);$

Algorithm 4.2(b) returns true if a window is complete inside another window otherwise it returns false.

Algorithm 4.2(b): Inside window

Input: Extract translated stroke windows points of $W_T.R_i$ of i^{th} stroke and $W_T.R_j$ of j^{th} stroke.

Output: I_s set as *true* if first stroke is inside another or vice versa otherwise *false*.

Step 1: *Get* $W_T.R_i = \{W_T x_{i_{min}}, W_T x_{i_{max}}, W_T y_{i_{min}}, W_T y_{i_{max}}\}$

$W_T.R_j = \{W_T x_{j_{min}}, W_T x_{j_{max}}, W_T y_{j_{min}}, W_T y_{j_{max}}\}$

Step 2: $Space.R_i = \left(W_T x_{i_{max}} - W_T x_{i_{min}}\right);$

$Space.R_j = \left(W_T x_{j_{max}} - W_T x_{j_{min}}\right);$

Step 3: $\boldsymbol{If}\ ((W_T x_{i_{min}} \geq W_T x_{j_{min}})\ \&\&\ (W_T x_{i_{max}} \leq W_T x_{j_{max}}))$

$||\ ((W_T x_{i_{min}} \leq W_T x_{j_{min}})\ \&\&\ (W_T x_{i_{max}} \geq W_T x_{j_{max}}))$

$\boldsymbol{If}\ \left((Space.R_i > Space.R_j)\ AND\ (Space.R_j/Space.R_i > 0.25)\right)$

$return\ (True);$

$\boldsymbol{If}\ \left((Space.R_i \leq Space.R_j)\ AND\ (Space.R_i/Space.R_j > 0.25)\right)$

$return\ (True);$

$return\ (False);$

$else$

$return\ (False);$

The outcomes of these two algorithms are illustrated in Figure 4.6 with the help of word /ਥਾਪਰ/ (Thapar) as given in Figure 4.3. It can be noted from Figure 4.6 that:

i) Second stroke window ($W.R_2$) is inside the first stroke window ($W.R_1$).

ii) $W.R_1$, $W.R_2$ and $W.R_3$ stroke windows are inside the sixth stroke window ($W.R_6$).

iii) Stroke window for the fourth stroke ($W.R_4$) overlaps with sixth and seventh stroke windows $W.R_6$ and $W.R_7$.

iv) Stroke window for fifth stroke ($W.R_5$) is inside the seventh stroke window $W.R_7$.

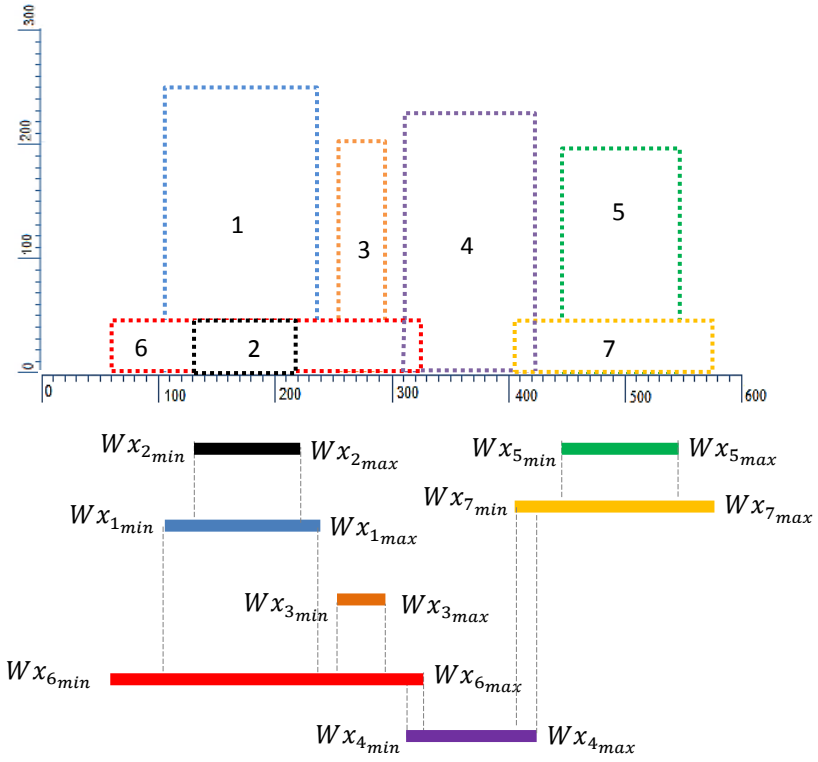


Figure 4.6: Illustration of overlapping and inside stroke windows

The information extracted in this section is useful for finding the association between strokes. This has been discussed in the next section.

4.2 Association of online handwritten strokes

After preliminary analysis of the strokes, an effort has been made to examine the association between the input strokes in this phase of pre-processing. After considering translation and overlapping of input strokes, the next task is to establish a relationship between the current stroke and previous stroke(s). In this work, a new methodology has been proposed to find association between strokes. This methodology consists of three steps: schema for stroke association, sets of associated strokes and reduction of associated sets. Thus, prior to the pre-processing, we have focused on finding the association

between two or more consecutive input strokes that would lead to formulation of a *Gurmukhi* character. A brief description of these three steps is presented in forthcoming subsections.

4.2.1 Schema for stroke association

Schema for stroke association provides a description of logical methodology used for finding the relationship between input strokes. This methodology defines the existence of association on the basis of partial and complete overlapping of strokes. A table is maintained in this step that contains basic information about a stroke and its relationship with other strokes. Basic information includes *stroke sequence number* and *association id*. Here, *stroke sequence number* represents input sequence of a stroke and first input stroke is marked as "1". *Association id* provides the relationship between current stroke and previously written stroke(s). Following association rules have been formed to associate an *association id* to a stroke.

Rule 1 Assign an *association id* "1" to first input stroke R_1.

Rule 2 For strokes $R_2, R_3, \ldots$, decide for the complete / partial overlapping with all previous strokes and assign the *association id(s)* as the *stroke sequence number* of the strokes(s) that have a complete / partial overlapping with these strokes $R_2, R_3, \ldots$

Rule 3 If Rule 2 does not provide an *association id*, we assign the *stroke sequence number* as the *association id* to the current stroke.

Algorithm 4.3 has been proposed to implement the schema for stroke association. Here, we input current stroke R_i as its window points $W_T.R_i$ to verify the association with previous input strokes.

Algorithm 4.3: Schema of Stroke Association

Input: Extract points $W_T x_{p_{min}}, W_T x_{p_{max}}, W_T y_{p_{min}}, W_T y_{p_{max}}$ of translated stroke window $W_T.R_i$ of current input stroke R_i.

Output: Association Table 4.1 with their *stroke sequence number* and *association id*.

Step 1: *Set* $Association_{FLAG} = False$;

Step 2: // **Association Rule 1**

$If\ (Assocation\ Table\ is\ empty)$

{

$\mathrm{A_{Table}}$ [$Stroke\ sequence\ number$] = 1;

A_{Table} [$Association\ Id$] = 1;

}

$else$

// **Association Rule 2**

// Here *count* is total number of previously input strokes.

$for\ all\ j = 1\ to\ count\ do$

{

$If\ \left(\mathrm{INSIDE\_WINDOW}(W_T.R_i, W_T.R_j)\right)$

{

// set i the *sequence number* of current input stroke

A_{Table} [$Stroke\ sequence\ number$] = i;

//set j as *association id* because current input stroke is associated with j^{th} stroke

A_{Table} [$Association\ Id$] = j;

Set $Association_{FLAG} = True$;

}

$esle\ if\ \left(\mathrm{OVERLAPPING\_WINDOW}(W_T.R_i, W_T.R_j) \geq threshold\right)$

{

// set i the *sequence number* of current input stroke

A_{Table} [$Stroke\ sequence\ number$] = i;

//set j as *association id* because current input stroke is associated with j^{th} stroke

A_{Table} [$Association\ Id$] = j;

Set $Association_{FLAG} = True$;

}

}

Step 3: // **Association Rule 3**

$If\ (Association_{FLAG}\ is\ False)$

{

// set i the *sequence number* of current input stroke

$A_{Table}\ [\ Stroke\ sequence\ number] = i;$

//set i as *association id* with because not associated with any previously input strokes

$A_{Table}\ [\ Association\ Id] = i;$

}

Step 4: *Exit*

This algorithm is now explained with the help of an example *Gurmukhi* script word /ਗੁਰਮੁਖੀ/. As shown in Figure 4.7, a writer has used eleven strokes to write this word. The strokes are shown in different colors to distinguish them. Sequence of writing these strokes has been provided with input *sequence number* through 1 to 11. The details pertaining to these sequentially input strokes have been shown in separate Tables 4.1(a)-(j) and final association results are presented in Table 4.1(k). The methodology adopted for recording the information and for verifying association is described below:

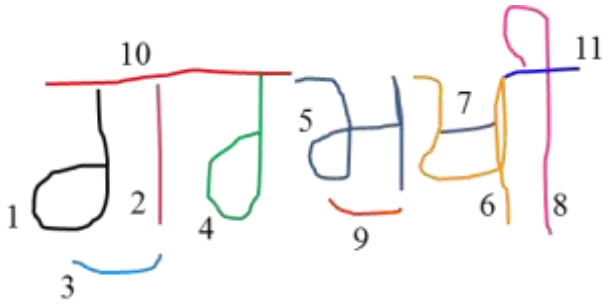


Figure 4.7: Sequence of input strokes for forming the word /ਗੁਰਮੁਖੀ/

With the input of first stroke, *stroke sequence number* has been marked as $'1'$ and *association id* $'1'$ has been assigned to this stroke.

Table 4.1(a): Association table with one stroke

Stroke sequence number	Shape of stroke	*Association id*
1		1

Now, *association id* is assigned to second stroke using Algorithm 4.3. Rule 3 shall be used here to assign the *association id* as the *stroke sequence number* of current input stroke and hence, marked ′2′ as shown in Table 4.1(b).

Table 4.1(b): Association table formed with two strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2

Next, we consider the stroke with sequence number ′3′. Algorithm 4.3 is again used to assign *association id(s)* to this stroke. The decision of partial/complete overlapping of this stroke shall invoke Rule 2 of Algorithm 4.3. This overlapping is first checked with stroke ′1′ and then with stroke ′2′. This schema shall look like the one given in Table 4.1(c).

Table 4.1(c): Association table formed with three strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2
3		1
3		2

Tables 4.1(d) – 4.1(k) contains the instances of schema when 4, 5, 6, 7, 8, 9, 10 and 11 strokes, respectively, have been taken as the input.

Table 4.1(d): Association table formed with four strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2
3		1
3		2
4		4

Table 4.1(e): Association table formed with five strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2
3		1
3		2
4		4
5		5

Table 4.1(f): Association table formed with six strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2
3		1
3		2
4		4
5		5
6		6

Table 4.1(g): Association table formed with seven strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2
3		1
3		2
4		4
5		5
6		6

7		6

Table 4.1(h): Association table formed with eight strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2
3		1
3		2
4		4
5		5
6		6
7		6
8		8

Table 4.1(i): Association table formed with nine strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2
3		1
3		2
4		4
5		5
6		6
7		6
8		8
9		5

Table 4.1(j): Association table formed with ten strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2

Table 4.1(k): Final association table formed with eleven strokes

Stroke sequence number	Shape of stroke	*Association id*
1		1
2		2

3		1	3		1
3		2	3		2
4		4	4		4
5		5	5		5
6		6	6		6
7		6	7		6
8		8	8		8
9		5	9		5
10		1	10		1
10		2	10		2
10		3	10		3
10		4	10		4
			11		8

4.2.2 Sets of associated strokes

Set of associated strokes for a given *Gurmukhi* word is obtained with the help of the set of *stroke sequence numbers* (S) and the set of *association ids* (A). *Stroke sequence number* and *association id* are two different identities having interdependency over each other. In the proposed approach, A is a subset of S as association is derived on the basis of set of strokes. A relation defined on them allows the formation of sets of associated strokes. The *association ids* and *stroke sequence numbers* of strokes constitute a relation R over $S \times A$. For this relation, the associated matrix $M_R = \{m_{ij}\}$ can be computed as:

$$m_{ij} = \begin{cases} 1, & if\ (s_i, a_j) \in R \\ 0, & otherwsie \end{cases}$$

Here, $s_i's$ are the *stroke sequence numbers* and $a_j's$ are *association ids* of the strokes in a given *Gurmukhi* word. This can be noted that $s_i \geq a_j$ for all the elements of relation R. This concept has been used to form the sets of associated strokes for *Gurmukhi* word under consideration.

Using association rules defined in Section 4.2.1, Table 4.1(k) was formed for the word /ਗੁਰਮੁਖੀ/ (*Gurmukhi*). This table contains the columns representing *stroke sequence number*, shape of stroke, and *association id*. Using this table, the relation R (defined over $S \times A$) can be expressed as:

$R = \{(1,1),\ (2,2),\ (3,1),\ (3,2),\ (4,4),\ (5,5),\ (6,6),\ (7,6),\ (8,8),\ (9,5),\ (10,1),\ (10,2),\ (10,3),\ (10,4),\ (11,8)\}$.

Consequently, M_R for this relation shall be:

$M_R =$

		Association id						
		1	**2**	**3**	**4**	**5**	**6**	**8**
Stroke sequence number	**1**	1	0	0	0	0	0	0
	2	0	1	0	0	0	0	0
	3	1	1	0	0	0	0	0
	4	0	0	0	1	0	0	0
	5	0	0	0	0	1	0	0
	6	0	0	0	0	0	1	0
	7	0	0	0	0	0	1	0
	8	0	0	0	0	0	0	1
	9	0	0	0	0	1	0	0
	10	1	1	1	1	0	0	0
	11	0	0	0	0	0	0	1
Associated Sets (A_i)		A_1	A_2	A_3	A_4	A_5	A_6	A_8

Seven associated sets $A_1, A_2, A_3, A_4, A_5, A_6,$ and A_8 are formed using this matrix M_R. These sets of associated strokes and their corresponding *stroke sequence numbers* are:

$A_1 = \{1, 3, 10\}$, $A_2 = \{2, 3, 10\}$, $A_3 = \{10\}$, $A_4 = \{4, 10\}$, $A_5 = \{5, 9\}$, $A_6 = \{6, 7\}$, and $A_8 = \{8, 11\}$. Now, union of *association ids* is carried out with corresponding sets of associated strokes in order to obtain final elements of associated sets. The elements in the sets of associated strokes will thus be:

$A_1 = \{1\} \cup \{1, 3, 10\} = \{1, 3, 10\}$, $A_2 = \{2\} \cup \{2, 3, 10\} = \{2, 3, 10\}$, $A_3 = \{3\} \cup \{10\} = \{3, 10\}$, $A_4 = \{4\} \cup \{4, 10\} = \{4, 10\}$, $A_5 = \{5\} \cup \{5, 9\} = \{5, 9\}$, $A_6 = \{6\} \cup \{6, 7\} = \{6, 7\}$, and $A_8 = \{8\} \cup \{8, 11\} = \{8, 11\}$.

4.2.3 Reduction of associated sets

The sets of associated strokes are further reduced. This is accomplished using a very simple strategy. If set of associated strokes A_i is a subset of A_j, we retain set A_j for further processing. Algorithm 4.4 has been used to implement this concept.

Algorithm 4.4: Reduction of associated sets

Input: All associated sets with marked as A_i.

Output: Reduced association sets (shown in Table 4.2)

Step 1: ***Do*** $for\ all\ \ i = 1$ ***to*** end

// for this example $i = 1, 2, 3, 4, 5, 6, 8.$

Do $for\ all\ \ j = i + 1$ ***to*** end

$if\ (A_i == NULL)\ then$

$break;$

$else$

$set\ D\ =\ A_i\ \cap\ A_j$

$if\ (D\ ==\ A_j)\ then$

$A_i\ =\ A_i\ \cup\ A_j$

$A_j\ =\ NULL;$

$end\ if$

$end\ if$

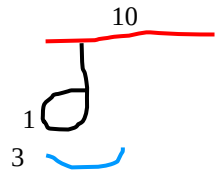
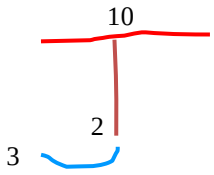
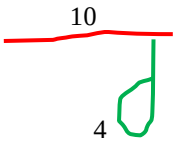
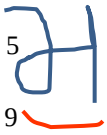
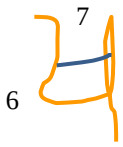
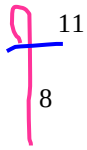
end for

end for

Step 2: ***Exit***

In section 4.2.2 seven associated sets were extracted. Out of those seven sets, set A_3 is a subset of A_1 and, also the subset of A_2. Using proposed Algorithm 4.4, A_3 is merged with A_1. This resulted in reduction of six associated sets *i.e.* A_1, A_2, A_4, A_5, A_6, and A_8. These sets contain the information pertaining to *stroke sequence number* associated with each other. In Table 4.2 we have provided that for a particular set like A_1, three *stroke sequence number* exist, *viz.* 1, 3 and 10.

Table 4.2: Final associated sets corresponding with their shapes

$A_1 = \{1, 3, 10\}$	$A_2 = \{2, 3, 10\}$	$A_4 = \{4, 10\}$	$A_5 = \{5, 9\}$	$A_6 = \{6, 7\}$	$A_8 = \{8, 11\}$
					

4.3 Pre-processing of online handwritten data

Pre-Processing is required to remove the noise or distortion from the input data which encounters certain hardware/software limitations. This noise is available in the input data as sharp edges, non-centred word, and unequal size of input character (Beigi *et al.*, 1994a; Jaeger *et al.*, 2003). Thus, pre-processing is introduced to bring input data in symmetry and get this raw data ready for feature extraction phase. A study by Jaeger *et al.* (2001) has provided that pre-processing steps in online handwriting recognition help significantly to improve the overall recognition rate. However, over a period few research studies have discussed major issues and challenges of pre-processing phase of on-line handwriting recognition for various scripts (Guerfali and Plamondon, 1993; Beigi, 1996; Beigi *et al.*, 1994a; Nagy, 2000). Thus, as already discussed in the previous sections raw points R_p will be transformed into pre-processed points P_p that will further help us to identify important features. Pre-processing phase of online handwritten character recognition mainly includes four common steps, *viz.* removal of duplicate points, size normalization and centring, interpolating missing points, and resampling. A detailed application of these pre-processing steps has been discussed in the forthcoming sub sections.

Pre-processing is an important step in the online handwritten character recognition process. When online handwritten data is collected, it is initially stored on the basis of movements of the pen at various positions. Further, based on sequential strokes input by the writer, the final picture of a character appears before us. Figure 4.8 provides a sample screen shot view of input raw stroke used by an individual writer to write a *Gurmukhi* character /ਪ/. Whatever input raw stroke is provided by the writer, it is stored with n number of points and will be presented as vector R_p as mentioned in section 4.1. Here, n is not fixed and may vary from stroke to stroke. Even if the same stroke is written again by the same user, n may be different for both the strokes written by the user. After introducing all the four important steps of pre-processing on this R_p, we get vector of pre-processed points P_p. In this, P_p we will get fixed m number of points. For the present research work, each input stroke stands fixed with $m = 64$.

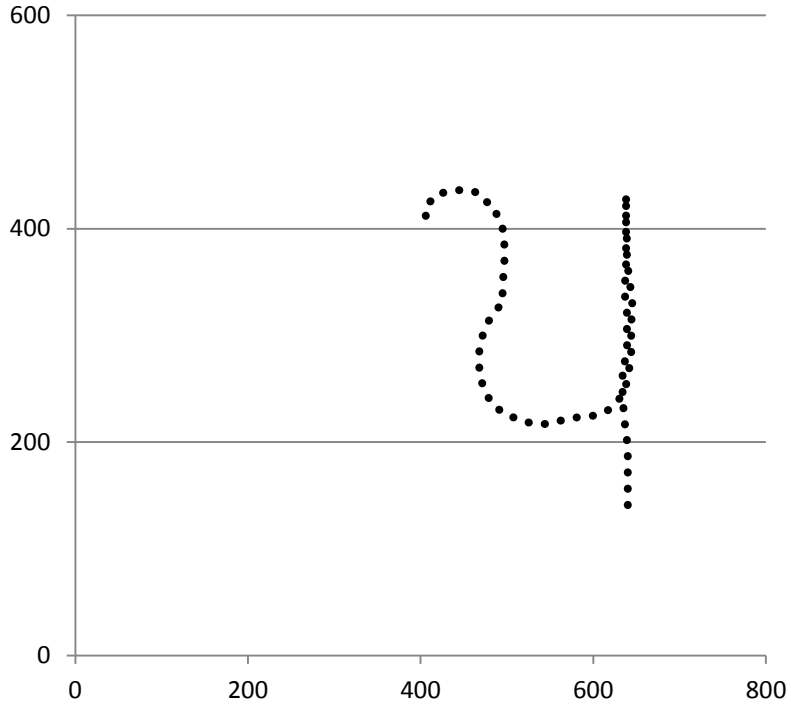


Figure 4.8: Handwritten stroke before pre-processing

4.3.1 Removal of duplicate points

In this phase of pre-processing, the duplicate points have been identified and eliminated, where the coordinates of any two points in a stroke were the same. The problem of duplicate points in writing a character arises when stroke is overlapped due to its unique

shape or character. As a result, the pixels capture duplicate points while writing the input stroke which need to be removed. Figure 4.9 presents a brief view of duplicate points identified while writing a particular stroke to form *Gurmukhi* character /ਪ/. Duplicate points originating in writing this character have been highlighted. However, this process does not change the shape of a stroke.

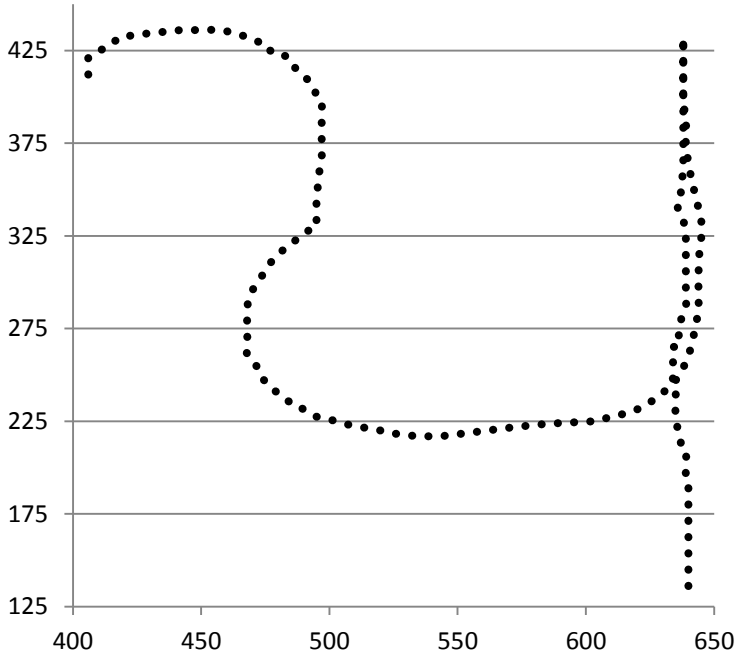


Figure 4.9: A stroke of *Gurmukhi* character /ਪ/ showing removal of duplicate or overlapped points

Algorithm 4.5 has been applied for removal of duplicate points present in the input data. In this Algorithm, n numbers of points have been input as R_p; and an attempt has been made to identify the duplicate points that have similar x, y coordinates. Output of this algorithm is obtained as P_p^{dup} in which we get n or less stroke points of input stroke R_p.

Algorithm 4.5: Removal of duplicate points

Input: Raw points of p^{th} stroke $R_p = \{(x_{p_1}, y_{p_1}), (x_{p_2}, y_{p_2}), \dots, (x_{p_{n-1}}, y_{p_{n-1}}), (x_{p_n}, y_{p_n})\}$

Output: Array of removed duplicate points P_p^{dup} of input stroke R_p.

Step 1: Store raw points of input stroke in array P_p^{dup}.

Step 2: $for\ all\ \ i = 1\ to\ n\ do$

$$for\ all\ \ j = i + 1\ to\ n\ do$$

$$if \begin{pmatrix} P_p^{dup}(x_{p_i}) == P_p^{dup}\left(x_{p_j}\right)\ AND \\ P_p^{dup}(y_{p_i}) == P_p^{dup}\left(y_{p_j}\right) \end{pmatrix}$$

Delete $P_p^{dup}\left(x_{p_j}\right)\ and\ P_p^{dup}\left(y_{p_j}\right)$

Step 3: $\boldsymbol{return}\ (\boldsymbol{P_p^{dup}})$

4.3.2 Size normalization and centering

Every writer has a unique style of writing the text. These different handwriting styles of users create the problem of variation in size of input. Hence, normalization is required as the size of handwritten character will vary from person to person and even with same person from time to time. It normalizes every stroke to the same size; centers to a constant frame; and places the text at a fixed distance from the origin. So, while collecting the sample data, the writers were at liberty to use their free style handwriting. The problem that emerged was that every writer wrote a sample character in a different size. Here, the system's robustness could not be assumed to understand this variation in size of input characters. So, to be on the safer side, the sample characters were re-sized to bring them to some standard measure. Further, it was considered that proportion between horizontal and vertical components of input data comprised of a basic characteristic of the character. Hence, it was not possible to move ahead with skewing or stretching the input data in order to re-size it. Some existing research studies have proposed size normalization that introduces scaling based on the height of stroke (Aparna *et al.*, 2004; Joshi *et al.*, 2005). However, it could not be applied on *Gurmukhi* script because of ascender and descender regions in the character that result in variation in the height of a character.

Size normalization and centering are important steps that contribute largely in the recognition process. For this purpose, initially, a comparison of input stroke's border frame with standard fixed size frame was carried out. This resulted in moving the stroke at the assumed center location. The steps are now performed to render the stroke in a size of 300×300. Therefore, any stroke entered as less than or more than the proposed size will be changed into a fixed size of 300×300. Figure 4.10 provides a view of unique

size of strokes input by the writers and how through size normalization it has been brought back to the proposed size. In the first case, a writer has such writing skills that he normally uses a small size stroke to write a character as shown in Figure 4.10 (a). Through the size normalization and centering process, the input character is brought to a standard size. A view of normalized character at standard 300×300 pixels is provided in Figure 4.10 (b).

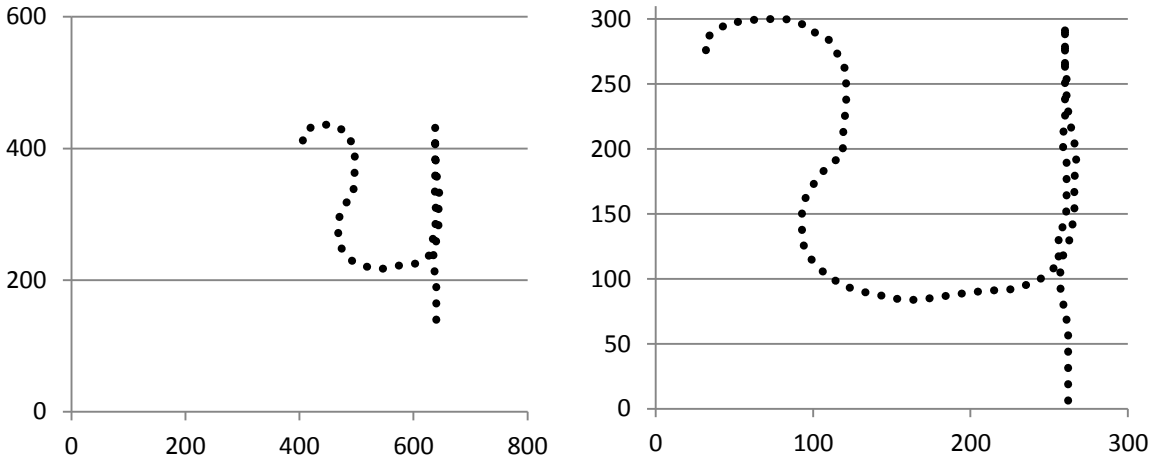


Figure 4.10: Handwriting character (a) before and (b) after normalization and centering

Figure 4.11(a) highlights the input stroke with raw points plotted in a vertical line; and horizontal line is displayed in Figure 4.11(b). Initially, the previously proposed algorithm for normalization and centering was applied (Sharma *et al.*, 2009). However, when this algorithm was applied, it give rise to the same problem that output obtained at 300×300 window was of small size especially when a user wrote in a horizontal/vertical line than the size of this 300×300 window. This problem is highlighted in Figures 4.11(c) and Figure 4.11(d). In such a case, the existing algorithm was not able to scatter the small size input stroke. Thus, Algorithms 4.6 and 4.7 were applied to resolve the problem. The output so obtained is displayed in Figures 4.11(e) and (f).

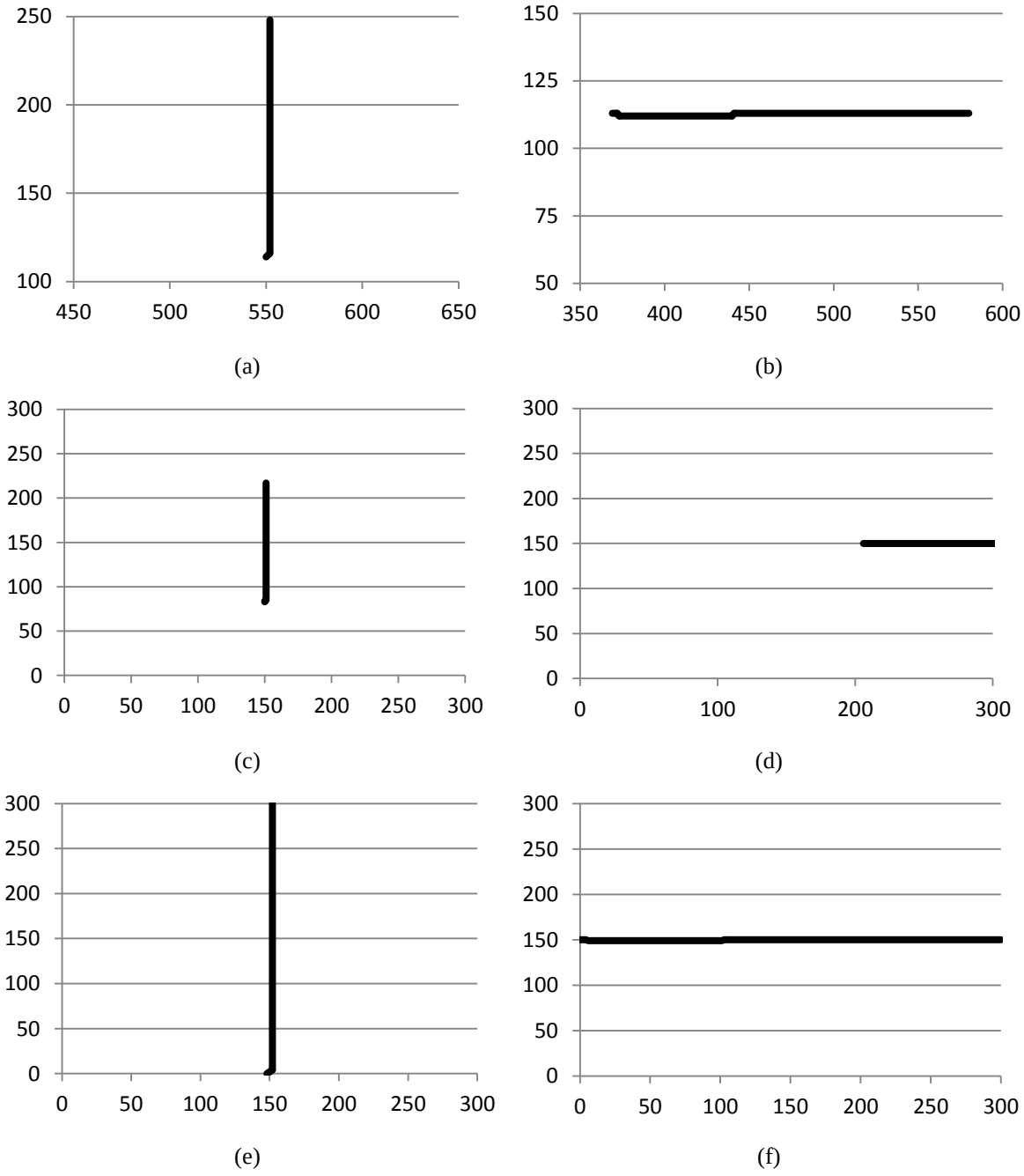


Figure 4.11: Handwritten stroke (a) and (b) are raw points, (c) and (d) are after normalization and centering by existing algorithms (Burr, 1983), (e) and (f) are proposed normalization and centering algorithms

The algorithms used for size normalization (Algorithm 4.6) and centering (Algorithm 4.7) are explained below.

Algorithm 4.6: Normalization

Input: Removed duplicate points of p^{th} stroke $P_p^{dup} = \{(x_{p_1}, y_{p_1}), (x_{p_2}, y_{p_2}), \ldots, (x_{p_{n-1}}, y_{p_{n-1}}), (x_{p_n}, y_{p_n})\}$

Output: Array of normalized points P_p^{norm} of input stroke P_p^{dup}.

Step 1: Set $\mathrm{X}_{frame} = 300$ and $\mathrm{Y}_{frame} = 300$.

Step 2: Find the maximum of *x*-axis and *y*-axis co-ordinate points from P_p^{dup}, as x_{min}, x_{max}, y_{min} and y_{max}.

Step 3: **Compute**

$S_x = \mathrm{X}_{frmae} / (x_{max} - x_{min})$;

$S_y = \mathrm{Y}_{frmae} / (y_{max} - y_{min})$;

Step 4: // Take minimum value from between S_x *and* S_y as scaling factor.

$fact = \mathrm{MIN}(S_x, S_y)$

Step 5: ***for*** $all \;\; i = 1 \; to \; n$ ***do***

// where, $P_p^{dup}(x_i)$ is the i^{th} point in *x*-axis.

$P_p^{norm}(x_i) = P_p^{dup}(x_i) - x_{min}) * fact$;

// where, $P_p^{dup}(y_i)$ is the i^{th} point in *y*-axis.

$P_p^{norm}(y_i) = P_p^{dup}(y_i) - y_{min}) * fact$;

Step 6: $return(P_p^{norm})$

Algorithm 4.7: Centering

Input: Normalized points of p^{th} stroke $P_p^{norm} = \{(x_{p_1}, y_{p_1}), (x_{p_2}, y_{p_2}), \ldots, (x_{p_{n-1}}, y_{p_{n-1}}), (x_{p_n}, y_{p_n})\}$

Output: Array of centering points P_p^{cent} of input stroke P_p^{norm}.

Step 1: Find the maximum of *x*-axis and *y*-axis co-ordinate points from P_p^{norm}, as x_{min}, x_{max}, y_{min} and y_{max}.

Step 2: ***for*** $all \;\; i = 1 \; to \; n$ ***do***

// where, $P_p^{norm}(x_i)$ is the i^{th} point in *x*-axis.

$P_p^{cent}(x_i) = P_p^{norm}(x_i) + ((\mathrm{X}_{frame} - P_p^{norm}(x_{max}))/2;$

// where, $P_p^{norm}(y_i)$ is the i^{th} point in *y*-axis.

$$P_p^{cent}(y_i) = P_p^{norm}(y_i) + ((\mathrm{Y}_{frame} - P_p^{norm}(y_{max}))/2;$$

Step 3: $return(P_p^{cent})$

4.3.3 Interpolating the missing points

When a writer writes a character in hurried manner, it creates the problem that some points of a stroke may get missed. Bezier and B-Spline are the two important techniques used to calculate the missing points. For the purpose of this research study, Bezier interpolation technique has been used as it helps in identifying the missing points from the fixed number of available points. It makes use of a set of four consecutive points to form a Bezier curve; and the next set of four points provides the next Bezier curve. Algorithm 4.8 has been applied in this research work for interpolating the missing points. In this algorithm, we input P_p^{cent} points of a stroke that we got after normalization and centering step. Applying this algorithm, we get vector P_p^{inter} which includes interpolated missing points. Here, we get output as P_p^{inter} which has more number of stroke points than input P_p^{cent}.

Algorithm 4.8: Interpolating missing points:

Input: Normalized points of p^{th} stroke $P_p^{cent} = \{(x_{p_1}, y_{p_1}), (x_{p_2}, y_{p_2}), \ldots, (x_{p_{n-1}}, y_{p_{n-1}}), (x_{p_n}, y_{p_n})\}$

Output: Array of centering points P_p^{inter} of input stroke P_p^{cent}.

Step 1: *Set* $t = Total\ number\ of\ points\ in\ stroke\ P_p^{cent}$.

Step 2: $\boldsymbol{for}\ (i = 1;\ i \leq t; i++)$

$Repeat\ \left((P_p^{cent}(x_i), (P_p^{cent}(x_{i+1})) > 1\right))$

Call Bezier $\left(P_p^{cent}(x_i), P_p^{cent}(x_{i+1}), P_p^{cent}(x_{i+2}), P_p^{cent}(x_{i+3})\right)$;

a) *Upgrade* P_p^{inter} ;

b) Set $P_p^{inter} = P_p^{inter} + 1$;

Step 3: *Exit*

Function Bezier $\left(P_p^{cent}(x_i), P_p^{cent}(x_{i+1}), P_p^{cent}(x_{i+2}), P_p^{cent}(x_{i+3})\right)$

$\boldsymbol{for}\ (u = 0.1;\ u < 1.0; u = u + 0.1)$

{

Calculate x co-ordinate of new points as

$$(1-u)^3 P_p^{cent}(x_i) + 3u(1-u)^2 P_p^{cent}(x_{i+1}) + 3(1-u)u^2 P_p^{cent}(x_{i+2}) + u^3 P_p^{cent}(x_{i+3})$$

Calculate y co-ordinate of new points as

$$(1-u)^3 P_p^{cent}(y_i) + 3u(1-u)^2 P_p^{cent}(y_{i+1}) + 3(1-u)u^2 P_p^{cent}(y_{i+2}) + u^3 P_p^{cent}(y_{i+3})$$

}

Figure 4.12 provides a sample description of interpolated missing points got after the use of above algorithm. In this particular stroke, we have input 171 raw points, and after using the interpolation methodology we got 901 points.

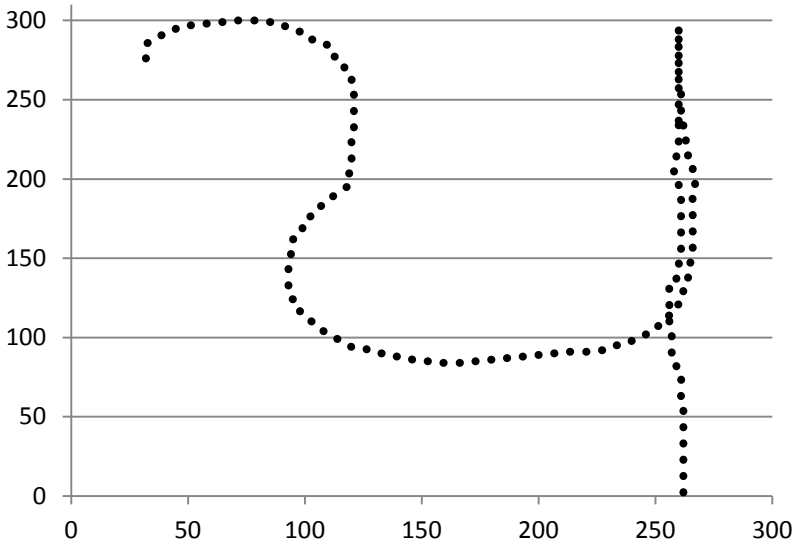


Figure 4.12: A handwritten stroke of *Gurmukhi* script interpolated missing points

4.3.4 Resampling

Resampling of points involves the points in the list to be equidistant from neighboring points. Equidistant resampling is done to bring each stroke at an equal interval in space along with its trajectory. After bringing uniform spacing between the data points, the representation of a stroke is proposed as hereunder:

$$P_p = \{p_{p_1}, p_{p_2}, \ldots, p_{p_m}\}$$

Here, P_p represents resampled pre-processed points of p^{th} input stroke; and m denotes the number of points in a stroke after pre-processing. In this research study, m is defined as 64. It means that, 64 (*x*, *y*) points for each stroke are fixed and these are equidistantly placed to a great extent. Algorithm 4.9 has been applied for resampling of data points and to get the output as resampled 64 data points. The effect of resampling is shown in Figure 4.13 where distance between the two successive data points is uniform.

Algorithm 4.9: Resampling

Input: Interpolated points of p^{th} stroke $P_p^{inter} = \{(x_{p_1}, y_{p_1}), (x_{p_2}, y_{p_2}), \ldots, (x_{p_{n-1}}, y_{p_{n-1}}), (x_{p_n}, y_{p_n})\}$

Output: Array of resampled pre-processed $P_p = \{(x_1, y_1), (x_2, y_2), \ldots\ldots, (x_{63}, y_{63}), (x_{64}, y_{64})\}$ of input stroke P_p^{inter}.

Step 1: Calculate *P* as a total number of points after process of interpolating the missing points.

Step 2: *For getting the* 64 *points calculate* $N = ROUNDUP(P/64)$;

Step 3: $for\ (i = 1; i \leq 64\ ; i++)$

a) *For all the points of a stroke in a list, compute the average from each N point of* $P_p^{inter}(x_i)$ and $P_p^{inter}(y_i)$

b) $Reassign\ values\ to\ P_p(x_i)\ and\ P_p(y_i)$;

Step 4: $return(P_p)$

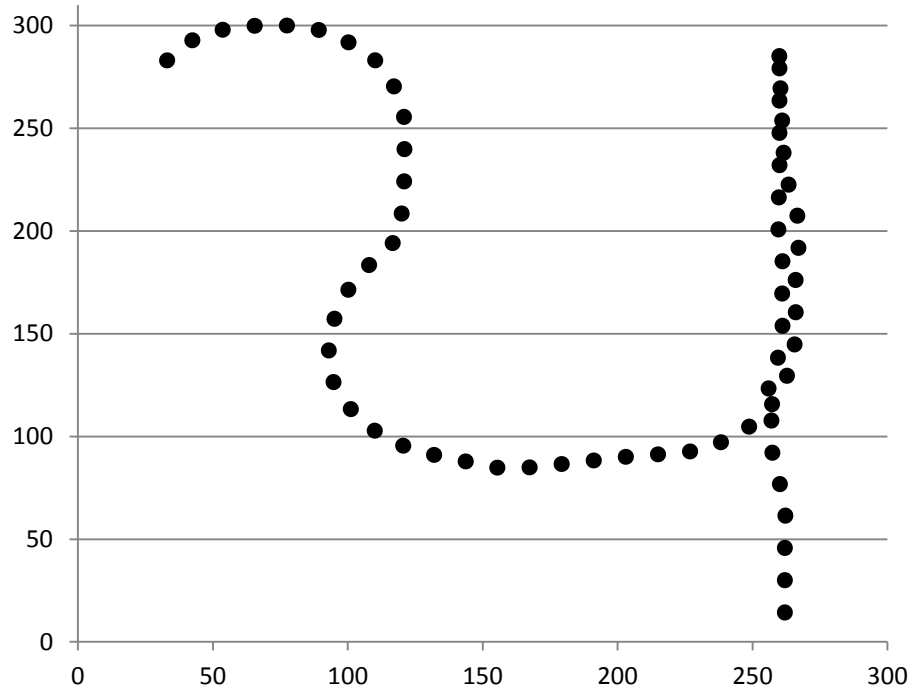


Figure 4.13: Resampling of data points

4.4 Zone identification technique for online handwritten *Gurmukhi* script

In order to identify the stroke's zone of each association, the writing space is divided into three zones namely, upper, middle and lower zone. The details about different zones have already been introduced in Section 1.3. All the strokes characterized from their associated sets are mapped to their respective zones based on their range. This zone identification provides the basic requirements for LibSVM classifier which finally helps in recognition of characters. The zone identification is performed on the basis of maximum and minimum range of each stroke on *y*-axis. This is performed in order of presence of stroke in its corresponding associated sets. For better understanding, an example demonstrating the zone identification of a character is shown below. From the defined rules, *strokes sequence number* $'1'$ and $'2'$ falls in middle zone, *strokes sequence number* $'3'$ and $'4'$ lies in upper and lower zones, respectively. This allows formations of a set which defines the zone division for all the strokes in a given word.

Algorithm 4.10: Zone Identification

Input: Associated set A_i

Output: Marked zone identification of each element (stroke) in A_i

Step 1: Compute Y_{max} and Y_{min} from all the elements in A_i

Step 2: Compute $M_{span} = Y_{max} - Y_{min}$

Step 3: Zone Identification

a) Upper Zone (U_Z)

$U_{max} = Y_{max}$

$U_{min} = Y_{max} - M_{span} * 20/100$

b) Middle Zone (M_Z)

$M_{max} = U_{min}$

$M_{min} = Y_{max} - M_{span} * 80/100$

c) Lower Zone (L_Z)

$L_{max} = M_{min}$

$L_{min} = Y_{min}$

Step 4: Let $n =$ count (A_i) // number of strokes in A_i

Step 5: ***for*** all n $(j = 1\ to\ n)$

$if\ (S_j(y_{max}) \leq U_{max}\ \text{AND}\ S_j(y_{min}) > U_{min})$

$S_j.Zone = \text{Upper}$

$else\ if\ (S_j(y_{max}) \leq M_{max}\ \text{AND}\ S_j(y_{min}) \geq M_{min})$

$S_j.Zone = \text{Middle}$

$esle\ if\ (S_j(y_{max}) < L_{max}\ \text{AND}\ S_j(y_{min}) \geq L_{min})$

$S_j.Zone = \text{Lower}$

$esle\ if\ (S_j(y_{max}) \geq U_{min}\ \text{AND}\ S_j(y_{min}) \geq M_{min})$

$if\ \left((S_j(y_{max}) - U_{min}) > (M_{max} - S_j(y_{min}))\right)$

$S_j.Zone = \text{Upper}$

$else$

$S_j.Zone = \text{Middle}$

$esle\ if\ (S_j(y_{max}) \leq M_{max}\ \text{AND}\ S_j(y_{min}) \geq L_{min})$

$if\ \left((S_j(y_{max}) - M_{min}) > (M_{min} - S_j(y_{min}))\right)$

$S_j.Zone = \text{Middle}$

$else$

$$S_j.Zone = \text{Lower}$$

$$esle\ if\ \left(S_j(y_{max}) \geq \text{U}_{min}\ \text{AND}\ S_j(y_{min}) \leq \text{L}_{max}\right)$$

$$S_j.Zone = \text{Middle}$$

endfor

Step 6: *Exit*

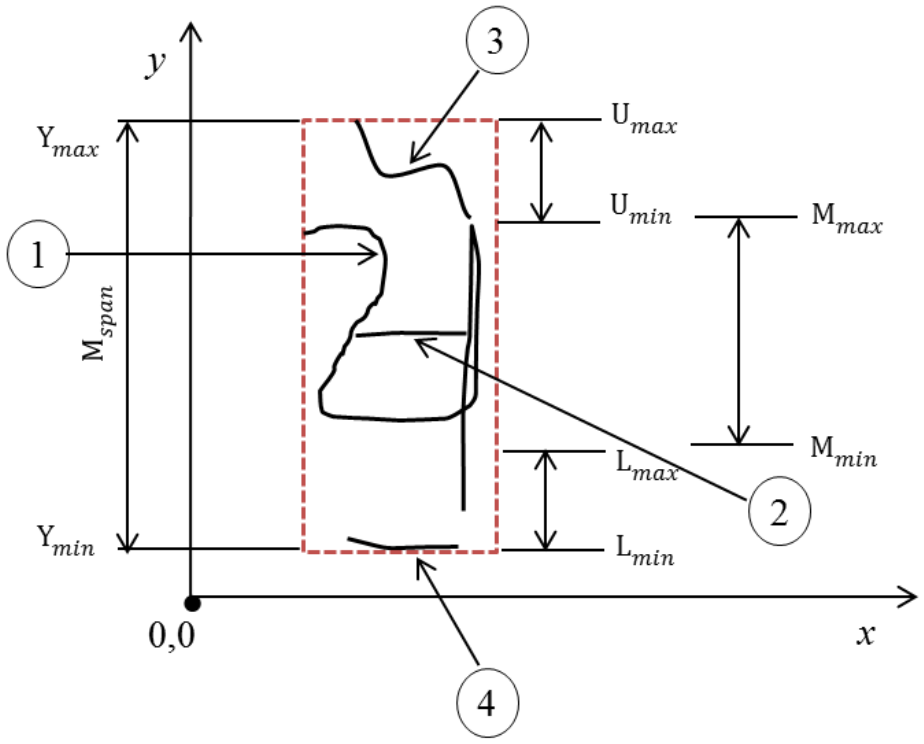
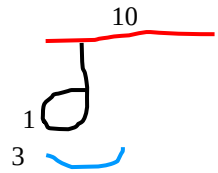
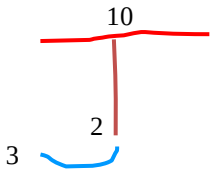
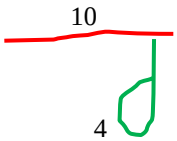
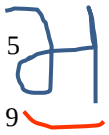
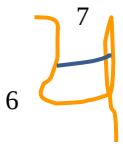
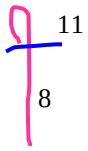


Figure 4.14: Zone identification of handwritten *Gurmukhi* strokes

Algorithm 4.10 has been used to identify the zones of strokes used for the formation of *Gurmukhi* word. The association sets that formulated the character in the previous section have been taken as input in the proposed algorithm. The algorithm scaled the association individually on the *y*-axis and classified the upper, middle and lower zone. The final results traced after the implementation of the algorithm over the considered *Gurmukhi* word is shown in Table 4.3. Each of the association set is individually analyzed and marked to obtain the final zone based division of each stroke. Association set A_1, contains *stroke sequence number* $'1'$ with middle zone, *stroke sequence number* $'3'$ in lower zone and *stroke sequence number* $'10'$ in upper zone.

Table 4.3: Final associated sets corresponding with their zone identification

$A_1 = \{1\ (M), 3\ (L), 10\ (U)\}$	$A_2 = \{2\ (M), 3(L), 10(U)\}$	$A_4 = \{4\ (M), 10\ (U)\}$
		
$A_5 = \{5\ (M), 9\ (L)\}$	$A_6 = \{6\ (M), 7\ (M)\}$	$A_8 = \{8\ (M), 11\ (U)\}$
		

4.4.1 Analysis of zone identification technique

The proposed zone identification technique has been tested on sample of *Gurmukhi* words written by 10 users. Total 2,960 samples have been collected for the purpose of this testing. It has been observed here that strokes written in upper zone are mainly confused with middle zone. Similarly, strokes written in lower zone get confused with middle zone. Same way a stroke written in middle zone if extended to upper or lower zone may create confusion with regard to its inclusion in a particular zone. Table 4.4 has provided accuracy of zone identification for input of *Gurmukhi* stroke by a writer. Average accuracy achieved for zone identification of stroke in upper zone, middle zone and lower zone is 95.5%, 98.9% and 93.8%. Overall accuracy for zone identification has been achieved as 96.1%.

Table 4.4: Writer wise zone identification accuracy (in %)

Users	**Upper zone**	**Confusion with**	**Middle zone**	**Confusion With**		**Lower zone**	**Confusion with**	**Overall zone identification**
		Middle		**Upper**	**Lower**		**Middle**	
User 1	93.0	7.0	99.6	0.3	0.1	96.0	4.0	**96.2**
User 2	98.5	1.5	100.0	0.0	0.0	99.1	0.9	**99.2**
User 3	94.8	5.2	100.0	0.0	0.0	95.3	4.7	**96.7**
User 4	96.5	3.5	98.4	1.6	0.0	96.3	3.7	**97.1**
User 5	98.5	1.5	98.2	1.1	0.7	82.5	17.5	**93.1**

User 6	92.5	7.5	97.8	1.3	0.9	98.4	1.6	**96.2**
User 7	96.7	3.3	98.1	1.2	0.7	86.4	13.6	**93.7**
User 8	96.1	3.9	99.7	0.3	0.0	95.3	4.7	**97.0**
User 9	92.1	7.9	98.5	0.9	0.6	93.5	6.5	**94.7**
User 10	96.5	3.5	98.8	0.7	0.5	95.1	4.9	**96.8**
Average	**95.5**	4.5	**98.9**	0.7	0.4	**93.8**	6.2	**96.1**

4.5 Feature extraction from online handwritten data

Pre-processing steps presented the input data into an acceptable form for feature extraction. Hence, points identified in pre-processing phase provided the base for feature extraction. Many existing studies on handwriting recognition have supported that identification of appropriate features is an important step in the handwriting recognition process (Suen *et al.*, 1980; Impedovo *et al.*, 1991; Gader and Khabou, 1996; Jinhai and Liu, 1999). Many piece of research work on feature extraction has concluded that high accuracy results of recognition can be achieved with appropriate features selected (Tier *et al.*, 1996). A recent study (Singh *et al.*, 2011) provided that on feature extraction for *Gurmukhi* script concluded that high accuracy result of 99.2% have been achieved for numeral recognition using projection histogram and 99.13% accuracy has been obtained using zonal density features. Hence, it was imperative for this research study to extract relevant features from input data that will result in reducing computational complexities. Extracting unique features requires very careful examination because features from one script to another. There is no standard method to extract features from a script (Jaeger *et al.*, 2001) and in addition, features identified as efficient for one script may not be efficient for another script.

Researchers have proposed wide range of different features for character recognition process that include geometrical features, statistical features, topological features, global features (Okamoto, 1999; Jose *et al.*, 2002; Dongre *et al.*, 2010). Broadly, features are classified as low level and high level features. Low level features focus on estimation of neighboring points whereas high level features focus on large scale and provide information regarding loop, headline, dots etc. In the present research study, we have worked on feature extraction and proposed three important features that include positional features, pre-processed features and features using reverse structuring. Detail explanation

of these extracted features has been discussed in the forthcoming sub sections. Mathematical formulation of feature vector is given below:

$$P_p = \{p_{p_1}, p_{p_2}, \dots, p_{p_m}\}$$

$$F_i = \begin{cases} P_p = \{(x_{p_1}, y_{p_1}), (x_{p_2}, y_{p_2}), \dots, (x_{p_{m-1}}, y_{p_{m-1}}), (x_{p_m}, y_{p_m})\} \\ \bar{P}_p = \{(x_{p_m}, y_{p_m}), (x_{p_{m-1}}, y_{p_{m-1}}), \dots, (x_{p_2}, y_{p_2}), (x_{p_1}, y_{p_1})\} \end{cases}$$

Here, F_i is a feature and i is refers to as Class ID.

P_p denotes resampled 64 points of x, y coordinates and feature vector size 128.

$\bar{P}_p$ is reverse structure of resampled 64 points of x, y coordinates and feature vector size 128.

An example of stored featured vector is mentioned below:

$$F_{Class\_ID} = \begin{Bmatrix} Class\_ID\ 154{:}\, x_1\ 2{:}\, y_1\ 3{:}\, x_2\ 4{:}\, y_2\ \dots\ 127{:}\, x_{64}\ 128{:}\, y_{64} \\ Class\_ID\ 154{:}\, x_{64}\ 2{:}\, y_{64}\ 3{:}\, x_{63}\ 4{:}\, y_{63}\ \dots\ 127{:}\, x_1\ 128{:}\, y_1 \end{Bmatrix}$$

Here, feature vector of size 128 has been stored with LibSVM class Id $'154'$, 128 (x, y) coordinates points as feature of a stroke.

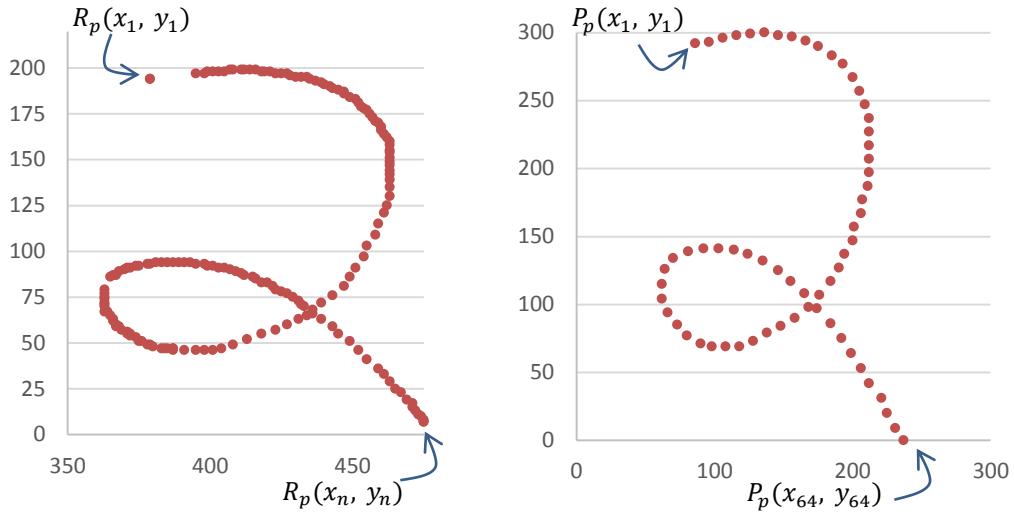
4.5.1 Extraction of pre-processed features

The important aspect of handwriting recognition system is the selection of good feature set, which is reasonably invariant with respect to shape variations caused by various writing styles. This section describes some of the features extracted from the raw data which have been used for classification. Points generated after pre-processing phase are used as features for recognition. In the present work, P_p resampled 64 (x, y) points for each stroke are fixed and are equidistant as far as possible. The extracted feature, *i.e.*, x and y coordinates of each stroke were stored in LibSVM format as $(x_1, y_1), (x_2, y_2), \dots\dots, (x_{63}, y_{63}), (x_{64}, y_{64})$ for recognition and thus obtaining the desired class to which a stroke belongs.

$$P_p = \{(x_1, y_1), (x_2, y_2), \dots\dots, (x_{63}, y_{63}), (x_{64}, y_{64})\}$$

Figure 4.15 provides a brief description of raw points R_p and pre-processed points P_p where plotted 64 (x, y) points ranging from $(x_1, y_1), (x_2, y_2), \dots\dots, (x_{63}, y_{63}), (x_{64}, y_{64})$ have been shown that will be stored in feature vector.

Figure 4.15: Illustration of (a) Raw points (b) preprocessed 64 points



4.5.2 Extraction of features using reverse structure

Here, we have targeted the problem of recognizing online handwritten character where a user may write that character in reverse order than the usual order of writing it. Hence, we used the same method of identifying 64 points (x, y) coordinates written by user in reverse order. For this purpose the extracted feature, *i.e.*, x and y coordinates of each stroke have been stored in LibSVM format as x_{64}, y_{64}, x_{63}, y_{63}, …, x_1, y_1 for recognition and obtained the desired class to which a stroke belongs.

$$Reverse\ Structure\ \bar{P}_p = \{(x_{64}, y_{64}), (x_{63}, y_{63}), \ldots\ldots, (x_2, y_2), (x_1, y_1)\}$$

Figure 4.16 provides a brief description of raw points R_p and pre-processed points $\bar{P}_p$ where plotted 64 (x, y) points ranging from $\{(x_{64}, y_{64}), (x_{63}, y_{63}), \ldots\ldots, (x_2, y_2), (x_1, y_1)\}$ have been shown that will be stored in feature vector.

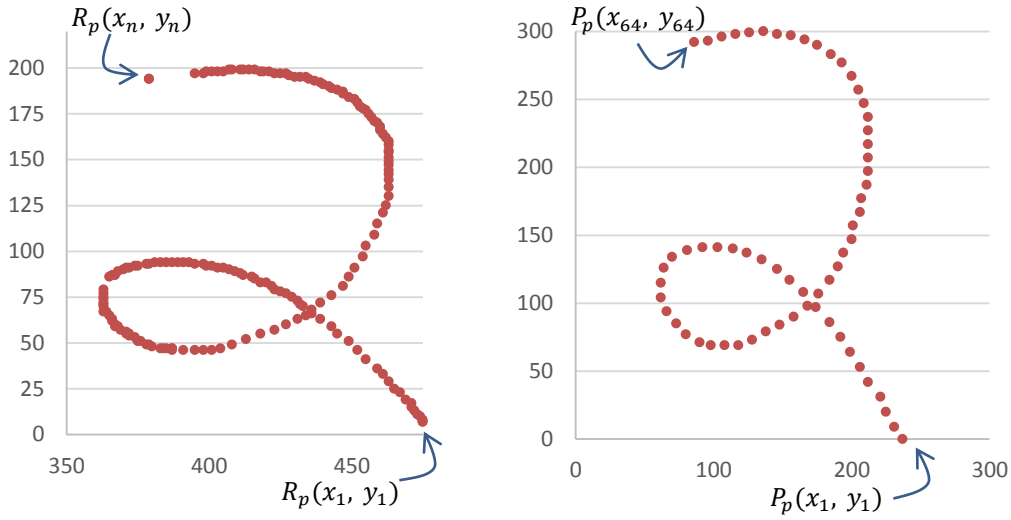


Figure 4.16: Illustration of (a) Raw points (b) Pre-processed 64 (*x*, *y*) points for reverse structuring

4.6 Chapter summary

In this chapter, proposed pre-processing, feature extraction and zone identification technique adopted for recognition of online handwritten *Gurmukhi* script have been discussed. Steps involved in pre-processing have also been explained. Pre-processing is used for removing noisy artifacts from handwriting and it also helps in correcting imperfections. Pre-processing involves preliminary analysis of stroke, association of strokes, size normalization and centering, interpolating missing points, and resampling. Detail working of proposed preprocessing steps introduced in this study has been explained. This chapter also provides proposed zone identification technique adopted for recognition of online handwritten *Gurmukhi* script. Last section of this chapter provides broad features extracted for the purpose of this study. Final recognition and post processing phase of this process has been explained in the next chapter.

CHAPTER 5

RECOGNITION AND PROPOSED POST-PROCESSING ALGORITHMS

Pre-processing steps, as discussed in previous chapters, help in removing noisy artifacts from handwriting and also in getting a uniform representation of input data. Previous chapter has also provided an insight on important features extracted and used in the present research work. In this chapter, proposed post-processing methodology has been provided, in detail. Experimentation for achieving high recognition accuracy has been discussed in Section 5.1. Section 5.2 describes the rule book formulated for post-processing. Methodology of post-segmentation of stroke sets, sequencing of stroke sets, and merging the stroke sets for the purpose of post-processing have been described in Section 5.3. This chapter also covers some of the ambiguities faced in recognition of online handwritten *Gurmukhi* script and their solutions in Section 5.4. Section 5.5 demonstrates the process for formation of *Gurmukhi* character, akshara and word. Finally, Section 5.6 provides the recognition accuracy achieved for *Gurmukhi* character/numeral, akshara and word formation.

SVM has emerged as a promising data classification technique that is applied for solving various data mining problems. SVM works well for a given set of training data, where each set can be marked as belonging to separate category. SVM classifier endeavors to provide a decision model for predicting the inclusion of new variable in one of the existing categories. The difficulties in constructing SVM model are selecting the kernel function and its parameters. For current research work, LibSVM tool with Radial Basis Function (RBF) kernel has been used as it maps every input to infinite dimensional space. Selecting the parameter values for SVM is also very crucial because if they are not set properly then the classification outcome will not be optimal. Table 5.1 given below provides a list of parameters and their default values used in RBF kernel.

Table 5.1: Parameters and their default values for RBF kernel in LibSVM

S. No.	Parameter	Default value
1.	Gamma (γ)	1 / number of features
2.	Coefficient (r)	0
3.	Degree (d)	3
4.	Epsilon (ε)	0.001
5.	Penalty factor (C)	1

Although number of learning parameters can be used for constructing SVM yet the most crucial among them is to decide about the proper value of penalty factor and learning rate. Hence, in this research study various combinations between complexity of decision rule and frequency of error has been controlled by changing the value of parameter *C* and γ. Objective function of SVM cannot allow extreme values of *C*. If the value of *C* is too large, the optimization will select a smaller margin hyper plane. Similarly, a smaller value of γ gives low bias and high variance whereas the large value of gamma gives higher bias and low variance. In practice, to obtain the optimal results, the parameter *C* is varied through a wide range of values by making use of a separate validation set known as cross-validation for verifying performance using only a training set (Taylor and Cristianini, 2004; Verma and Sharma, 2014).

5.1 Experimentation for achieving higher recognition accuracy

Gurmukhi characters are categorized into three writing zones *viz.* Upper zone, Middle zone and Lower zone (as discussed in Section 1.3). Hence, to resolve this problem, Algorithm 4.10 has been used for zone identification. In this research work, three RBF engines have been designed *viz.* Engine A (Upper zone), Engine B (Middle zone) and Engine C (Lower zone). Once the zone of a stroke is identified they are tested on their respective engine.

In the present study, cross-validation has been done at 2-fold, 3-fold, 4-fold and 5-fold for three separate engines *viz.* Engine A, Engine B, and Engine C. Lastly, a separate Engine D has also been used to test the samples from combined (Upper + Middle + Lower) zones. A total of 826 samples were collected for seven classes of Upper zone. In Engine A, the sample of 826 *Gurmukhi* strokes has been used to train the engine and tested at 2-fold, 3-fold, 4-fold and 5-fold cross-validation. Comparative results of accuracy for all these levels have been obtained and provided in Table 5.2. Highest accuracy of 98.7% has

been achieved at 5-fold cross-validation. As most of the *Gurmukhi* characters are included in Middle zone. For Engine B, a sample of 10,740 *Gurmukhi* strokes has been used to train the engine and tested again at 2-fold, 3-fold, 4-fold and 5-fold cross-validation. Total numbers of 86 classes have been considered for Middle zone. Among these results, the highest accuracy of 96.7% has been achieved at 5-fold level. Three classes of Lower zone of *Gurmukhi* strokes have been considered and a total of 317 collected samples have been used to train the Engine C. In this case, highest accuracy of 100.0% has been achieved at 2-fold, 3-fold, 4-fold and 5-fold level. Finally, a separate testing of 11,883 samples of *Gurmukhi* script has been done with Engine D, considering all the three zones in aggregation. Highest accuracy of 95.3% has been achieved at 5-fold cross validation in this experiment.

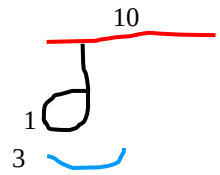
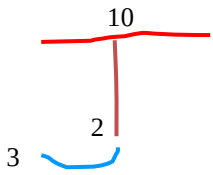
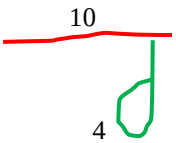
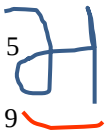
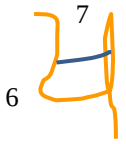
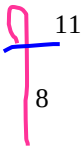
Table 5.2: Performance of LibSVM for different engines

	Accuracy			
	2-fold	**3-fold**	**4-fold**	**5-fold**
Engine A: **Upper Zone** *Number of Samples:* 826 *Number of Classes:* 7	98.2%	98.1%	98.2%	**98.7%**
C (*penalty factor*)	32	32	32	16
γ (*learning rate*)	0.001953	0.001953	0.001953	0.007812
Engine B: **Middle Zone** *Number of Samples:* 10,740 *Number of Classes:* 86	96.2%	96.5%	**96.7%**	**96.7%**
C (*penalty factor*)	8	8	32	128
γ (*learning rate*)	0.001953	0.001953	0.001953	0.001953
Engine C: **Lower Zone** *Number of Samples:* 317 *Number of Classes:* 3	**100.0%**	**100.0%**	**100.0%**	**100.0%**
C (*penalty factor*)	2	2	2	2
γ (*learning rate*)	0.007812	0.007812	0.007812	0.007812
Average	**98.1%**	**98.2%**	**98.5%**	**98.3%**
Engine D: **Upper + Middle + Lower Zone** *Number of Samples:* 11,883 *Number of Classes:* 96	94.6%	94.9%	**95.3%**	**95.3%**
C (*penalty factor*)	32	8	32	128
γ (*learning rate*)	0.001953	0.001953	0.001953	0.007812

After cross-validation, accuracy achieved at different folds was compared and highest accuracy achieved at fixed parameters was recorded. As mentioned earlier in Section 4.4, a *Gurmukhi* word /ਗੁਰਮੁਖੀ/ was considered for the purpose of zone identification. Based on the zone identification and feature extraction (as discussed in Section 4.5), it is decided

that a stroke will pass to which particular LibSVM engine. As shown in Table 5.3 below, each stroke in association set will include one more parameter *i.e.*, class ID, obtained from the trained LibSVM engine.

Table 5.3: Final association sets with zone marking and class ID

$A_1 = \{1\ (M)\ (164), 3\ (L)(101), 10\ (U)(121)\}$	$A_2 = \{2\ (M)\ (162), 3(L)\ (101), 10(U)(121)\}$
	
$A_4 = \{4\ (M)(164), 10\ (U)\ (121)\}$	$A_5 = \{5\ (M)\ (152), 9\ (L)\ (101)\}$
	
$A_6 = \{6\ (M)(159), 7\ (M)\ (163)\}$	$A_8 = \{8\ (M)\ (216), 11\ (U)\ (121)\}$
	

5.2 Formulation of rule book for post-processing

An important contribution of the present research work is in the form of framing a rule book for post-processing. This rule book has been provided for the formation of character. In this work, smallest unit of writing a *Gurmukhi* character is a stroke. Combination of strokes forms a character and further collection of characters with or without vowel modifiers form a word. Notation and definition for forming rule book are provided below.

Notations: For formation of *Gurmukhi* text, three sets, *i.e.*, character (C), vowels (V), nasals (N), and numerals (NU) are used, as described below.

C = {ਉ, ਅ, ਏ, ਸ, ਹ, ਕ, ਖ, ਗ, ਘ, ਙ, ਚ, ਛ, ਜ, ਝ, ਞ, ਟ, ਠ, ਡ, ਢ, ਣ, ਤ, ਥ, ਦ, ਧ, ਨ, ਪ, ਫ, ਬ, ਭ, ਮ, ਯ, ਰ, ਲ, ਵ, ੜ}

V = {Mukta, ਾ, ਿ, ੀ, ੁ, ੂ, ੇ, ੈ, ੋ, ੌ, ਂ}

N = {ੱ, ੰ, ਁ}

NU = {ੀ, ੁ, ੂ, ੇ, ੈ, ੋ, ੌ, ਂ, ਃ, ੰ}

Definition 5.1: [*Gurmukhi* Script Strokes Tuple (*T*)]: In a tuple *T*, we have a *Unicode* and *d*. Here *Unicode* is an element from C, V, N and NU; and *d* is the number of possible strokes that form C, V, N and NU. LibSVM gives the class ID and *w* is the weight assigned to the stroke in that tuple. This is written as:

$$T_{[\text{Unicode}]d} = \{LibSVM\_Id_{(w)}\}$$

Table 5.4 given below provides rule book with possible combinations of class ID that will lead to a formation of character/nasal/vowel modifiers or numeral. In this rule book, weight (w) is an important consideration because if any stroke does not have a proper weight for the formation of character then it is considered as *noise*. Assigning a weight to a particular stroke is decided depending upon its requirement in the formation of a character. However, that particular stroke used in different Tuple may have different weight because that stroke may play a different role in different character formation. In any Tuple, for the formation of a character, it has been hypothesized that minimum one and maximum four strokes are required.

In the present approach, real time post-processing is done, where with an input of every new stroke post-processor will recognize the character formed instead of recognizing the character at the end of all input strokes. To implement this, each stroke used for formation of a character has been assigned three weights w_1, w_2, and w_3. These weights are computed taking into consideration number of strokes input in an association set, the number of strokes matched in a tuple and total number of strokes (d) in an activated tuple. LibSVM gives output in the form of stroke number for every stroke written by a writer but does not know the character that will ultimately be formed. However, on the basis of the strokes, proposed post-processor algorithm gives an indication about particular set *i.e.*, character/vowel/nasal.

In this work, weights w_1, w_2 and w_3 are computed as under

$$w_1 = \frac{\text{Number of strokes matched in a tuple } (m)}{\text{Total number of strokes in an activated tuple } (d)}$$

$$w_2 = \frac{\text{Number of strokes matched in a tuple } (m)}{\text{Number of strokes input in an association set } (a)}$$

w_3 = is sum of weights of matched strokes in a tuple

Now, the tuples having w_3 > *threshold* (0.8) are the possible tuples for character, vowel or nasal formation. Out of the activated tuples (having w_3 > 0.8), the tuple with maximum ($w_1 + w_2$) is selected for character/vowel/nasal formation.

Table 5.4: *Gurmukhi* character/nasal/vowel/numeral with their possible combination of class Id(s)

S. No.	Unicode	Character shape (name)		*Gurmukhi* Script Strokes Tuple (*T*) $T_{[Unicode]d}$ = {*LibSVM Id* $_{(w)}$}
1	U+0A73	ੳ	/*ooṛá*/	$T_{[U+0A73]1}$={ **141** (1.0) }
				$T_{[U+0A73]2}$={ **141** (0.8), **121** (0.2)}
				$T_{[U+0A73]2}$={ **141** (0.8), **163** (0.2)}
				$T_{[U+0A73]1}$={ **142** (1.0)}
				$T_{[U+0A73]2}$={ **142** (0.8), **121** (0.2)}
				$T_{[U+0A73]2}$={ **142** (0.8), **163** (0.2)}
2	U+0A05	ਅ	/*āiṛá*/	$T_{[U+0A05]1}$={ **144** (0.8)}
				$T_{[U+0A05]2}$={ **145** (0.6), **162** (0.4)}
3	U+0A72	ੲ	/*íṛí*/	$T_{[U+0A72]2}$={**146** (0.5), **147** (0.5)}
				$T_{[U+0A72]3}$={**146** (0.4), **147** (0.4), **121** (0.2)}
				$T_{[U+0A72]2}$={**146** (0.5), **148** (0.5)}
				$T_{[U+0A72]3}$={**146** (0.4), **148** (0.4), **121** (0.2)}
				$T_{[U+0A72]2}$={**146** (0.5), **149** (0.5)}
				$T_{[U+0A72]3}$={**146** (0.4), **149** (0.4), **121** (0.2)}
				$T_{[U+0A72]2}$={**148** (0.5), **149** (0.5)}
				$T_{[U+0A72]3}$={**148** (0.4), **149** (0.4), **121** (0.2) }
				$T_{[U+0A72]1}$={**150** (1.0) }
				$T_{[U+0A72]2}$={**150** (0.8), **121** (0.2)}
				$T_{[U+0A72]1}$={**177** (1.0) }
				$T_{[U+0A72]2}$={**177** (0.8), **121** (0.2)}
				$T_{[U+0A72]2}$={**182** (0.5), **147** (0.5)}
				$T_{[U+0A72]3}$={**182** (0.4), **147** (0.4), **121** (0.2) }
				$T_{[U+0A72]2}$={**182** (0.5), **148** (0.5)}
				$T_{[U+0A72]3}$={**182** (0.4), **148** (0.4), **121** (0.2)}

				$T_{[U+0A72]2}$={**182** (0.5), **149** (0.5)}
				$T_{[U+0A72]3}$={**182** (0.4), **149** (0.4), **121** (0.2)}
				$T_{[U+0A72]2}$={**225** (0.5), **147** (0.5)}
				$T_{[U+0A72]3}$={**225** (0.4), **147** (0.4), **121** (0.2)}
				$T_{[U+0A72]2}$={**225** (0.5), **148** (0.5)}
				$T_{[U+0A72]3}$={**225** (0.4), **148** (0.4), **121** (0.2)}
				$T_{[U+0A72]2}$={**225** (0.5), **149** (0.5)}
				$T_{[U+0A72]3}$={**225** (0.4), **149** (0.4), **121** (0.2)}
4	U+0A38	ਸ	/*sassá*/	$T_{[U+0A38]2}$={**151** (0.5), **154** (0.5) }
				$T_{[U+0A38]3}$={**151** (0.4), **162** (0.3) , **121** (0.3) }
				$T_{[U+0A38]2}$={**152** (0.6), **121** (0.4) }
				$T_{[U+0A38]1}$={**153** (1.0) }
5	U+0A39	ਹ	/*háhá*/	$T_{[U+0A39]1}$={**155** (1.0) }
				$T_{[U+0A39]2}$={**155** (0.8), **121** (0.2) }
				$T_{[U+0A39]1}$={**156** (1.0) }
6	U+0A15	ਕ	/*kakká*/	$T_{[U+0A15]1}$={**157** (1.0) }
				$T_{[U+0A15]2}$={**157** (0.8), **121** (0.2) }
				$T_{[U+0A15]1}$={**158** (1.0) }
7	U+0A16	ਖ	/*khakkhá*/	$T_{[U+0A16]2}$={**159** (0.6), **163** (0.4) }
				$T_{[U+0A16]1}$={**160** (1.0) }
				$T_{[U+0A16]3}$={**161** (0.4), **162** (0.3) , **163** (0.3) }
8	U+0A18	ਘ	/*kaggá*/	$T_{[U+0A18]1}$={**166** (1.0) }
				$T_{[U+0A18]2}$={**167** (0.7), **162** (0.3) }
9	U+0A19	ਙ	/*ṅaṅá*/	$T_{[U+0A19]1}$={**168** (1.0) }
				$T_{[U+0A19]2}$={**168** (0.8), **121** (0.2) }
				$T_{[U+0A19]1}$={**169** (1.0) }
10	U+0A1A	ਚ	/*chachchá*/	$T_{[U+0A1A]1}$={**170** (1.0) }
				$T_{[U+0A1A]2}$={**170** (0.8), **121** (0.2) }
				$T_{[U+0A1A]1}$={**171** (1.0) }
				$T_{[U+0A1A]2}$={**226** (0.5), **162** (0.5) }
				$T_{[U+0A1A]3}$={**226** (0.4), **162** (0.4) , **121** (0.2) }
				$T_{[U+0A1A]2}$={**226** (0.5), **154** (0.5) }

11	U+0A1B	ਛ	/*chhachhá*/	$T_{[U+0A1B]1}$={**172** (1.0) } $T_{[U+0A1B]2}$={**172** (0.8), **121** (0.2) } $T_{[U+0A1B]1}$={**173** (1.0) }
12	U+0A1C	ਜ	/*jajjá*/	$T_{[U+0A1C]2}$={**174** (0.5), **154** (0.5) } $T_{[U+0A1C]2}$={**174** (0.5), **162** (0.5) } $T_{[U+0A1C]3}$={**174** (0.4), **162** (0.4) , **121** (0.2) } $T_{[U+0A1C]1}$={**175** (1.0) } $T_{[U+0A1C]2}$={**222** (0.5), **162** (0.5) } $T_{[U+0A1C]3}$={**222** (0.4), **162** (0.4) , **121** (0.2) } $T_{[U+0A1C]2}$={**223** (0.5), **162** (0.5) } $T_{[U+0A1C]3}$={**223** (0.4), **162** (0.4) , **121** (0.2) }
13	U+0A1D	ਝ	/*cajhá*/	$T_{[U+0A1D]2}$={**148** (0.5), **157** (0.5) } $T_{[U+0A1D]3}$={**148** (0.4), **157** (0.4) , **121** (0.2) } $T_{[U+0A1D]2}$={**148** (0.5), **158** (0.5) } $T_{[U+0A1D]3}$={**148** (0.4), **158** (0.4) , **121** (0.2) } $T_{[U+0A1D]3}$={**176** (1.0)} $T_{[U+0A1D]2}$={**176** (0.8), **121** (0.2) } $T_{[U+0A1D]2}$={**177** (0.5), **157** (0.5) } $T_{[U+0A1D]3}$={**177** (0.4), **157** (0.4) , **121** (0.2) } $T_{[U+0A1D]2}$={**177** (0.5), **158** (0.5) } $T_{[U+0A1D]3}$={**177** (0.4), **158** (0.4) , **121** (0.2) }
14	U+0A1E	ਞ	/*náñá*/	$T_{[U+0A1E]2}$={**147** (0.5), **179** (0.5)} $T_{[U+0A1E]3}$={**147** (0.4), **179** (0.4), **121** (0.2) } $T_{[U+0A1E]2}$={**148** (0.5), **179** (0.5) } $T_{[U+0A1E]3}$={**148** (0.4), **179** (0.4) , **121** (0.2) } $T_{[U+0A1E]1}$={**180** (1.0) } $T_{[U+0A1E]2}$={**180** (0.8), **121** (0.2) } $T_{[U+0A1E]2}$={**181** (0.5), **147** (0.5) } $T_{[U+0A1E]3}$={**181** (0.4), **147** (0.4) , **121** (0.2) }
15	U+0A1F	ਟ	/*ṭāīnká*/	$T_{[U+0A1F]1}$={**146** (1.0) } $T_{[U+0A1F]2}$={**146** (0.8), **121** (0.2) } $T_{[U+0A1F]1}$={**182** (1.0) }

16	U+0A20	ਠ	/*ṭhaṭhṭhá*/	$T_{[U+0A20]1}$={**183** (1.0) } $T_{[U+0A20]1}$={**184** (1.0) } $T_{[U+0A20]2}$={**184** (0.8), **121** (0.2) }
17	U+0A21	ਡ	/*ḍaḍḍá*/	$T_{[U+0A21]1}$={**185** (1.0) } $T_{[U+0A21]2}$={**186** (0.8), **121** (0.2) }
18	U+0A22	ਢ	/*taḍhá*/	$T_{[U+0A22]1}$={**187** (1.0) } $T_{[U+0A22]2}$={**187** (0.8), **121** (0.2) } $T_{[U+0A22]1}$={**188** (1.0) }
19	U+0A23	ਣ	/*ṇáṇá*/	$T_{[U+0A23]1}$={**190** (1.0) } $T_{[U+0A23]2}$={**190** (0.8), **121** (0.2) }
20	U+0A24	ਤ	/*tattá*/	$T_{[U+0A24]1}$={**191** (1.0) } $T_{[U+0A24]1}$={**192** (1.0) } $T_{[U+0A24]2}$={**192** (0.7), **121** (0.3) }
21	U+0A25	ਥ	/*thaththá*/	$T_{[U+0A25]3}$={**159** (0.4), **163** (0.3) , **121** (0.3) } $T_{[U+0A25]2}$={**160** (0.6), **121** (0.4) } $T_{[U+0A25]4}$={**161** (0.2), **162** (0.2) , **163** (0.2), **121** (0.3)} $T_{[U+0A25]3}$={**161** (0.4), **154** (0.3) , **163** (0.3) } $T_{[U+0A25]2}$={**163** (0.4) , **195** (0.6)}
22	U+0A26	ਦ	/*daddá*/	$T_{[U+0A26]1}$={**193** (1.0) } $T_{[U+0A26]2}$={**193** (0.8), **121** (0.2) } $T_{[U+0A26]1}$={**194** (1.0) }
23	U+0A27	ਧ	/*tadhá*/	$T_{[U+0A27]2}$={**159** (0.6), **121** (0.4) } $T_{[U+0A27]2}$={**161** (0.5), **154** (0.5) } $T_{[U+0A27]3}$={**161** (0.4) , **162** (0.3), **121** (0.3) } $T_{[U+0A27]1}$={**195** (1.0) }
24	U+0A28	ਨ	/*nanná*/	$T_{[U+0A28]2}$={**146** (0.5), **197** (0.5) } $T_{[U+0A28]3}$={**146** (0.4), **197** (0.4) , **121** (0.2) } $T_{[U+0A28]2}$={**182** (0.6), **197** (0.4) } $T_{[U+0A28]3}$={**182** (0.4), **197** (0.4) , **121** (0.2) } $T_{[U+0A28]1}$={**196** (1.0) } $T_{[U+0A28]2}$={**196** (0.8), **121** (0.2) } $T_{[U+0A28]2}$={**225** (0.5), **197** (0.5) }

				$T_{[U+0A28]3}$={**225** (0.4), **197** (0.4) , **121** (0.2) }
25	U+0A2A	ਪ	/*pappá*/	$T_{[U+0A2A]1}$={**159** (1.0) }
				$T_{[U+0A2A]2}$={**161** (0.5), **162** (0.5) }
26	U+0A2B	ਫ	/*phaphphá*/	$T_{[U+0A2B]2}$={**200** (0.8), **121** (0.2) }
				$T_{[U+0A2B]1}$={**201** (1.0) }
27	U+0A2C	ਬ	/*babbá*/	$T_{[U+0A2C]2}$={**202** (0.6), **162** (0.4) }
				$T_{[U+0A2C]3}$={**202** (0.6), **162** (0.2) , **121** (0.2) }
				$T_{[U+0A2C]2}$={**202** (0.6), **154** (0.4) }
				$T_{[U+0A2C]1}$={**203** (1.0) }
				$T_{[U+0A2C]2}$={**203** (0.8), **121** (0.2) }
28	U+0A2D	ਭ	/*pabhá*/	$T_{[U+0A2D]2}$={**204** (0.8), **121** (0.2) }
				$T_{[U+0A2D]1}$={**205** (1.0) }
29	U+0A2E	ਮ	/*mammá*/	$T_{[U+0A2E]2}$={**151** (0.5), **162** (0.5)}
				$T_{[U+0A2E]1}$={**152** (1.0)}
30	U+0A2F	ਯ	/*yayyá*/	$T_{[U+0A2F]2}$={**207** (0.5), **154** (0.5) }
				$T_{[U+0A2F]3}$={**207** (0.4), **162** (0.4) , **121** (0.2) }
				$T_{[U+0A2F]1}$={**208** (1.0) }
				$T_{[U+0A2F]2}$={**209** (0.8), **121** (0.2) }
31	U+0A30	ਰ	/*rárá*/	$T_{[U+0A30]1}$={**164** (1.0) }
				$T_{[U+0A30]2}$={**164** (0.8), **121** (0.2) }
				$T_{[U+0A30]1}$={**165** (1.0) }
32	U+0A32	ਲ	/*lallá*/	$T_{[U+0A32]3}$={**148** (0.3), **149** (0.3) , **197** (0.4) }
				$T_{[U+0A32]4}$={**148** (0.3), **149** (0.3) , **197** (0.3), **121** (0.1)}
				$T_{[U+0A32]2}$={**148** (0.5), **198** (0.5) }
				$T_{[U+0A32]3}$={**148** (0.4), **198** (0.4) , **121** (0.2) }
				$T_{[U+0A32]2}$={**150** (0.6), **197** (0.4) }
				$T_{[U+0A32]3}$={**150** (0.6), **197** (0.2) , **121** (0.2) }
				$T_{[U+0A32]2}$={**150** (0.6), **147** (0.4) }
				$T_{[U+0A32]3}$={**150** (0.6), **147** (0.2) , **121** (0.2) }
				$T_{[U+0A32]2}$={**177** (0.4), **197** (0.4)}
				$T_{[U+0A32]3}$={**177** (0.4), **197** (0.4) , **121** (0.2) }
				$T_{[U+0A32]2}$={**177** (0.6), **147** (0.4) }

				$T_{[U+0A32]3}$={**177** (0.6), **147** (0.2) , **121** (0.2) } $T_{[U+0A32]1}$={**210** (1.0) } $T_{[U+0A32]2}$={**210** (0.8), **121** (0.2) } $T_{[U+0A32]2}$={**224** (0.4), **225** (0.4)} $T_{[U+0A32]3}$={**224** (0.4), **225** (0.4) , **121** (0.2) }
33	U+0A35	ਵ	/*vává*/	$T_{[U+0A35]1}$={**179** (1.0) } $T_{[U+0A35]2}$={**179** (0.8), **121** (0.2)} $T_{[U+0A35]1}$={**181** (1.0)}
34	U+0A5C	ੜ	/*ṛáṛá*/	$T_{[U+0A5C]2}$={**212** (0.5), **147** (0.5) } $T_{[U+0A5C]3}$={**212** (0.4), **147** (0.4), **121** (0.2) }
35	U+0A3E	ਾ	/*kanná*/	$T_{[U+0A3E]1}$={**154** (1.0) } $T_{[U+0A3E]1}$={**162** (1.0) }
36	U+0A3F	ਿ	/*sihárí*/	$T_{[U+0A3F]1}$={**215** (1.0) }
37	U+0A40	ੀ	/*bihárí*/	$T_{[U+0A40]1}$={**216** (1.0) }
38	U+0A47	ੇ	/*lã̃*/	$T_{[U+0A47]1}$={**122** (1.0) }
39	U+0A48	ੈ	/*duláwã̃*/	$T_{[U+0A48]1}$={**123** (1.0) }
40	U+0A41	ੁ	/*āūnkaṛ*/	$T_{[U+0A41]1}$={**101** (1.0) }
41	U+0A42	ੂ	/*dulāinkṛe*/	$T_{[U+0A42]1}$={**106** (1.0) }
42	U+0A4B	ੋ	/*hoṛá*/	$T_{[U+0A4B]1}$={**124** (1.0) }
43	U+0A4C	ੌ	/*kanāūṛá*/	$T_{[U+0A42]1}$={**133** (1.0) }
44	U+0A70	ਂ	/*ṭipí*/	$T_{[U+0A70]1}$={**128** (1.0) }
45	U+0A71	ੱ	/*adhak*/	$T_{[U+0A71]1}$={**126** (1.0) }
46	U+0A66	੦	/*sifár*/	$T_{[U+0A66]1}$={**360** (1.0) }
47	U+0A67	੧	/*ikk*/	$T_{[U+0A67]1}$={**351** (1.0) }
48	U+0A68	੨	/*do*/	$T_{[U+0A68]1}$={**352** (1.0) }
49	U+0A69	੩	/*tinn*/	$T_{[U+0A69]1}$={**353** (1.0) }
50	U+0A6A	੪	/*chár*/	$T_{[U+0A6A]1}$={**354** (1.0) }
51	U+0A6B	੫	/*pánj*/	$T_{[U+0A6B]1}$={**355** (1.0) }

52	U+0A6C	੬	/*chhe*/	$T_{[U+0A6C]1}$={**356** (1.0) } $T_{[U+0A6C]2}$={**361** (0.5), **362** (0.5) }
53	U+0A6D	੭	/*sátt*/	$T_{[U+0A6D]1}$={**357** (1.0) }
54	U+0A6E	੮	/*áṭh*/	$T_{[U+0A6E]1}$={**358** (1.0) }
55	U+0A6F	੯	/*nãũ*/	$T_{[U+0A6F]1}$={**359** (1.0) } $T_{[U+0A6F]2}$={**358** (0.5), **362** (0.5) }

5.3 Post-processing

In post-processing phase, misclassified results are corrected with the help of orthography knowledge. At this stage of online handwriting recognition process, recognized strokes have been verified through identified features of a stroke in the *Gurmukhi* script. Post-processing is an important step in handwriting recognition process as it endeavors to bring corrections in the formation of a character that comes from the recognizer. Hence, after post-processing recognition accuracy increases considerably and more promising results are obtained. In this research work, post-processing phase has been implemented in three steps, *viz. post-segmentation of stroke sets*, *sequencing of stroke sets*, and *merging the stroke sets* which are discussed in detail in the forthcoming sub sections.

Figure 5.1 provides a broad overview of post processing phase implemented in this study. As explained in Section 5.1, association sets were obtained along with their information about *stroke sequence number*, class ID and identified zone. This information becomes input for post-segmentation, where the association sets are divided into two categories, *viz.* consonants based association sets (A^{C}) and vowel/nasal based association sets (A^{V}). In the step of sequencing of stroke sets, A^{C} and A^{V} are sorted based on X_{min} and X_{max} of strokes, respectively. Finally, in merging the stroke sets a rule book is referred for each association set and their weights (w_1, w_2 and w_3) are computed. With this, each association set provides a Unicode of *Gurmukhi* script and by combining these Unicodes, desired output is obtained.

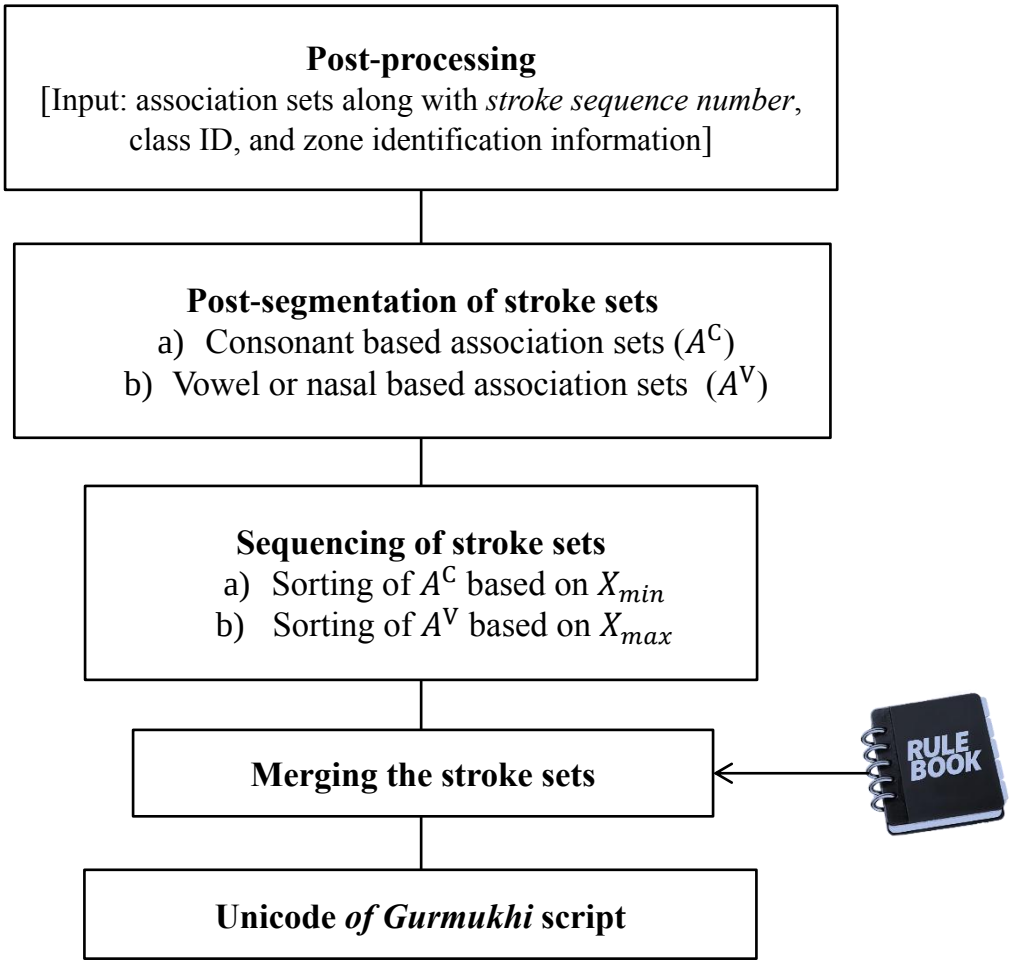


Figure 5.1: Post-processing phase

5.3.1 Post-segmentation of stroke sets

Post-segmentation of strokes has been proposed as an important step in the post-processing phase of online handwritten *Gurmukhi* script in this work. In this step, the set of associations are processed to determine the final shape of *Gurmukhi* character. To assist this process, set M, as illustrated below, has been constructed. The set contains class IDs used to form vowel modifiers and nasal included in *Gurmukhi* script. Each association set is checked for vowel modifier or nasal as mentioned in set M. If an association set contains any of the strokes in the set M, then that stroke is removed from the current association set. A new association set is created for the same, *i.e.*, vowel/nasal based association set (A^{V}). Remaining elements in association set will be known as a separate set, *i.e.*, consonant based association set (A^{C}).

M = { 101 ( ), 106 ( ), 122 ( ), 123 ( ), 124 ( ), 126 ( ),
128 ( ), 133 ( ), 154 ( ), 162 ( ), 215 ( ), 216 ( ) }

Algorithm 5.1 has been used to implement post-segmentation of stroke sets. Here, we input each association set A_i along with the information about their *stroke sequence number*, class ID, and zone identification. Further, based upon the proposed methodology this association set is divided into two or more subsets, *viz.* A^{C} and A^{V}.

Algorithm 5.1: Post-segmentation of stroke sets

Input: Association sets along with their *stroke sequence number*, class ID, and zone identification.

Output: Association Table 5.5, consonant based association set (A^{C}) and vowel/nasal based association set (A^{V}).

Step 1: Verify each association set (A_i), whether any class ID of association set belongs to set M.

Step 2: If *Step* 1 is true, separate that class ID along with *stroke sequence number*, and zone identification to newly formed set called vowel/nasal based association set (A^{V}) and remaining elements form a set, called consonant based association set (A^{C}).

This algorithm is now explained with the help of an example of *Gurmukhi* script word /ਗੁਰਮੁਖੀ/. As shown in Table 5.5, the transformation of association set as created in Section 5.1 have been converted into consonant based association set (A^{C}) and vowel/nasal based association set (A^{V}). Here, before post-segmentation of association sets, there were seven association sets, *i.e.*, $A_1, A_2, A_3, A_4, A_5, A_6,$ and A_8. After post-segmentation, these association sets have been classified into two separate association sets with their respective information.

Table 5.5: Association sets before and after post-segmentation

Association sets before post-segmentation	**A^{C} and A^{V} association sets after post-segmentation**
$A_1 = \{1\ (M)\ (164), 3\ (L)(101), 10\ (U)(121)\}$	$A_1^{C} = \{1\ (M)\ (164), 10\ (U)(121)\}$ $A_1^{V} = \{3\ (L)(101)\}$
$A_2 = \{2\ (M)\ (162), 3(L)\ (101), 10(U)(121)\}$	$A_2^{C} = \{\ 10(U)(121)\}$ $A_2^{V} = \{2\ (M)\ (162)\}$ $A_3^{V} = \{3\ (L)(101)\}$
$A_4 = \{4\ (M)(164), 10\ (U)\ (121)\}$	$A_3^{C} = \{4\ (M)(164), 10\ (U)\ (121)\}$
$A_5 = \{5\ (M)\ (152), 9\ (L)\ (101)\}$	$A_4^{C} = \{5\ (M)\ (152)\}$ $A_4^{V} = \{9\ (L)(101)\}$
$A_6 = \{6\ (M)(159), 7\ (M)\ (163)\}$	$A_5^{C} = \{6\ (M)(159), 7\ (M)\ (163)\}$
$A_8 = \{8\ (M)\ (216), 11\ (U)\ (121)\}$	$A_6^{C} = \{11\ (U)\ (121)\}$ $A_5^{V} = \{8\ (M)\ (216)\}$

5.3.2 Sequencing of stroke sets

Next important step introduced in the post-processing phase is sequencing of stroke sets. Consonant based association set (A^{C}) and vowel/nasal based association set (A^{V}) obtained as output sets in the Section 5.3.1 become input here. From these classified sets of A^{C} and A^{V}, sorted sets A^{S} are obtained based on X_{min} and X_{max} respectively. Table 5.6 presented below provides detail of sorted sets obtained as A_1^{S}, A_2^{S}, A_3^{S}, A_4^{S}, A_5^{S}, A_6^{S}, A_7^{S}, A_8^{S}, A_9^{S}, A_{10}^{S}, and A_{11}^{S}. Algorithm 5.2 has been proposed for sequencing of stroke sets. This algorithm provides that for sequencing of stroke sets two important steps are required. In the first step, X_{min} is computed from all the elements in A_i^{C} and X_{max} is computed from all the elements in A_j^{V} (except for class ID $'121'$ is called *head line*). Here, $i = 1, 2, .., 6$ and $j = 1, 2, .., 5$. These computed X_{min} and X_{max} are used for sorting the stroke sets. This process helps in joining the vowel modifiers/nasal with consonants.

Algorithm 5.2: Sequencing of stroke sets

Input: Consonant based association sets (A^{C}) and vowel/nasal based association sets (A^{V}).

Output: Table 5.6, sorted sets A^{S} are obtained from A^{C} and A^{V} based on X_{min} and X_{max} respectively.

Step 1: Find X_{min} and X_{max} form A^{C} and A^{V} on the basis of their raw points (except class ID $'121'$) respectively.

Step 2: Consider X_{min} of A^{C} and X_{max} of A^{V} for sorting except considering class ID $'121'$. If any association set A^{C} or A^{V} has only one stroke, with class ID $'121'$, then ignore it for sorting and place it at the end of the sorted list.

Table 5.6: Sorted association sets based on *x*-dimension

Sorted association sets
$A_1^{\mathrm{S}} = \{1\ (M)\ (164), 10\ (U)(121)\}$
$A_2^{\mathrm{S}} = \{2\ (M)\ (162)\}$
$A_3^{\mathrm{S}} = \{3\ (L)(101)\}$
$A_4^{\mathrm{S}} = \{3\ (L)(101)\}$
$A_5^{\mathrm{S}} = \{4\ (M)(164), 10\ (U)\ (121)\}$
$A_6^{\mathrm{S}} = \{5\ (M)\ (152)\}$
$A_7^{\mathrm{S}} = \{9\ (L)(101)\}$
$A_8^{\mathrm{S}} = \{6\ (M)(159), 7\ (M)\ (163)\}$
$A_9^{\mathrm{S}} = \{8\ (M)\ (216)\}$
$A_{10}^{\mathrm{S}} = \{\ 10(U)(121)\}$
$A_{11}^{\mathrm{S}} = \{11\ (U)\ (121)\}$

5.3.3 Merging the stroke sets

As discussed in previous section, in a tuple, for the formation of a character, minimum one and maximum four strokes are required. If a character is formed using more than one stroke then forthcoming strokes are known as sub-strokes with respect to initial input stroke. Merging the stroke sets has been proposed as final step in the post-processing phase. In this step, a systematic procedure has been adopted to merge initial stroke and sub-strokes required to form a character. Merging of stroke sets resolves the problem where sub-strokes are separated from initial stroke due to sub-stroke falling in different association or post-segmentation of stroke set. The procedure adopted has been discussed

below for six different cases. However, there are few characters of *Gurmukhi* script that do not require merging of stroke sets. They are provided in set *X* below.

X = {ੳ, ਕ, ਙ, ਛ, ਝ, ਟ, ਹ, ਠ, ਡ, ਢ, ਤ, ਦ, ਫ, ਭ, ਰ, ਲ, ਵ, ੜ, ਿ, ੀ, ੁ, ੇ, ੈ, ੰ, ੱ, ਾ, ੦, ੧, ੨, ੩, ੪, ੫, ੬, ੭, ੮, ੯}.

Remaining characters of *Gurmukhi* script will fall under one of the following six cases.

In **Case I**, a set *P*, as given below, has been defined that includes stroke shapes along with their class ID. If any association set has class ID that belongs to this set *P*, then that class ID will require either class ID ′154′ (⁀) or ′162′ (⏐) for formation of a *Gurmukhi* character. If required class ID (′154′ or ′162′) is present in immediate next association set, then it will borrow that stroke from there. Otherwise, it will be considered as *noise*.

P = {145 (), 151 (), 161 ( ), 167 ( ), 174 (),
202 (), 207 (), 222 (), 223 (), 226 ()}

Case II considers those strokes which themselves are a *Gurmukhi* character, but when they join with another stroke, it changes shape to form a new *Gurmukhi* character. Hence, if class ID ′146′ () or ′182′ () or ′225′ () are present in any association set and next input stroke after sequencing is class ID ′197′ (), then character /ਣ/ called /*ṭāīnká*/ will be recognized as /ਨ/ called /*nanná*/.

Case III handles for those input strokes which are in same association set, but may form two different *Gurmukhi* characters depending upon their overlapping on *y*-axis. As shown below, *Gurmukhi* character /ਏ/ called /*íṛí*/ or /ਣ/ called /*ṇáṇá*/ can be formed with three different combinations. This formation of *Gurmukhi* character will depend upon overlapping of sub-stroke class ID ′147′ with initial stroke class ID ′146′ or ′182′ or ′225′.

Class ID ′147′ () + Class ID ′146′ () = ਏ or ਣ

Class ID ′147′ () + Class ID ′182′ () = ਏ or ਣ

Class ID ′147′ () + Class ID ′225′ ( ) = ਏ or ਣ

Case IV considers merging of class ID ′122′ ( ) and class ID ′124′ ( ) for their possibility of overlapping with each other on *y*-axis, provided they are present in same association set. Hence, merging of class ID ′122′ and ′124′ results in formation of /ੰ/ called vowel modifier /*kanāūṛá*/.

Case V provides for a special instance where repetition of a particular stroke used for formation of vowel modifier leads to formation of a new *Gurmukhi* vowel modifier. Hence, in this case two vowel modifiers /ੁ/ called /*āūnkaṛ*/ or /ੰ/ called /*lã́*/ have been considered where their repetitive occurrence will lead to formation of /ੂ/ called vowel modifier /*dulāinkṛe*/ or /ੰ/ called vowel modifier /*duláwã́*/ respectively.

In **Case VI**, merging of two different Unicodes has been considered. Here, size of two different strokes class ID ′164′ ( ) or ′165′ () with class ID ′154′ ( ) or ′162′ ( ) is matched in order to determine the formation of *Gurmukhi* character ਗ /*gaggá*/ or else it will become ਰਾ /*rá*/.

5.4 Ambiguities in online handwritten *Gurmukhi* script and their solutions

Few problems associated with online handwritten *Gurmukhi* script have been discussed in Chapter 1. These problems create ambiguity in recognizing online handwritten *Gurmukhi* script. Hence, in this proposed methodology, an attempt has been made to resolve some of the critical problems. To resolve these ambiguities, firstly broad set of rules (as discussed in previous sections) pertaining to such problems are identified. In the second step, identified rules are applied for disambiguation. In the forthcoming sub sections, the problems involving ambiguity in online handwritten *Gurmukhi* script and the methodology adopted to resolve these problems have been discussed.

5.4.1 Sequencing and merging

The problem of sequencing and merging arises in online handwritten *Gurmukhi* script when all input strokes belong to same character/word. However, sequence of input strokes may be different which creates difficulty in merging them to form the desired character/word. The *Gurmukhi* word /ਕਿਸਾਨ/, as shown in Figure 1.5, can be written with seven strokes. It is possible that a writer may write the vowel modifier (ਿ) as the last stroke. This way of writing the strokes in abrupt order will create ambiguity for the recognition engine. In this study, this problem has been resolved by sequencing of stroke sets in post-processing phase. Figure 5.2 provides *Gurmukhi* word /ਕਿਸਾਨ/ that has been written with seven strokes. Each stroke belongs to its own association set, *viz.* A_1, A_2, A_3, A_4, A_5 and A_6. Here, association set A_6 has been used for vowel modifier (ਿ). However, after sequencing of stroke sets a proper order is reframed. Now, A_6 will become A_2^S, A_1 will change to A_1^S and so on (as shown in Figure 5.2). This process will give required output for recognized *Gurmukhi* word /ਕਿਸਾਨ/.

5.4.2 Slant writing

Other problem associated with *Gurmukhi* handwriting recognition is slanted writing style of a writer. It happens when a writer follows a distinguished handwriting style in which all the input strokes are written from left bottom to right top direction. Figure 5.3 provides an overview of *Gurmukhi* word /ਰਵਿੰਦਰ/ written in slant writing style. To resolve this problem, each stroke has been included in a separate association set. In the given example (Figure 5.3), each stroke has been shown in a separate association set, *viz.* A_1, A_2, A_3, A_4 and A_5. If any stroke belongs to only one association set and does not share any other association set, then that slant writing will be recognized in desired manner.

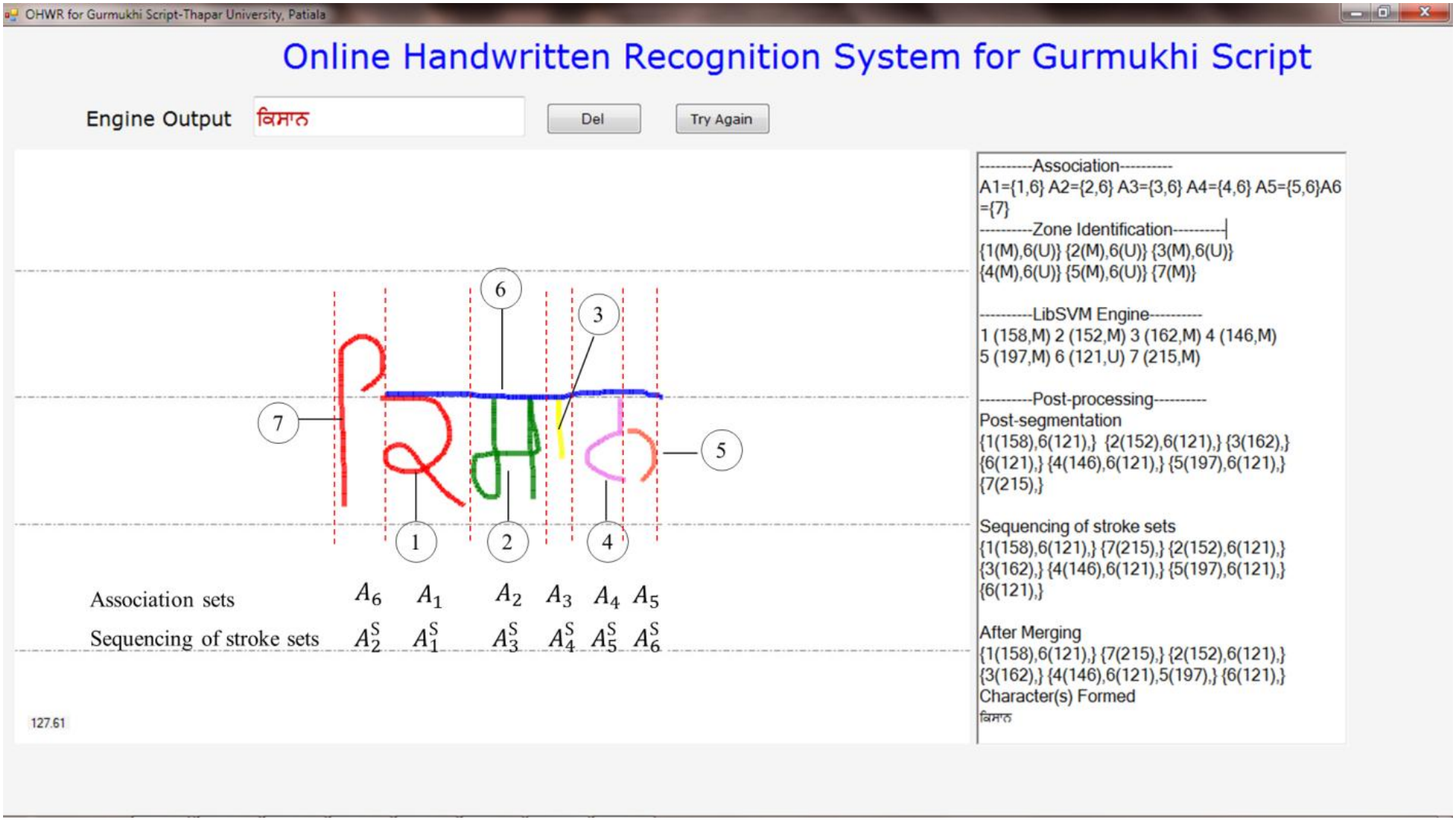


Figure 5.2: Disambiguation through sequencing and merging

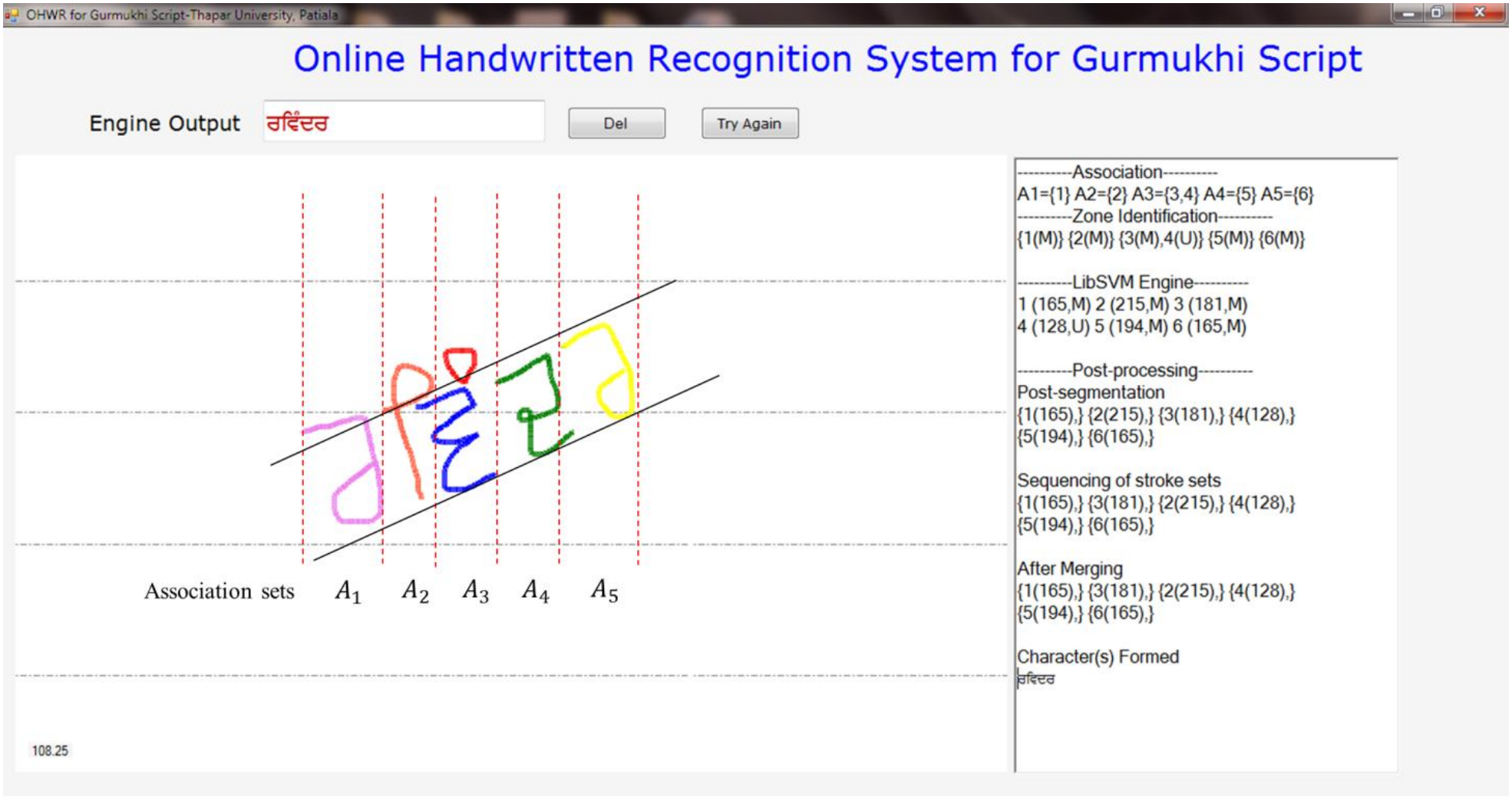


Figure 5.3: Resolving the problem of slant writing style

5.4.3 Gradual reduction in handwriting size

Some writers are habitual of writing *Gurmukhi* words in a pattern that they write the first character of the word significantly large and as they proceed further the size of next characters written keep on reducing. Figure 5.4 illustrates the problem of *Gurmukhi* word /ਸਰਕਾਰੀ/ written by a writer in a unique handwriting style where the previously input character is comparatively larger than the next input characters. As shown in Figure 5.4, when a writer inputs the first stroke, its size on *y*-axis is y_1. At the completion of all the input strokes to form a *Gurmukhi* word, size of the last stroke on *y*-axis is y_2. It is apparent here that size of $y_2 < y_1$. This problem has been resolved with the help of schema of stroke association (as discussed in Section 4.2).

5.4.4 Dot (*bindî)* handling

When a writer just touches the pen tip at the writing surface it is recognized as dot (*bindî*). However, in *Gurmukhi* script, dot (*bindî*) is used at two different places, *i.e.*, upper *bindí* and *pairi bindí*. To recognize this dot (*bindî*), firstly its location in a particular zone is identified. Hence, if it is located in upper writing zone then system will recognize it as upper *bindí*. If it is located in lower zone then system will recognize it as *pairi bindí*. However, if this *bindí* is located somewhere in middle zone then it will be considered as *noise*. As shown in Figure 5.5, *Gurmukhi* word /ਮੁਸ਼ਕਲਾਂ/ has been written by a writer. This word includes both upper *bindí* and *pairi bindí*. Here, location of upper *bindí* in upper zone and *pairi bindí* in lower zone have been shown and that leads to accurate recognition of both.

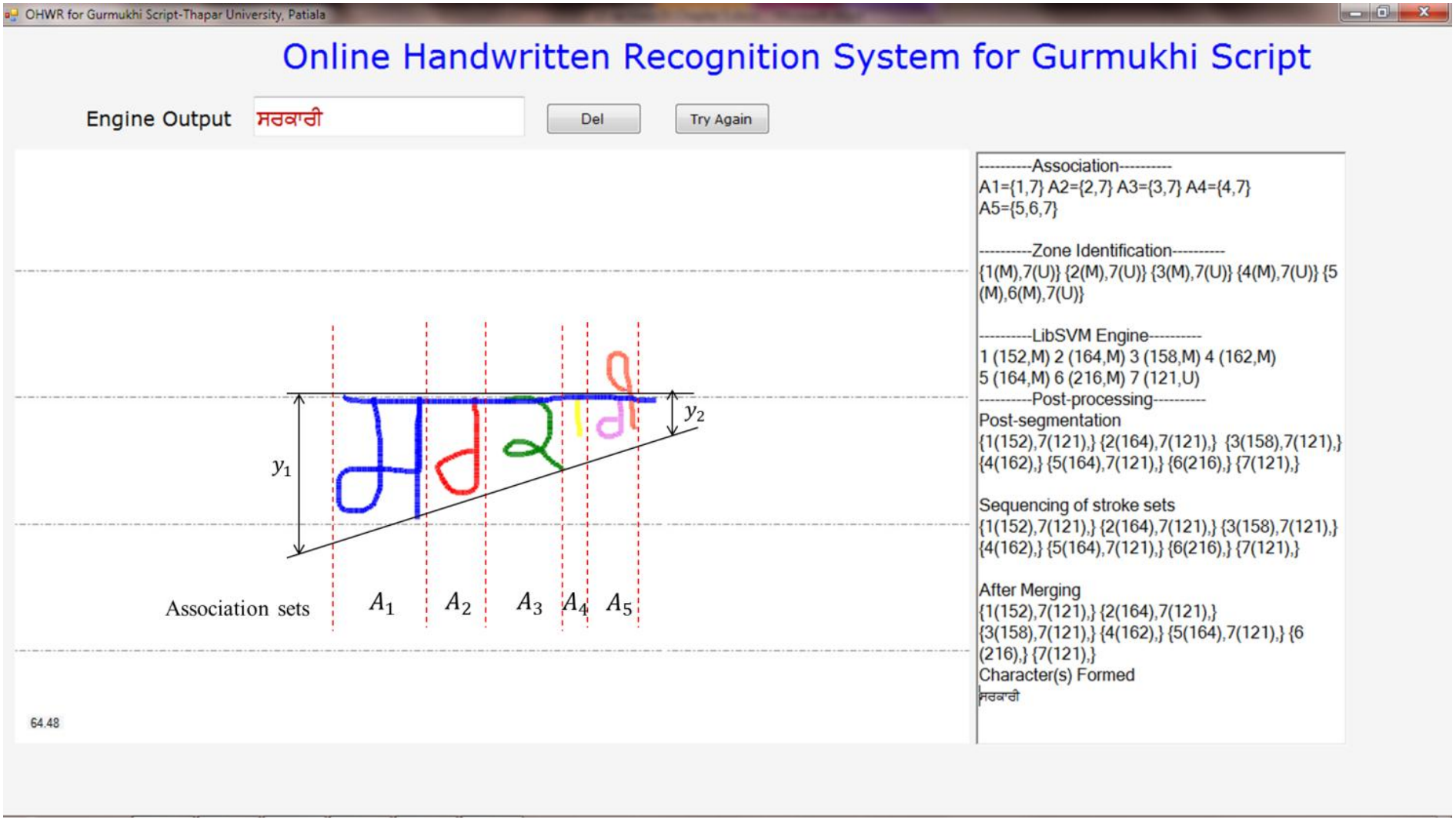


Figure 5.4: Resolving the problem of gradual size reduction

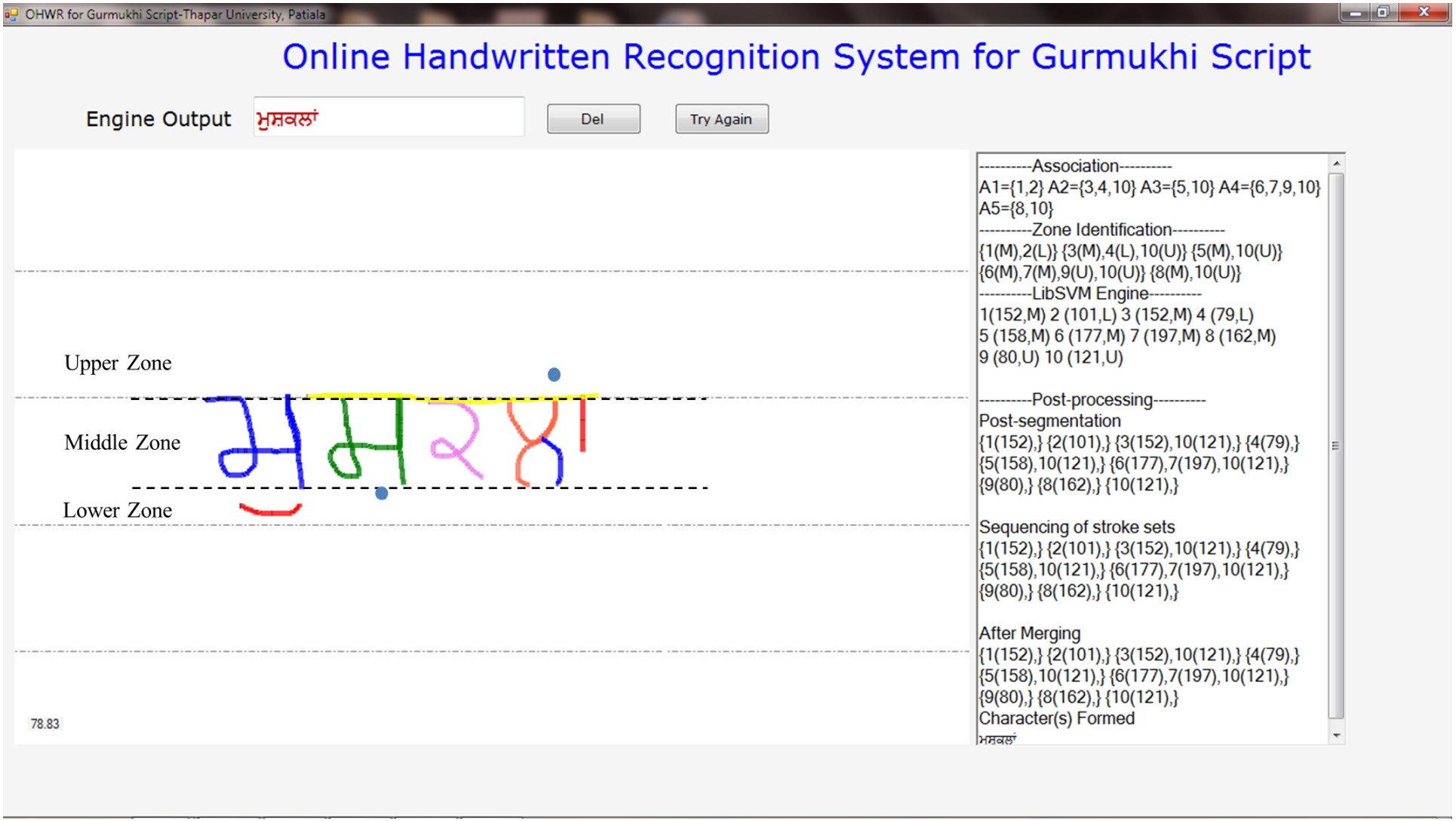


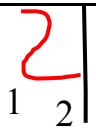
Figure 5.5: Resolving the problem of dot (*bindî*) handling

5.5 Demonstration of recognition process for *Gurmukhi* script

The proposed recognition process becomes smooth after all the above discussed ambiguities are resolved. This section provides brief demonstration of recognition process adopted for *Gurmukhi* script in this study. Demonstration of recognition process for *Gurmukhi* script has been presented in four different cases. Table 5.7 provides demonstration of character formation for *Gurmukhi* script. The character is formed by collection of one/multiple stroke(s). In Table 5.7, *Gurmukhi* character /ਧ/ has been written by a writer with the combination of three strokes. Detail information pertaining to each input stroke has been provided here. For example, when first stroke () is input by a writer then after pre-processing and zone identification, it becomes association set $A_1 = \{1\ (M)\}$. After this association set passes through LibSVM classifier, class ID for this stroke is obtained. In the next phase proposed post-processing steps are applied and activated tuples for A_1^{M} are obtained. Same process is further applied for all the next input strokes. Final output of *Gurmukhi* character /ਧ/ is thus achieved.

Formation of numeral, akshara and word of *Gurmukhi* script has been demonstrated in Table 5.8, Table 5.9, and Table 5.10, respectively.

Table 5.7: Demonstration for character formation of *Gurmukhi* script

Character formation						
Input *Stroke sequence number* (with shape)	**Association sets after pre-processing and zone identification**	**Association sets after LibSVM classifier**	**Post-processing**			**Output**
			Post-segmentation of stroke sets	**Sequencing of stroke sets**	**Merging the stroke sets**	
	$A_1 = \{1\ (M)\}$	$A_1 = \{1\ (M)161\}$	$A_1^{\mathrm{C}} = \{1\ (M)161\}$	$A_1^{\mathrm{S}} = \{1\ (M)161\}$	$A_1^{\mathrm{M}} = \{1\ (M)161\}$	
			Activated tuple for A_1^{M}	a) $T_{[U+0A16]3}$={**161 (0.4)**, 162 (0.3) , 163 (0.3) } b) $T_{[U+0A25]4}$={**161 (0.2)**, 162 (0.2) , 163 (0.2), 121 (0.3)} c) $T_{[U+0A25]3}$={**161 (0.4)**, 154 (0.3) , 163 (0.3) } d) $T_{[U+0A27]3}$={**161 (0.4)** , 162 (0.3), 121 (0.3) } e) $T_{[U+0A27]2}$={**161 (0.5)**, 154 (0.5) } f) $T_{[U+0A2A]2}$={**161 (0.5)**, 162 (0.5) }		**No output**
	$A_1 = \{1\ (M)\}$ $A_2 = \{2\ (M)\}$	$A_1 = \{1\ (M)161\}$ $A_2 = \{2\ (M)162\}$	$A_1^{\mathrm{C}} = \{1\ (M)161\}$ $A_1^{\mathrm{V}} = \{2\ (M)162\}$	$A_1^{\mathrm{S}} = \{1\ (M)161\}$ $A_2^{\mathrm{S}} = \{2\ (M)162\}$	$A_1^{\mathrm{M}} = \begin{Bmatrix} 1\ (M)161, \\ 2\ (M)162 \end{Bmatrix}$	
			Activated tuple for A_1^{M}	a) $T_{[U+0A16]3}$={**161 (0.4)**, **162 (0.3)**, 163 (0.3) } b) $T_{[U+0A25]4}$={**161 (0.2)**, **162 (0.2)**, 163 (0.2), 121 (0.3)} c) $T_{[U+0A27]3}$={**161 (0.4)** , **162 (0.3)**, 121 (0.3) } d) $T_{[U+0A2A]2}$={**161 (0.5)**, **162 (0.5)** }		ਪ ***/pappá/***

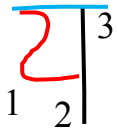
	$A_1 = \begin{Bmatrix} 1\ (M), \\ 3\ (U) \end{Bmatrix}$ $A_2 = \begin{Bmatrix} 2\ (M), \\ 3\ (U) \end{Bmatrix}$	$A_1 = \begin{Bmatrix} 1\ (M)\ 161, \\ 3\ (U)\ 121 \end{Bmatrix}$ $A_2 = \begin{Bmatrix} 2\ (M)\ 162, \\ 3\ (U)\ 121 \end{Bmatrix}$	$A_1^{\mathrm{C}} = \begin{Bmatrix} 1\ (M)161, \\ 3\ (U)121 \end{Bmatrix}$ $A_2^{\mathrm{C}} = \{3\ (M)121\}$ $A_1^{\mathrm{V}} = \{2\ (M)162\}$	$A_1^{\mathrm{S}} = \begin{Bmatrix} 1\ (M)161, \\ 3\ (U)121 \end{Bmatrix}$ $A_2^{\mathrm{S}} = \{2\ (M)162\}$ $A_3^{\mathrm{S}} = \{3\ (U)121\}$	$A_1^{\mathrm{M}} = \begin{Bmatrix} 1\ (M)161\ , \\ 3\ (U)121, \\ 2\ (M)162 \end{Bmatrix}$ $A_2^{\mathrm{M}} = \{3\ (U)121\}$	ঘ ***/tadhá/***
			Activated tuple for A_1^{M}	a) $T_{[U+0A25]4}$={**161** (0.2), **162** (0.2) , 163 (0.2), **121** (0.3)} b) $T_{[U+0A27]3}$={**161** (0.4) , **162** (0.3), **121** (0.3) }		
			Activated tuple for A_2^{M}	72 tuples activated, but $w_3 < 0.8$. So, A_2^{M} is treated as *noise*.		

Table 5.8: Demonstration for numeral formation of *Gurmukhi* script

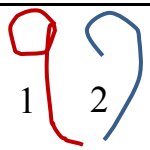
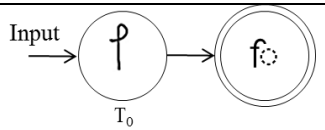
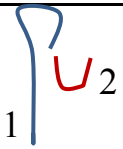
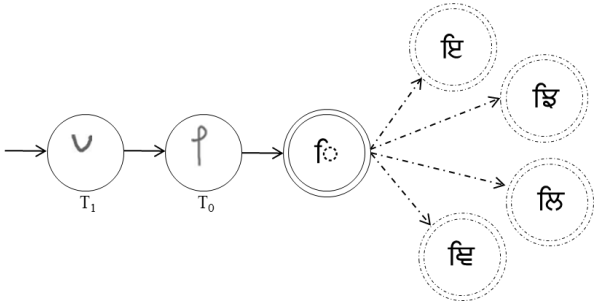
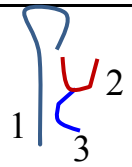
Numeral formation						
Input *Stroke sequence number* (with shape)	**Association sets after pre-processing and zone identification**	**Association sets after LibSVM classifier**	**Post-processing**			**Output**
			Post-segmentation of stroke sets	**Sequencing of stroke sets**	**Merging the stroke sets**	
	$A_1 = \{1\ (M)\}$	$A_1 = \{1\ (M)351\}$	$A_1^C = \{1\ (M)351\}$	$A_1^S = \{1\ (M)351\}$	$A_1^M = \{1\ (M)351\}$	੧ ***/ikk/***
			Activated tuple for A_1^M	a) $T_{[U+0A67]1}$={**351** (1.0) }		
	$A_1 = \{1\ (M)\}$ $A_2 = \{2\ (M)\}$	$A_1 = \{1\ (M)351\}$ $A_2 = \{2\ (M)357\}$	$A_1^C = \{1\ (M)351\}$ $A_2^C = \{2\ (M)357\}$	$A_1^S = \{1\ (M)351\}$ $A_2^S = \{2\ (M)357\}$	$A_1^M = \{1\ (M)351\}$ $A_2^M = \{2\ (M)357\}$	੧ੋ ***/ikk/ /sátt/***
			Activated tuple for A_1^M	a) $T_{[U+0A67]1}$={**351** (1.0) }		
			Activated tuple for A_2^M	a) $T_{[U+0A6D]1}$={**357** (1.0) }		

Table 5.9: Demonstration for akshara formation of *Gurmukhi* script

Akshara formation

Input *Stroke sequence number* (with shape)	Association sets after pre-processing and zone identification	Association sets after LibSVM classifier	Post-processing: Post-segmentation of stroke sets	Post-processing: Sequencing of stroke sets	Post-processing: Merging the stroke sets	Output
 1	$A_1 = \{1\ (M)\}$	$A_1 = \{1\ (M)215\}$	$A_1^{V} = \{1\ (M)215\}$	$A_1^{S} = \{1\ (M)215\}$	$A_1^{M} = \{1\ (M)215\}$	ਝ */sihárí/*
			w_1 w_2 w_3			
Activated tuple for A_1^{M}	$T_{[U+0A3F]1}$={**215** (1.0) }		1.0 1.0 **1.0**			
 1 2	$A_1 = \{1\ (M)\}$ $A_2 = \{2\ (M)\}$	$A_1 = \{1\ (M)215\}$ $A_2 = \{2\ (M)148\}$	$A_1^{V} = \{1\ (M)215\}$ $A_1^{C} = \{2\ (M)148\}$	$A_1^{S} = \{2\ (M)148\}$ $A_2^{S} = \{1\ (M)215\}$	$A_1^{M} = \{2\ (M)148\}$ $A_2^{M} = \{1\ (M)215\}$	
			w_1 w_2 w_3			ਝ */sihárí/*
Activated tuple for A_1^{M}	8 tuples for ਏ */íṛí/* [U+0A72]		- - <0.8			
	4 tuples for ਝ */cajhá/* [U+0A1D]		- - <0.8			
	2 tuples for ਞ */náñá/* [U+0A1E]		- - <0.8			
	4 tuples for ਲ */lallá/* [U+0A32]		- - <0.8			
Activated tuple for A_2^{M}	$T_{[U+0A3F]1}$={**215** (1.0) }		1.0 1.0 **1.0**			



$A_1 = \{1\ (M)\}$

$A_2 = \begin{Bmatrix} 2\ (M), \\ 3\ (M) \end{Bmatrix}$

$A_1 = \{1\ (M)215\}$

$A_2 = \begin{Bmatrix} 2\ (M)148, \\ 3(M)149 \end{Bmatrix}$

$A_1^{V} = \{1\ (M)215\}$

$A_1^{C} = \begin{Bmatrix} 2\ (M)148, \\ 3(M)149 \end{Bmatrix}$

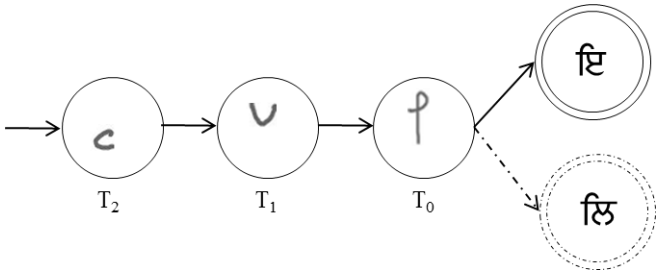
$A_1^{S} = \begin{Bmatrix} 2\ (M)148, \\ 3(M)149 \end{Bmatrix}$

$A_2^{S} = \{1\ (M)215\}$

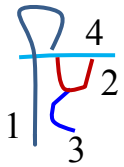
$A_1^{M} = \begin{Bmatrix} 2\ (M)148, \\ 3(M)149 \end{Bmatrix}$

$A_2^{M} = \{1\ (M)215\}$

		w_1	w_2	w_3
Activated tuple for A_1^{M}	$T_{[U+0A72]2}$={**148 (0.5)**, **149 (0.5)**}	1.0	1.0	**1.0**
	$T_{[U+0A72]3}$={**148 (0.4)**, **149 (0.4)**, 121 (0.2)}	0.3	1.0	0.4
	$T_{[U+0A32]3}$={**148 (0.3)**, **149 (0.3)**, 197 (0.4)}	0.7	1.0	0.6
	$T_{[U+0A32]4}$={**148 (0.3)**, **149 (0.3)**, 197 (0.3), 121 (0.1)}	0.5	1.0	0.6
Activated tuple for A_2^{M}	$T_{[U+0A3F]1}$={**215 (1.0)** }	1.0	1.0	**1.0**



ਫ + ਿ = ਫਿ



$A_1 = \{1\ (M)\}$

$A_2 = \begin{Bmatrix} 2\ (M), \\ 3\ (M), \\ 4\ (U) \end{Bmatrix}$

$A_1 = \{1\ (M)215\}$

$A_2 = \begin{Bmatrix} 2\ (M)148, \\ 3\ (M)149, \\ 4\ (U)121 \end{Bmatrix}$

$A_1^{V} = \{1\ (M)215\}$

$A_1^{C} = \begin{Bmatrix} 2\ (M)148, \\ 3\ (M)149, \\ 4\ (U)121 \end{Bmatrix}$

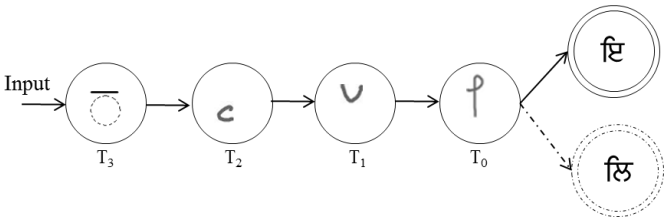
$A_1^{S} = \begin{Bmatrix} 2\ (M)148, \\ 3\ (M)149, \\ 4\ (U)121 \end{Bmatrix}$

$A_2^{S} = \{1\ (M)215\}$

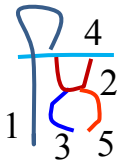
$A_1^{M} = \begin{Bmatrix} 2\ (M)148, \\ 3\ (M)149, \\ 4\ (U)121 \end{Bmatrix}$

$A_2^{M} = \{1\ (M)215\}$

		w_1	w_2	w_3
Activated tuple for A_1^{M}	$T_{[U+0A72]3}$={**148 (0.4)**, **149 (0.4)**, **121 (0.2)**}	0.3	1.0	0.4
	$T_{[U+0A32]4}$={**148 (0.3)**, **149 (0.3)**, 197 (0.3), **121 (0.1)**}	0.5	1.0	0.7
Activated tuple for A_2^{M}	$T_{[U+0A3F]1}$={**215 (1.0)** }	1.0	1.0	**1.0**



ਫ + ਿ = ਫਿ



$A_1 = \{1\ (M)\}$

$A_2 = \begin{Bmatrix} 2\ (M), \\ 3\ (M), 4(U) \\ 5(M) \end{Bmatrix}$

$A_1 = \{1\ (M)215\}$

$A_2 = \begin{Bmatrix} 2\ (M)148, \\ 3\ (M)149, \\ 4\ (U)121, \\ 5\ (M)197 \end{Bmatrix}$

$A_1^{\mathrm{V}} = \{1\ (M)215\}$

$A_1^{\mathrm{C}} = \begin{Bmatrix} 2\ (M)148, \\ 3\ (M)149, \\ 4\ (U)121, \\ 5\ (M)197 \end{Bmatrix}$

$A_1^{\mathrm{S}} = \begin{Bmatrix} 2\ (M)148, \\ 3\ (M)149, \\ 4\ (U)121, \\ 5\ (M)197 \end{Bmatrix}$

$A_2^{\mathrm{S}} = \{1\ (M)215\}$

$A_1^{\mathrm{M}} = \begin{Bmatrix} 2\ (M)148, \\ 3\ (M)149, \\ 4\ (U)121, \\ 5\ (M)197 \end{Bmatrix}$

$A_2^{\mathrm{M}} = \{1\ (M)215\}$

ਲ + ਿ = ਲਿ

		w_1	w_2	w_3
Activated tuple for A_1^{M}	$T_{[U+0A32]4}$={**148 (0.3)**, **149 (0.3)**, **197 (0.3)**, **121 (0.1)**}	1.0	1.0	**1.0**
Activated tuple for A_2^{M}	$T_{[U+0A3F]1}$={**215 (1.0)** }	1.0	1.0	**1.0**

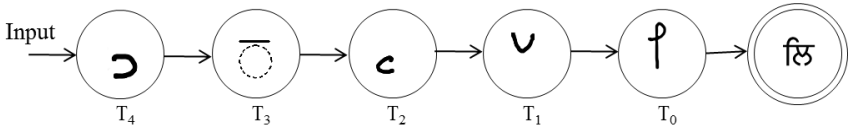


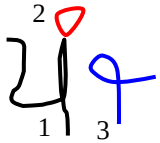
Table 5.10: Demonstration for word formation of *Gurmukhi* script

Word formation						
Input *Stroke sequence number* (with shape)	**Association sets after pre-processing and zone identification**	**Association sets after LibSVM classifier**	**Post-processing**			**Final output**
			Post-segmentation of stroke sets	**Sequencing of stroke sets**	**Merging the stroke sets**	
 1	$A_1 = \{1\ (M)\}$	$A_1 = \{1\ (M)159\}$	$A_1^{\mathrm{C}} = \{1\ (M)159\}$	$A_1^{\mathrm{S}} = \{1\ (M)159\}$	$A_1^{\mathrm{M}} = \{1\ (M)159\}$	ਪ

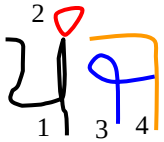
Activated tuple for A_1^{M}	w_1	w_2	w_3	Output
$T_{[U+0A2A]1}$={**159 (1.0)** }	1.0	1.0	**1.0**	ਪ

 2 1	$A_1 = \{1\ (M), 2\ (U)\}$	A_1 $= \{1\ (M)159,$ $2\ (U)\ 128\}\}$	$A_1^{\mathrm{C}} = \{1\ (M)159\}$ $A_1^{\mathrm{V}} = \{2\ (U)128\}$	$A_1^{\mathrm{S}} = \{1\ (M)159\}$ $A_2^{\mathrm{S}} = \{1\ (U)128\}$	$A_1^{\mathrm{M}} = \{1\ (M)159\}$ $A_2^{\mathrm{M}} = \{2\ (U)128\}$	ਪੰ

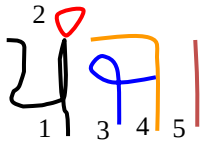
		w_1	w_2	w_3	Output
Activated tuple for A_1^{M}	$T_{[U+0A2A]1}$={**159 (1.0)** }	1.0	1.0	**1.0**	ਪ
Activated tuple for A_2^{M}	$T_{[U+0A70]1}$={**128 (1.0)** }	1.0	1.0	**1.0**	◌ੰ

	$A_1 = \begin{Bmatrix} 1\ (M), \\ 2\ (U) \end{Bmatrix}$ $A_2 = \{3\ (M)\}$	$A_1 = \begin{Bmatrix} 1\ (M)\ 159, \\ 2\ (U)128 \end{Bmatrix}$ $A_2 = \{3\ (M)\ 174\}$	$A_1^{\mathrm{C}} = \{1\ (M)159\}$ $A_1^{\mathrm{V}} = \{2\ (U)128\}$ $A_2^{\mathrm{C}} = \{3\ (M)174\}$	$A_1^{\mathrm{S}} = \{1\ (M)159\}$ $A_2^{\mathrm{S}} = \{1\ (U)128\}$ $A_3^{\mathrm{S}} = \{3\ (M)174\}$	$A_1^{\mathrm{M}} = \{1\ (M)159\}$ $A_2^{\mathrm{M}} = \{2\ (U)128\}$ $A_3^{\mathrm{M}} = \{3\ (M)174\}$	

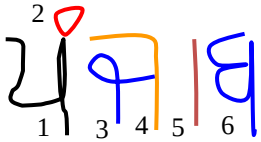
		w_1	w_2	w_3	Output	
Activated tuple for A_1^{M}	$T_{[\mathrm{U+0A2A}]1}$={**159** **(1.0)** }	1.0	1.0	**1.0**	ਪ	ਪੰ
Activated tuple for A_2^{M}	$T_{[\mathrm{U+0A70}]1}$={**128** **(1.0)** }	1.0	1.0	**1.0**	ੰ	
Activated tuple for A_3^{M}	3 tuples are activated. But $\boldsymbol{w_3 < 0.8}$, for all the tuples.	-	-	-	*noise*	

	$A_1 = \begin{Bmatrix} 1\ (M), \\ 2\ (U) \end{Bmatrix}$ $A_2 = \begin{Bmatrix} 3\ (M), \\ 4\ (M) \end{Bmatrix}$	$A_1 = \begin{Bmatrix} 1\ (M)\ 159, \\ 2\ (U)128 \end{Bmatrix}$ $A_2 = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M)154 \end{Bmatrix}$	$A_1^{\mathrm{C}} = \{1\ (M)159\}$ $A_1^{\mathrm{V}} = \{2\ (U)128\}$ $A_2^{\mathrm{C}} = \{3\ (M)174\}$ $A_2^{\mathrm{V}} = \{4\ (M)154\}$	$A_1^{\mathrm{S}} = \{1\ (M)159\}$ $A_2^{\mathrm{S}} = \{1\ (U)128\}$ $A_3^{\mathrm{S}} = \{3\ (M)174\}$ $A_4^{\mathrm{S}} = \{4\ (M)154\}$	$A_1^{\mathrm{M}} = \{1\ (M)159\}$ $A_2^{\mathrm{M}} = \{2\ (U)128\}$ $A_3^{\mathrm{M}} = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M)154 \end{Bmatrix}$	

		w_1	w_2	w_3	Output	
Activated tuple for A_1^{M}	$T_{[\mathrm{U+0A2A}]1}$={**159** **(1.0)** }	1.0	1.0	**1.0**	ਪ	ਪੰਜ
Activated tuple for A_2^{M}	$T_{[\mathrm{U+0A70}]1}$={**128** **(1.0)** }	1.0	1.0	**1.0**	ੰ	
Activated tuple for A_3^{M}	$T_{[\mathrm{U+0A1C}]2}$={**174** **(0.5)**, **154** **(0.5)** }	1.0	1.0	**1.0**	ਜ	

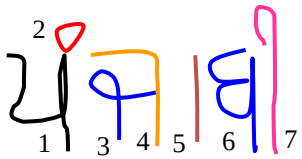
	$A_1 = \begin{Bmatrix} 1\ (M), \\ 2\ (U) \end{Bmatrix}$ $A_2 = \begin{Bmatrix} 3\ (M), \\ 4\ (M) \end{Bmatrix}$ $A_3 = \{5\ (M)\}$	$A_1 = \begin{Bmatrix} 1\ (M)\ 159, \\ 2\ (U) 128 \end{Bmatrix}$ $A_2 = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M) 154 \end{Bmatrix}$ $A_3 = \{5\ (M)\ 162\}$	$A_1^{\mathrm{C}} = \{1\ (M) 159\}$ $A_1^{\mathrm{V}} = \{2\ (U) 128\}$ $A_2^{\mathrm{C}} = \{3\ (M) 174\}$ $A_2^{\mathrm{V}} = \{4\ (M) 154\}$ $A_3^{\mathrm{V}} = \{5\ (M) 162\}$	$A_1^{\mathrm{S}} = \{1\ (M) 159\}$ $A_2^{\mathrm{S}} = \{1\ (U) 128\}$ $A_3^{\mathrm{S}} = \{3\ (M) 174\}$ $A_4^{\mathrm{S}} = \{4\ (M) 154\}$ $A_5^{\mathrm{S}} = \{5\ (M) 162\}$	$A_1^{\mathrm{M}} = \{1\ (M) 159\}$ $A_2^{\mathrm{M}} = \{2\ (U) 128\}$ $A_3^{\mathrm{M}} = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M) 154 \end{Bmatrix}$ $A_4^{\mathrm{M}} = \{5\ (M) 162\}$

		w_1	w_2	w_3	Output	ਪੰਜਾ
Activated tuple for A_1^{M}	$T_{[\mathrm{U+0A2A}]1}$={**159** **(1.0)**}	1.0	1.0	**1.0**	ਪ	
Activated tuple for A_2^{M}	$T_{[\mathrm{U+0A70}]1}$={**128** **(1.0)**}	1.0	1.0	**1.0**	◌ੰ	
Activated tuple for A_3^{M}	$T_{[\mathrm{U+0A1C}]2}$={**174** **(0.5)**, **154** **(0.5)**}	1.0	1.0	**1.0**	ਜ	
Activated tuple for A_4^{M}	$T_{[\mathrm{U+0A3E}]1}$={**162** **(1.0)**}	1.0	1.0	**1.0**	◌ਾ	



$A_1 = \begin{Bmatrix} 1\ (M), \\ 2\ (U) \end{Bmatrix}$	$A_1 = \begin{Bmatrix} 1\ (M)\ 159, \\ 2\ (U) 128 \end{Bmatrix}$	$A_1^{\mathrm{C}} = \{1\ (M) 159\}$	$A_1^{\mathrm{S}} = \{1\ (M) 159\}$	$A_1^{\mathrm{M}} = \{1\ (M) 159\}$
		$A_1^{\mathrm{V}} = \{2\ (U) 128\}$	$A_2^{\mathrm{S}} = \{1\ (U) 128\}$	$A_2^{\mathrm{M}} = \{2\ (U) 128\}$
$A_2 = \begin{Bmatrix} 3\ (M), \\ 4\ (M) \end{Bmatrix}$	$A_2 = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M) 154 \end{Bmatrix}$	$A_2^{\mathrm{C}} = \{3\ (M) 174\}$	$A_3^{\mathrm{S}} = \{3\ (M) 174\}$	$A_3^{\mathrm{M}} = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M) 154 \end{Bmatrix}$
		$A_2^{\mathrm{V}} = \{4\ (M) 154\}$	$A_4^{\mathrm{S}} = \{4\ (M) 154\}$	
$A_3 = \{5\ (M)\}$	$A_3 = \{5\ (M)\ 162\}$	$A_3^{\mathrm{V}} = \{5\ (M) 162\}$	$A_5^{\mathrm{S}} = \{5\ (M) 162\}$	$A_4^{\mathrm{M}} = \{5\ (M) 162\}$
$A_4 = \{6\ (M)\}$	$A_4 = \{6\ (M)\ 203\}$	$A_3^{\mathrm{C}} = \{6\ (M) 203\}$	$A_6^{\mathrm{S}} = \{6\ (M) 203\}$	$A_5^{\mathrm{M}} = \{6\ (M) 203\}$

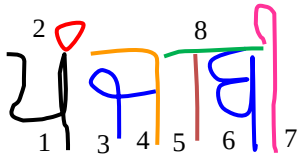
		w_1	w_2	w_3	Output	
Activated tuple for A_1^{M}	$T_{[U+0A2A]1}$={**159 (1.0)** }	1.0	1.0	**1.0**	ਪ	ਪੰਜਾਬ
Activated tuple for A_2^{M}	$T_{[U+0A70]1}$={**128 (1.0)** }	1.0	1.0	**1.0**	◌ੰ	
Activated tuple for A_3^{M}	$T_{[U+0A1C]2}$={**174 (0.5), 154 (0.5)** }	1.0	1.0	**1.0**	ਜ	
Activated tuple for A_4^{M}	$T_{[U+0A3E]1}$={**162 (1.0)** }	1.0	1.0	**1.0**	◌ਾ	
Activated tuple for A_5^{M}	$T_{[U+0A2C]1}$={**203 (1.0)** }	1.0	1.0	**1.0**	ਬ	



$A_1 = \begin{Bmatrix} 1\ (M), \\ 2\ (U) \end{Bmatrix}$	$A_1 = \begin{Bmatrix} 1\ (M)\ 159, \\ 2\ (U) 128 \end{Bmatrix}$	$A_1^{C} = \{1\ (M) 159\}$	$A_1^{S} = \{1\ (M) 159\}$	$A_1^{M} = \{1\ (M) 159\}$	
		$A_1^{V} = \{2\ (U) 128\}$	$A_2^{S} = \{1\ (U) 128\}$	$A_2^{M} = \{2\ (U) 128\}$	
$A_2 = \begin{Bmatrix} 3\ (M), \\ 4\ (M) \end{Bmatrix}$	$A_2 = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M) 154 \end{Bmatrix}$	$A_2^{C} = \{3\ (M) 174\}$	$A_3^{S} = \{3\ (M) 174\}$	$A_3^{M} = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M) 154 \end{Bmatrix}$	
		$A_2^{V} = \{4\ (M) 154\}$	$A_4^{S} = \{4\ (M) 154\}$		
$A_3 = \{5\ (M)\}$	$A_3 = \{5\ (M)\ 162\}$	$A_3^{V} = \{5\ (M) 162\}$	$A_5^{S} = \{5\ (M) 162\}$	$A_4^{M} = \{5\ (M) 162\}$	
$A_4 = \{6\ (M)\}$	$A_4 = \{6\ (M)\ 203\}$	$A_3^{C} = \{6\ (M) 203\}$	$A_6^{S} = \{6\ (M) 203\}$	$A_5^{M} = \{6\ (M) 203\}$	
$A_5 = \{7\ (M)\}$	$A_5 = \{7\ (M)\ 216\}$	$A_4^{V} = \{7\ (M) 216\}$	$A_7^{S} = \{7\ (M) 216\}$	$A_6^{M} = \{7\ (M) 216\}$	

		w_1	w_2	w_3	Output
Activated tuple for A_1^{M}	$T_{[U+0A2A]1}$={**159** **(1.0)** }	1.0	1.0	**1.0**	ਪ
Activated tuple for A_2^{M}	$T_{[U+0A70]1}$={**128** **(1.0)** }	1.0	1.0	**1.0**	◌ੰ
Activated tuple for A_3^{M}	$T_{[U+0A1C]2}$={**174** **(0.5)**, **154** **(0.5)** }	1.0	1.0	**1.0**	ਜ
Activated tuple for A_4^{M}	$T_{[U+0A3E]1}$={**162** **(1.0)** }	1.0	1.0	**1.0**	◌ਾ
Activated tuple for A_5^{M}	$T_{[U+0A2C]1}$={**203** **(1.0)** }	1.0	1.0	**1.0**	ਬ
Activated tuple for A_6^{M}	$T_{[U+0A40]1}$={**216** **(1.0)** }	1.0	1.0	**1.0**	◌ੀ

ਪੰਜਾਬੀ



$A_1 = \begin{Bmatrix} 1\ (M), \\ 2\ (U) \end{Bmatrix}$

$A_2 = \begin{Bmatrix} 3\ (M), \\ 4\ (M) \end{Bmatrix}$

$A_3 = \{5\ (M), 8(U)\}$

$A_4 = \{6\ (M), 8(U)\}$

$A_5 = \{7\ (M)\}$

$A_1 = \begin{Bmatrix} 1\ (M)\ 159, \\ 2\ (U)128 \end{Bmatrix}$

$A_2 = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M)154 \end{Bmatrix}$

$A_3 = \begin{Bmatrix} 5\ (M)\ 162, \\ 8\ (U)121 \end{Bmatrix}$

$A_4 = \begin{Bmatrix} 6\ (M)\ 203, \\ 8\ (U)121 \end{Bmatrix}$

$A_5 = \{7\ (M)\ 216\}$

$A_1^{\mathrm{C}} = \{1\ (M)159\}$

$A_1^{\mathrm{V}} = \{2\ (U)128\}$

$A_2^{\mathrm{C}} = \{3\ (M)174\}$

$A_2^{\mathrm{V}} = \{4\ (M)154\}$

$A_3^{\mathrm{V}} = \{5\ (M)162\}$

$A_3^{\mathrm{C}} = \{8\ (U)121\}$

$A_4^{\mathrm{C}} = \begin{Bmatrix} 6\ (M)\ 203, \\ 8\ (U)121 \end{Bmatrix}$

$A_4^{\mathrm{V}} = \{7\ (M)216\}$

$A_1^{\mathrm{S}} = \{1\ (M)159\}$

$A_2^{\mathrm{S}} = \{1\ (U)128\}$

$A_3^{\mathrm{S}} = \{3\ (M)174\}$

$A_4^{\mathrm{S}} = \{4\ (M)154\}$

$A_5^{\mathrm{S}} = \{5\ (M)162\}$

$A_6^{\mathrm{S}} = \begin{Bmatrix} 6\ (M)\ 203, \\ 8\ (U)121 \end{Bmatrix}$

$A_7^{\mathrm{S}} = \{7\ (M)216\}$

$A_8^{\mathrm{S}} = \{8\ (U)121\}$

$A_1^{\mathrm{M}} = \{1\ (M)159\}$

$A_2^{\mathrm{M}} = \{2\ (U)128\}$

$A_3^{\mathrm{M}} = \begin{Bmatrix} 3\ (M)\ 174, \\ 4\ (M)154 \end{Bmatrix}$

$A_4^{\mathrm{M}} = \{5\ (M)162\}$

$A_5^{\mathrm{M}} = \begin{Bmatrix} 6\ (M)\ 203, \\ 8\ (U)121 \end{Bmatrix}$

$A_6^{\mathrm{M}} = \{7\ (M)216\}$

$A_7^{\mathrm{M}} = \{8\ (U)121\}$

		w_1	w_2	w_3	Output
Activated tuple for A_1^{M}	$T_{[U+0A2A]1}$={**159** **(1.0)** }	1.0	1.0	**1.0**	ਪ
Activated tuple for A_2^{M}	$T_{[U+0A70]1}$={**128** **(1.0)** }	1.0	1.0	**1.0**	◌ੰ
Activated tuple for A_3^{M}	$T_{[U+0A1C]2}$={**174** **(0.5)**, **154** **(0.5)** }	1.0	1.0	**1.0**	ਜ
Activated tuple for A_4^{M}	$T_{[U+0A3E]1}$={**162** **(1.0)** }	1.0	1.0	**1.0**	◌ਾ
Activated tuple for A_5^{M}	$T_{[U+0A2C]1}$={**203** **(1.0)** }	1.0	1.0	**1.0**	ਬ
Activated tuple for A_6^{M}	$T_{[U+0A40]1}$={**216** **(1.0)** }	1.0	1.0	**1.0**	◌ੀ
Activated tuple for A_7^{M}	72 tuples activated, but $w_3 < 0.8$. So, A_2^{M} is treated as *noise*.	-	-	-	*noise*

ਪੰਜਾਬੀ

5.6 Experimental results and discussion

This section consists of the results of the experiments conducted in this work for online handwritten *Gurmukhi* characters and words using proposed methodology. The data required for this purpose was collected from different writers who use *Gurmukhi* script. Next sub sections provide separate discussions on recognition results achieved for *Gurmukhi* characters, numerals, aksharas and words.

5.6.1 Recognition results for *Gurmukhi* characters and numerals

In *Gurmukhi* script a character can be formed with the help of a single stroke or multiple strokes. Although some characters may be formed with one stroke only, but formation of some other characters require combination of strokes. In the present approach, real time post-processing has been done, where with input of every new stroke post-processor recognizes the character formed instead of recognizing the character at the end of all input strokes. However, a major problem associated with online handwritten *Gurmukhi* script is with regard to confusion created by similar words. A brief discussion of this confusion matrix has been given in Chapter 1 (Table 1.6). This problem of similar looking strokes results in low recognition accuracy for that particular character. Table 5.11 provides detail of recognition accuracy achieved for all the *Gurmukhi* characters along with the information on number of samples collected, number of samples recognized, and characters with which confusion is noticed.

Table 5.11: Recognition rates for *Gurmukhi* characters

***Gurmukhi* Character**	**Number of Samples**	**Recognized**	**Misclassified with**	**Accuracy (in %)**
ੳ	41	41	-	100.0
ਅ	47	45	2 (ਘ)	95.7
ੲ	45	43	2 (ਟ)	95.6
ਸ	49	48	1 (ਮ)	98.0
ਹ	41	38	2 (ਰ), 1 (ਤ)	92.7
ਕ	44	44	-	100.0
ਖ	43	41	1 (ਧ), 1 (ਮ)	95.3
ਗ	27	25	2 (ਰ)	92.6
ਘ	30	29	1 (ਅ)	96.7
ਙ	26	26	-	100.0
ਚ	42	39	2 (ਜ), 1 (ਰ)	92.9

ਛ	38	38	-	100.0
ਜ	38	37	1 (ਚ)	97.4
ਝ	36	34	2 (ਲ)	94.4
ਞ	32	29	3 (ਵ)	90.6
ਟ	41	40	1 (ਦ)	97.6
ਠ	33	33	-	100.0
ਡ	42	39	2 (ਤ), 1 (ਭ)	92.9
ਢ	39	35	2 (ਦ), 1 (ਟ), 1 (ਫ)	89.7
ਣ	43	40	2 (ਏ), 1 (ਟ)	93.0
ਤ	40	36	2 (ਭ), 1 (ਡ) , 1 (ਹ)	90.0
ਥ	46	45	1 (ਖ)	97.8
ਦ	43	40	3 (ਢ)	93.0
ਧ	46	44	2 (ਖ)	95.7
ਨ	42	41	1 (ਟ)	97.6
ਪ	38	38	-	100.0
ਫ	37	34	1 (ਦ), 2 (ਢ)	91.9
ਬ	44	44	-	100.0
ਭ	43	40	2 (ਤ), 1 (ਡ)	93.0
ਮ	44	44	-	100.0
ਯ	36	33	3 (ਧ)	91.7
ਰ	47	45	2 (ਹ)	95.7
ਲ	44	41	2 (ਏ), 1 (ਝ)	93.2
ਵ	38	37	1 (ਬ)	97.4
ੜ	35	33	1 (ਤ), 1 (ਡ)	94.3
Total	**1400**	**1339**	**61**	**95.6**

Table 5.12 provides range wise recognition results for *Gurmukhi* characters. Among these results, least recognition rate has been obtained for *Gurmukhi* character /ਢ/ as it greatly confuses with other *Gurmukhi* characters, *viz.* /ਦ/, /ਟ/, and /ਫ/. While collecting samples, it was noticed that 21 *Gurmukhi* characters /ਉ/, /ਅ/, /ਹ/, /ਕ/, /ਘ/, /ਙ/, /ਛ/, /ਞ/, /ਟ/, /ਠ/, /ਡ/, /ਢ/, /ਤ/, /ਦ/, /ਪ/, /ਫ/, /ਬ/, /ਭ/, /ਮ/, /ਰ/, and /ਵ/ are those which may be written with single stroke or multiple stroke, but writers have preferred to write these characters with single stroke. It has been observed that characters written with single stroke achieve higher recognition accuracy than characters written with multiple strokes. However, two *Gurmukhi* characters /ਥ/ and /ੜ/ are in the category where a character is always written with multiple strokes. In spite of this, a high recognition accuracy has been achieved for these characters. It is also worth mentioning here that higher recognition accuracy has

been achieved for /ਖ/, /ਸ਼/, and /ਜ਼/ as compared to *Gurmukhi* characters /ਲ਼/, /ਗ਼/, and /ਫ਼/ (Table 5.13).

Table 5.12: Range wise recognition rate for *Gurmukhi* characters

Range of recognition rate (in %)	***Gurmukhi* Character(s)**
Less than 90.0	ਢ
90.0 – 92.0	ਫ, ਯ, ਵ, ਤ
92.1 – 94.0	ਲ, ਣ, ਦ, ਭ, ਚ, ਡ, ਹ, ਗ
94.1 – 96.0	ਅ, ਧ, ਰ, ਏ, ਖ, ਝ, ੜ
96.1 – 98.0	ਥ, ਟ, ਨ, ਜ, ਵ, ਘ
98.1 – 100.0	ੳ, ਕ, ਙ, ਛ, ਠ, ਪ, ਬ, ਮ, ਸ

Table 5.13: Recognition rate for *Gurmukhi* characters with /*pairi bindí*/

***Gurmukhi* Character**	ਖ਼	ਸ਼	ਜ਼	ਲ਼	ਗ਼	ਫ਼
Recognition rate (in %)	97.4	98.2	96.5	93.2	91.5	92.4

In Table 5.14 recognition results achieved for *Gurmukhi* numerals has been presented. An overall recognition accuracy of 96.8% has been achieved for *Gurmukhi* numeral recognition. Among *Gurmukhi* numerals, two digits /੬/ and /੯/ may be written with the help of either single stroke or multiple strokes. If these numerals are written with multiple strokes, their recognition accuracy decreases by 0.7% and 2.1%, respectively, when these numerals are written with single stroke.

Table 5.14: Recognition rate (in %) for *Gurmukhi* numerals

Gurmukhi numerals	੦	੧	੨	੩	੪	੫	੬	੭	੮	੯
Single stroke sequence	98.2	96.7	97.0	95.9	97.3	97.2	96.5	95.2	97.2	96.7
Multiple stroke sequence	-	-	-	-	-	-	95.8	-	-	94.6

5.6.2 Recognition results for *Gurmukhi* aksharas and words

This section includes a discussion on experimental results carried out for the recognition of online handwritten *Gurmukhi* aksharas and words using the procedure discussed in previous sections. Here, use of valid combination of two different classes of *Gurmukhi* script for akshara formation has been discussed. These combinations are

either formed with a set of 35 *Gurmukhi* characters, 9 vowel modifiers, *viz.* /ਾ/, /ਿ/, /ੀ/, /ੁ/, /ੂ/, /ੇ/, /ੈ/, /ੋ/ and /ੌ/ or 2 nasal symbols, *viz.* /ਂ/ and /ਃ/. With the help of the proposed algorithm, real time akshara formation using online handwritten stroke input has been carried out by testing a total sample of 8,880 aksharas collected from 30 different writers.

Detailed experiment results are presented in following tables. Table 5.15 contains the recognition accuracy of handwritten *Gurmukhi* characters with the combination of all vowel modifiers. In this case, there are few *Gurmukhi* characters which are written alone and no combination of vowel modifier can be attached to them in order to form *Gurmukhi* akshara. In majority of experimental results, high degree of recognition accuracy has been achieved. However, the recognition accuracy is quite low whenever any *Gurmukhi* akshara is formed with the combination of *Gurmukhi* character with /ੈ/ and /ੌ/. The main problem associated with writing these vowel modifiers is that while writing them there is a need to take the pen up and down twice. Results shown in Table 5.15 reveal that promising results have been achieved in case of aksharas formed using three vowel modifier, *viz.* /ਾ/, /ਿ/, and /ੀ/. Moreover, higher recognition accuracy has also been achieved in case of few characters, *viz.* /ੳ/, /ਅ/, /ਸ/, /ਕ/, /ਙ/, /ਟ/, /ਠ/, /ਪ/, and /ਮ/ especially when they are written with their vowel modifiers to form akshara.

Table 5.15: Recognition accuracy (in %) achieved for *Gurmukhi* aksharas

	ਾ	ਿ	ੀ	ੁ	ੂ	ੇ	ੈ	ੋ	ੌ
ੳ	-	-	-	86.7	83.3	-	-	-	-
ਅ	86.7	-	-	-	-	-	86.7	-	83.3
ੲ	-	86.7	76.7	-	-	73.3	-	-	-
ਸ	83.3	76.7	86.7	80.0	60.0	73.3	86.7	90.0	80.0
ਹ	70.0	70.0	73.3	80.0	93.3	73.3	70.0	76.7	63.3
ਕ	93.3	73.3	86.7	93.3	86.7	93.3	70.0	93.3	63.3
ਖ	86.7	73.3	76.7	83.3	90.0	63.3	66.7	90.0	70.0
ਗ	76.7	70.0	83.3	73.3	63.3	73.3	63.3	73.3	66.7
ਘ	90.0	73.3	70.0	83.3	73.3	80.0	86.7	80.0	70.0
ਙ	73.3	86.7	86.7	86.7	83.3	83.3	73.3	93.3	70.0
ਚ	86.7	70.0	70.0	83.3	80.0	70.0	63.3	93.3	66.7
ਛ	80.0	76.7	73.3	83.3	86.7	90.0	73.3	73.3	70.0

ਜ	76.7	80.0	83.3	80.0	86.7	86.7	70.0	73.3	70.0
ਝ	90.0	86.7	86.7	83.3	63.3	73.3	70.0	86.7	56.7
ਞ	73.3	70.0	73.3	86.7	83.3	83.3	63.3	80.0	60.0
ਟ	76.7	73.3	86.7	93.3	83.3	70.0	83.3	76.7	70.0
ਠ	83.3	86.7	86.7	70.0	83.3	86.7	70.0	90.0	60.0
ਡ	76.7	80.0	76.7	76.7	60.0	86.7	66.7	86.7	66.7
ਢ	73.3	86.7	73.3	73.3	66.7	73.3	60.0	76.7	66.7
ਣ	73.3	83.3	80.0	76.7	73.3	83.3	70.0	73.3	66.7
ਤ	73.3	76.7	73.3	70.0	76.7	73.3	73.3	70.0	70.0
ਥ	83.3	80.0	70.0	93.3	70.0	76.7	83.3	70.0	80.0
ਦ	86.7	86.7	73.3	86.7	70.0	83.3	66.7	70.0	76.7
ਧ	90.0	90.0	76.7	76.7	86.7	70.0	80.0	76.7	63.3
ਨ	86.7	73.3	73.3	76.7	86.7	86.7	73.3	76.7	76.7
ਪ	83.3	90.0	86.7	83.3	93.3	86.7	60.0	86.7	56.7
ਫ	73.3	76.7	70.0	80.0	83.3	83.3	66.7	86.7	73.3
ਬ	83.3	90.0	70.0	83.3	76.7	80.0	63.3	73.3	76.7
ਭ	70.0	76.7	86.7	70.0	80.0	70.0	83.3	73.3	73.3
ਮ	83.3	86.7	93.3	86.7	73.3	83.3	70.0	80.0	60.0
ਯ	70.0	76.7	86.7	80.0	73.3	90.0	60.0	80.0	56.7
ਰ	80.0	86.7	76.7	90.0	80.0	83.3	66.7	70.0	63.3
ਲ	80.0	70.0	80.0	70.0	60.0	80.0	76.7	83.3	80.0
ਵ	73.3	83.3	83.3	70.0	83.3	93.3	83.3	73.3	66.7
ੜ	76.7	76.7	80.0	80.0	83.3	80.0	63.3	90.0	66.7

Table 5.16 provides the detail comparative results for experimental work on *Gurmukhi* aksharas as written by different writers. Results of 8,880 samples collected from all the writers are compiled in Table 5.16 which clearly reveals that there is a minor variation in these recognition results. This variation has been found mainly because of difference in the handwriting style of the writers. It is worth mentioning here that different categories of writers were included under this sample testing based on their frequency of writing *Gurmukhi* script. Hence, it was obvious that a significant difference may prevail in the clarity of handwriting between the one who uses *Gurmukhi* script more frequently than one who uses it rarely. However, the robustness of proposed methodology has been proved with promising recognition accuracy despite high degree of variation in handwriting samples.

Table 5.16: Writer wise recognition accuracy (in %) achieved for *Gurmukhi* aksharas

Writer ID	Recognition rate	Writer ID	Recognition rate	Writer ID	Recognition rate
Writer 1	82.8	Writer 11	79.4	Writer 21	79.7
Writer 2	74.7	Writer 12	80.4	Writer 22	82.8
Writer 3	78.0	Writer 13	81.4	Writer 23	78.4
Writer 4	76.7	Writer 14	73.6	Writer 24	67.9
Writer 5	76.0	Writer 15	72.6	Writer 25	83.1
Writer 6	79.7	Writer 16	78.7	Writer 26	81.8
Writer 7	82.4	Writer 17	80.7	Writer 27	80.7
Writer 8	81.1	Writer 18	82.1	Writer 28	73.3
Writer 9	77.7	Writer 19	71.6	Writer 29	80.1
Writer 10	70.6	Writer 20	78.0	Writer 30	69.6

Stability in recognition accuracy achieved in testing has been depicted in Figure 5.6. These results provide that a very low degree of variation exists in recognition results compiled for first 5, 10, 15, 20, 25 and 30 writers.

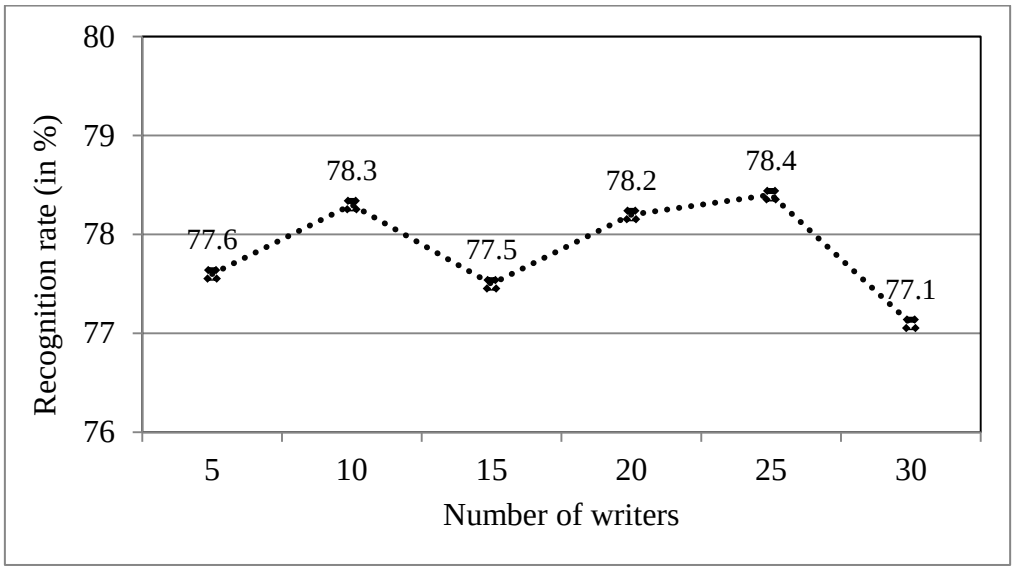


Figure 5.6: Stability of online handwritten *Gurmukhi* aksharas recognizer

Table 5.17 provides recognition results for aksharas formed with two nasal symbols /ੰ/ and /ੱ/. Results for *Gurmukhi* nasal /ੱ/ have not been presented in this table because this particular symbol is used between two consonants to form a word only. It has been observed that the results achieved on accuracy of aksharas with nasal symbol /ੱ/ are low for all the writers owing to the complexity of finding 64 normalized points in pre-processing for /ੱ/.

Table 5.17: Recognition rate for *Gurmukhi* characters with nasal symbol (/ੰ/ and /ਂ/)

	◌ੰ	◌ਂ		◌ੰ	◌ਂ		◌ੰ	◌ਂ
ਅ	81.2	72.6	ਝ	78.3	72.0	ਨ	78.3	71.1
ਸ	83.3	74.0	ਞ	74.5	64.9	ਪ	74.5	66.2
ਹ	77.1	68.3	ਟ	80.7	71.2	ਫ	80.7	73.5
ਕ	83.4	77.1	ਠ	84.4	74.7	ਬ	84.4	76.6
ਖ	80.5	70.7	ਡ	77.4	67.7	ਭ	77.4	70.7
ਗ	77.9	68.8	ਢ	73.1	65.3	ਮ	73.1	65.2
ਘ	81.9	73.3	ਣ	77.4	67.8	ਯ	77.4	70.1
ਙ	85.6	77.8	ਤ	74.3	65.8	ਰ	74.3	65.9
ਚ	76.7	68.8	ਥ	81.4	74.5	ਲ	81.4	72.0
ਛ	84.1	75.2	ਦ	77.4	69.2	ਵ	77.4	69.8
ਜ	83.1	73.6	ਧ	80.8	72.4	ੜ	80.8	73.7

The unit of *Gurmukhi* script that conveys a meaningful expression to combination of *Gurmukhi* consonants with vowel modifiers and nasals is the *Gurmukhi* word. For the purpose of testing the online handwritten *Gurmukhi* script recognizer, a sample of 900 *Gurmukhi* words has been collected from 30 writers. These writers were requested to write 10 *Gurmukhi* words of two characters, three characters and four characters each. Results obtained for two characters, three characters and four characters *Gurmukhi* words have been provided in Table 5.18. This table also includes word level and Unicode level recognition accuracy. Results presented in this table clearly reveal that Unicode level accuracy is higher than word level accuracy for all the two, three and four character *Gurmukhi* words. Results presented in this table reveal that a higher rate of recognition accuracy has been achieved for recognition of two character *Gurmukhi* words. Whereas when the complexity increases and difficult words with three and four characters are written by a writer, the accuracy declines.

Table 5.18: Recognition rate for two, three and four character *Gurmukhi* words

Writer ID	**Two character words**		**Three character words**		**Four character words**	
	Word level	Unicode level	Word level	Unicode level	Word level	Unicode level
Writer 1	73.3	77.5	73.3	79.1	76.6	79.0
Writer 2	83.3	87.5	73.3	79.8	60.0	66.5
Writer 3	66.7	71.8	66.6	73.5	63.3	68.9
Writer 4	66.7	71.8	70.0	73.8	76.6	82.2
Writer 5	76.7	81.0	80.0	82.5	63.3	68.8
Writer 6	76.7	82.4	73.3	75.5	53.3	60.2

Writer 7	73.3	77.7	70.0	73.4	66.6	73.5
Writer 8	66.7	70.4	63.3	66.5	63.3	68.0
Writer 9	73.3	75.9	66.6	73.3	63.3	68.5
Writer 10	73.3	76.0	66.6	70.5	73.3	76.1
Writer 11	80.0	82.3	66.6	68.7	63.3	66.9
Writer 12	70.0	72.9	73.3	76.0	70.0	75.3
Writer 13	76.7	79.0	70.0	76.9	73.3	78.7
Writer 14	56.7	58.9	66.6	71.7	60.0	64.9
Writer 15	70.0	75.2	63.3	69.9	70.0	72.5
Writer 16	80.0	84.2	76.6	82.5	60.0	65.5
Writer 17	73.3	77.9	70.0	76.2	70.0	75.4
Writer 18	70.0	72.9	63.3	65.4	63.3	68.6
Writer 19	76.7	79.2	70.0	74.5	60.0	64.7
Writer 20	80.0	83.2	76.6	83.5	73.3	80.0
Writer 21	73.3	76.6	70.0	73.1	76.6	81.0
Writer 22	83.3	87.8	70.0	73.9	73.3	79.1
Writer 23	76.7	81.6	70.0	72.6	70.0	74.4
Writer 24	73.3	76.7	76.6	82.9	60.0	68.1
Writer 25	76.7	78.9	76.6	80.2	76.6	83.5
Writer 26	66.7	71.3	76.6	82.2	56.6	63.0
Writer 27	70.0	72.9	76.6	83.5	76.6	79.1
Writer 28	76.7	81.6	80.0	85.6	60.0	64.1
Writer 29	73.3	78.0	60.0	66.9	73.3	79.1
Writer 30	80.0	84.6	63.3	70.1	70.0	76.9

5.7 Chapter summary

This chapter describes post-processing methodology proposed in this research work. Rule book formulated for recognition of online handwritten *Gurmukhi* script has been provided here. Post-segmentation of stroke sets, sequencing of stroke sets, and merging the stroke sets included in the methodology of post-processing have been explained here in detail. In post-segmentation of stroke sets procedure of categorizing the association sets has been explained. These classified sets are further sorted in the next step, *i.e.*, sequencing of stroke sets. Finally, a Unicode of *Gurmukhi* script is computed from the weights allotted to association sets.

Disambiguation of some critical problems faced in the online handwritten *Gurmukhi* script has been discussed here. In this chapter, detail demonstration of proposed recognition engine has been provided separately for character, numeral, akshara and word

formation. Proposed methodology for recognition of online handwritten *Gurmukhi* script has been tested on selected sample collected from different category of writers. Recognition results for such testing have been discussed here separately for *Gurmukhi* characters, numerals, aksharas and words.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

Handwriting recognition has been admitted to be highly promising area of research because of its wide range of applications. Although research in this area is alive for more than four decades yet it is still admitted to be a challenging area. Realizing the potential applications of online handwritten recognition systems, world-wide researchers have endeavored to refine existing methods. Initially most of the research in this direction was for designing a recognition system for English language. However, during last two decades researchers have achieved appreciable recognition results for other popular languages that include Chinese, Japanese, Arabic *etc*. Among Indian scripts most of research has been proposed for Devanagari script because it is national language of India and spoken by a large population of India. The main goal of this research study was to develop an efficient system for recognition of online handwritten *Gurmukhi* script. Well-structured methodology proposed for achieving this objective has offered high recognition accuracy.

This research study has proposed algorithms for association of online handwritten strokes, post-segmentation, sequencing of strokes sets, and merging the stroke sets that have been applied in various phases of online handwritten *Gurmukhi* script recognition system. Brief introduction of *Gurmukhi* script, standard way of writing it and difficulties in recognition of *Gurmukhi* script have been discussed in Chapter 1. Motivation behind the proposed research that worked as a driving force to pursue research on recognition of online handwritten *Gurmukhi* script has also been described in this chapter. This chapter has also briefly highlighted contribution of present research work. A comprehensive review of literature on various stages involved in online handwriting recognition process has been presented in Chapter 2. Chapter 3 has explained the methodology used and system designed for carrying out this research work. In Chapter 4, a modified algorithm of pre-processing for normalization of horizontal and vertical bars has been proposed. This chapter has also explained the association rules proposed in this work. Recognition and post-processing

phases for *Gurmukhi* script and detailed experimental results are discusses at length in Chapter 5.

6.1 Brief Contribution of the proposed research

Handwriting recognition process uses sequential standardized steps to give the output. Various steps involved in handwriting recognition process includes: data collection, pre-processing, feature extraction, recognition and post-processing. The broad steps of proposed method are similar to the sequential steps used in traditional architecture of handwriting recognition process. However, in present study some novel techniques have been introduced within that broad framework. Forthcoming sections briefly discuss the contribution of present research work.

6.1.1 Data collection, pre-processing and feature extraction

Creating a reliable and authentic database had been one of the objectives of this research work. Under this study, first objective was to create an annotated set of 30,000 (including 2000 unique words) words of *Gurmukhi* script. Hence, the desired dataset was collected from 144 different users and special attributes like number of strokes, digitized coordinates and their start/ending index were recorded. Pre-processing stage has been used for removing noisy artifacts from handwriting and also helped in correcting imperfections. Steps in pre-processing included preliminary analysis of stroke, association of strokes, removal of duplicate points, size normalization, centering and resampling. Some algorithms have been proposed for pre-processing stage. In this research work a new methodology of finding association between the strokes has been developed which is further bifurcated under three sub steps, *viz.* schema for stroke association, sets of associated strokes and reduction of associated sets.

Normalization of handwritten data includes removal of variations that occur in position, size of strokes and separation between data points in the stroke. Under this research work, normalization of stroke has been proposed for creating symmetry of slanting or tilted strokes. The input handwritten character is normalized to 300×300 pixels and placed in the center of the fixed frame. Points generated after pre-processing phase are used as features for

recognition. Slanted strokes have been managed with the help of association rule. A methodology has been proposed for removal of noise and partial strokes present in online handwriting.

6.1.2 Recognition using LibSVM classifier

Classification of *Gurmukhi* strokes collected from raw data has been performed using LibSVM classifier. For LibSVM classification, Radial Basis Function (RBF) kernel has been used. This kernel nonlinearly maps samples into a higher dimensional space. A comparative analysis has been conducted for stroke recognition for 96 classes. In this research work, three RBF engines have been designed *viz*. Engine A (for upper zone), Engine B (for middle zone) and Engine C (for lower zone). Cross-validation has been used at 2-fold, 3-fold, 4-fold and 5-fold for these three engines *viz.* Engine A, Engine B, and Engine C. In engine A, a total of 826 samples were used for training and highest accuracy of 98.7% has been achieved at 5-fold cross-validation. For Engine B, a sample of 10,740 *Gurmukhi* strokes has been used to train the engine and the highest accuracy of 96.7% has been achieved at 5-fold level. A total of 317 collected samples have been used to train Engine C. In this case, an accuracy of 100.0% has been achieved at 2-fold, 3-fold, 4-fold and 5-fold level. Application of SVM has achieved excellent recognition results for various pattern recognition problems. Under current study, 95.6% recognition accuracy for *Gurmukhi* characters has been achieved.

6.1.3 Post-processing

To remove the problems of classification/ segmentation and improve accuracy of results, post-processing techniques are applied in the process of online handwritten character recognition. In this research work, real time post-processing has been implemented. For every new stroke, post-processor forms the new character instead of forming the character at the end of all input strokes. New algorithms for post-segmentation of stroke sets, sequencing of stroke sets and merging the stroke sets for post-processing in handwritten *Gurmukhi* script recognition have been presented. In case of post-segmentation of stroke sets, separation of vowels modifiers which overlap or associate with consonants has been presented. Sequencing of stroke sets portrays a problem associated with handwriting style of a writer where he may

write some strokes at end which originally takes the position in between the *Gurmukhi* word. In this part of post-processing phase, we have proposed a systematic way where sequencing of consonant/vowel modifiers is corrected along *x*-axis. Hence, in this step classified sets of A^{C} and A^{V} obtained from post-segmentation are sorted as A^{S} based on X_{min} and X_{max} respectively.

Merging the stroke sets has been proposed as final step in the post-processing phase. In this step, a systematic procedure has been adopted to merge initial stroke and sub-strokes required to form a character. Merging of stroke sets resolves the problem where sub-strokes are separated from initial stroke due to sub-stroke falling in different association or post-segmentation of stroke set. Hence, merging the stroke sets presented in this research work has provided a new way to associate these partial strokes so as to formulate a complete character.

6.1.4 Experimental results

The present approach provided real time post-processing where with input of every new stroke post-processor recognizes the character formed instead of recognizing the character at the end of all input strokes. In *Gurmukhi* script, there are few characters which are quite similar in their writing style. This problem created confusion in recognizing those characters. However, the robustness of proposed recognition engine has been proved with achieving 95.6% recognition accuracy at character level. Some selected *Gurmukhi* characters that are not misclassified with any other *Gurmukhi* character, *viz.* /ੳ/, /ਕ/, /ਙ/, /ਛ/, /ਠ/, /ਬ/, and /ਮ/ have achieved highest recognition accuracy. Testing results have also revealed that 21 *Gurmukhi* characters /ੳ/, /ਅ/, /ਹ/, /ਕ/, /ਘ/, /ਙ/, /ਛ/, /ਵ/, /ਟ/, /ਠ/, /ਡ/, /ਢ/, /ਤ/, /ਦ/, /ਪ/, /ਫ/, /ਬ/, /ਭ/, /ਮ/, /ਰ/ and /ੜ/ are those which may be written with single stroke or multiple strokes, but writers have preferred to write these characters with single stroke. Here, characters written with single stroke achieve higher recognition accuracy than characters written with multiple strokes. Finally, /ਥ/, /ਟ/, /ਨ/, /ਜ/, /ੜ/, /ਘ/, /ੳ/, /ਕ/, /ਙ/, /ਛ/, /ਠ/, /ਪ/, /ਬ/, /ਮ/ and /ਸ/ are the *Gurmukhi* characters that have achieved recognition accuracy in the range of 96%-100%. This study also worked on recognition of *Gurmukhi* numerals. An overall recognition

accuracy of 96.8% has been achieved for *Gurmukhi* numerals. The present study also provides for use of valid combination of two different classes of *Gurmukhi* script for akshara formation. For the purpose of testing, 8,880 aksharas were collected from 30 different writers. Overall, 77.9% recognition accuracy has been achieved for akshara formation. It is apparent here that promising accuracy of 80.1%, 79.5% and 79.1% has been achieved in case of aksharas formed using three vowel modifier, *viz.* /◌ਾ/, /ਿ◌/, and /◌ੀ/, respectively. Results of akshara recognition highlight that the recognition accuracy is quite low whenever any *Gurmukhi* akshara is formed with the combination of *Gurmukhi* character with /◌ੈ/ and /◌ੌ/. Finally, a sample of 900 *Gurmukhi* words has been collected from 30 writers. These writers were asked to write 10 *Gurmukhi* words of two characters, three characters and four characters each. Overall, 70.5% and 75.2% recognition accuracy has been achieved at word level and Unicode level, respectively. Result presented in previous chapter reveal that high rate of accuracy has been achieved for recognition of two character *Gurmukhi* words. However, with increase in complexity for difficult words with three and four characters the accuracy declines minutely.

6.2 Future scope

The proposed objectives of this research study have been met very well. As stated in section 1.2 of chapter 1, an annotated data set of 30,000 words of *Gurmukhi* script (including 2000 unique words) for training and testing the OHWR engine has been created. Finally, using this dataset for testing, a writer independent system has been developed for recognition of online handwritten *Gurmukhi* characters, numerals and words. In the present study, most of constraints regarding complex nature of *Gurmukhi* script that include variation in handwriting style and slanted way of writing a word have been considered. Even in the presence of these constraints promising recognition accuracy has been achieved. However, the results achieved in the present study motivate to extend the present work with HMM and neural network as recognition methods. There is always a possibility of refining the existing recognition accuracy when sample size is efficiently increased. Hence, even applying the same methodology, higher recognition accuracy can be achieved by including large sample

size or more typical *Gurmukhi* words. To sum up, present research can be extended in following directions:

- The complexities of pre-processing methodology can further be reduced in order to reduce the recognition time of a stroke.
- Writer adaptation can be incorporated.
- Checking of spellings to correct errors at word level can be introduced.
- Features extracted always add to achieving higher recognition accuracy at post-processing stage. Hence, higher accuracy can be achieved by identifying some efficient features of strokes in *Gurmukhi* script.
- There is possibility of fusing incremental learning in order to avoid retraining the entire system when a new stroke or character is introduced.
- *Gurmukhi* script comprises of many structural similarities with other Asian scripts such as Devanagari, Gujrati and Bangla. Therefore, the recognition method used in present study can further be used for such scripts using their databases.
- Recognition of strokes with a low number of raw points poses some confusion in recognition process. So, a separate classifier can be used for such strokes.

REFERENCES

[1] Agrawal M., Bhaskarabhatla A.S., and Madhvanath S., "Data collection for online handwriting corpus creation in Indic scripts," in *Proc. International Conference on Speech and Language Technologies*, vol. 1, pp. 149-154, 2004.

[2] Ahmed P. and Suen C.Y., "Computer recognition of totally unconstrained handwritten zip codes," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 1, no. 1, pp. 1-15, 1987.

[3] Ait-Mohand K., Paquet T., and Ragot N., "Combining Structure and Parameter Adaptation of HMMs for Printed Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 9, pp.1716-1732, Sept. 2014.

[4] Alaei A., Nagabhushan, P. and Pal U., "A new two-stage scheme for the recognition of Persian handwritten characters," in *Proc. 12th International Conference on Frontiers in Handwriting Recognition*, pp. 130-135, 2010.

[5] Alahari K., Putrevu S.L., and Jawahar C.V., "Discriminant substrokes for online handwriting recognition," in *Proc. International Conference on Document Analysis and Recognition*, vol. 1, pp. 499-505, 2005.

[6] Almuallim H. and Yamaguchi S., "A Method of Recognition of Arabic Cursive Handwriting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 715-722, 1987.

[7] Al-Taani A.T., "An efficient feature extraction algorithm for the recognition of handwritten Arabic digits," *International Journal of Computational Intelligence*, vol. 2, no. 2, pp. 107-111, 2005.

[8] Aparna K. H., Subramanian V., Kasirajan M., Prakash G. V., Chakravarthy V. S., and Madhvanath S., "Online handwriting recognition for Tamil," in *Proc. International Workshop on Frontiers in Handwriting Recognition*, pp. 438-443, 2004.

[9] Babu V., Prasanth L., Sharma R., Rao G.V., and Bharath A., "HMM-based Online Handwriting Recognition System for Telugu Symbols," in *Proc. the Ninth International Conference on Document Analysis and Recognition*, vol. 1, pp. 63-67, 2007.

[10] Bahlmann C., Haasdonk B., and Burkhardt H., "Online handwriting recognition with support vector machines - a kernel approach," in *Proc. Eighth International Workshop on Frontiers in Handwriting Recognition*, pp. 49-54, 2002.

[11] Bahri H., "Teach Yourself Panjabi," Publication Bureau, Punjabi University Patiala, 2011.

[12] Balasubramanian A., "Document Annotation and Retrieval Systems," MS Thesis, International Institute of Information Technology, Hyderabad, India, 2006.

[13] Beigi H., "Pre-processing and dynamics of online handwriting data, feature extraction and recognition," in *Proc. 5th International Workshop on Frontiers in Handwriting Recognition*, pp. 255-258, 1996.

[14] Beigi H.S.M., Nathan K., Clary G.J., and Subrahmonia J., "Challenges of Handwriting Recognition in Farsi, Arabic and other Languages with Similar Writing Styles an on-line Digit Recognizer," in *Proc. Second Annual Conference on Technological Advancements in Developing Countries*, pp. 1-9, 1994b.

[15] Beigi H.S.M., Nathan K., Clary G.J., and Subrahmonia J., "Size normalization in on-line unconstrained handwriting recognition," in *Proc. IEEE International Conference Image Processing*, pp.169-173, 1994a.

[16] Bengio Y. and LeCun Y., "Word normalization for on-line handwritten word recognition," in *Proc. International Conference on Pattern Recognition*, vol. 2, pp. 409-413, 1994.

[17] Bharath A., Vijayasenan D., and Madhvanath S., "An approach to identify unique styles in online handwriting recognition," in *Proc. of International Conference on Document Analysis and Recognition*, vol. 2, pp. 775–778, 2005.

[18] Bhaskarabhatla A.S., Madhvanath S., Kumar M., Balasubramanian A., and Jawahar C.V., "Representation and annotation of online handwritten data," in *Proc. Ninth International Workshop on Frontiers in Handwriting Recognition*, pp.136-141, 2004.

[19] Bhattacharya U. and Chaudhuri, B.B., "Handwritten numeral databases of Indian scripts and multistage recognition of mixed numerals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 444-457, 2009.

[20] Bhattacharya U., Gupta B.K., and Parui S.K., "Direction Code Based Features for Recognition of Online Handwritten Characters of Bangla," in *Proc. of Ninth International Conference on Document Analysis and Recognition*, vol.1, pp. 58-62, 2007.

[21] Bhattacharya U., Nigam A., Rawat Y.S., and Parui S.K., "An Analytic Scheme for Online Handwritten Bangla Cursive Word Recognition," in *Proc. of 11th International Conference on Frontiers in Handwriting Recognition*, pp. 320-325, 2008.

[22] Bhattacharya U., Shridhar M., Parui S.K., Sen P.K. and Chaudhuri B.B., "Offline recognition of handwritten Bangla characters: an efficient two-stage approach," *Pattern Analysis and Applications*, vol. 15, no. 4, pp. 445-458, 2012.

[23] Biswas C., Bhattacharya U., and Parui S.K., "HMM Based Online Handwritten Bangla Character Recognition Using Dirichlet Distributions," in *Proc.of International Conference on Frontiers in Handwriting Recognition*, pp.600-605, 2012.

[24] Blumenstein M. and Verma B., "A new segmentation algorithm for handwritten word recognition," in *Proc. the International Joint Conference on Neural Networks*, pp. 878-882, 1999.

[25] Bozinovic R.M. and Shrihari S.N., "Off-line cursive script word recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, pp. 68–83, 1989.

[26] Brakensiek A., Kosmala A., and Rigoll G., "Comparing normalization and adaptation techniques for on-line handwriting recognition," in *Proc. International Conference on Pattern Recognition*, vol. 3, pp. 73-76, 2002.

[27] Brault J. and Plamondon R., "Segmenting handwritten signatures at their perceptually important points," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 953-957, 1993

[28] Britto A.S., Sabourin R., Bortolozzi F., and Suen C.Y., "The recognition of handwritten numeral strings using a two-stage HMM-based method," *International Journal on Document Analysis and Recognition*, vol. 5, no. 2/3, pp. 102-117, 2003.

[29] Brown M.K. and Ganpathy S., "Preprocessing techniques for cursive script word recognition," *Pattern recognition*, vol. 16, no. 5, pp. 447-458, 1983.

[30] Burr D.J., "Designing a Handwriting Reader," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 5, pp. 554-559, 1983.

[31] Byan H. and Lee S.W., "A survey on pattern recognition applications of support vector machines," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 17, no. 3, pp. 459-486, 2003.

[32] Caillault E. and Viard-Gaudin C., "Using Segmentation Constraints in an Implicit Segmentation Scheme for On-line Word Recognition," in *Proc. 10th International Workshop on Frontiers in Handwriting Recognition*, pp. 607-612, 2006.

[33] Carbonnel S. and Anquetil E., "Lexicon organization and string edit distance learning for lexical post-processing in handwriting recognition," in *Proc. the 9th International Workshop on Frontiers in Handwriting Recognition*, pp. 462-467, 2004.

[34] Casey R.G. and Lecolinet E., "A survey of methods and strategies in character segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 690-706, 1996.

[35] Chan K.F. and Yeung D.Y., "Recognizing on-line handwritten alphanumeric characters through flexible structural matching," *Pattern Recognition*, vol. 32, no. 7, pp. 1099-1114, 1999.

[36] Cheriet M., Kharma N., Liu C.L., and Suen C.Y., "Character Recognition Systems: A Guide for Students and Practitioners," Wiley Inter-Science, 2007

[37] Cho S.J. and Kim J., "Bayesian network modeling of strokes and their relationships for on-line handwriting recognition," in *Proc. International Conference on Document Analysis and Recognition*, pp. 86-90, 2001.

[38] Connell S.D., Sinha R.M.K., and Jain A.K., "Recognition of unconstrained online Devanagari characters," in *Proc. of International Conference on Pattern Recognition*, vol. 2, pp. 368-371, 2000.

[39] Daifallah K., Zarka N., and Jamous H., "Recognition-Based Segmentation Algorithm for On-Line Arabic Handwriting," in *Proc. 10th International Conference on Document Analysis and Recognition*, pp. 886-890, 2009.

[40] Devijver P.A. and Kittler J., Pattern Recognition – A Statistical Approach, Prentice Hall, 1982.

[41] Dongre V.J. and Mankar V.H., "A Review of Research on Devnagari Character Recognition," *International Journal of Computer Application*, vol. 12, no.2, pp 8-15, 2010.

[42] Duda R.O., Hart P.E., and Stork D.G., Pattern Classification, New York: John Wiley and Sons, 2001.

[43] Duerr B., Haettich W., Tropf H., and Winkler G., "A Combination of Statistical and Syntactical Pattern Recognition Applied to Classification of Unconstrained Handwritten Numerals," *Pattern Recognition*, vol. 12, no. 3, pp. 189-199, 1980.

[44] Duneau L. and Dorizzi B., "Online cursive script recognition: a system that adapts to an unknown user," in *Proc. International Conference on Pattern Recognition*, vol. 2, pp. 24-28, 1994.

[45] Dunn C.E., and Wang P.S P., "Character Segmentation Techniques for Handwritten Text – A Survey," in *Proc. the 11th International Conference on Pattern Recognition*, pp. 577-580, 1992.

[46] Elnagar A. and Alhajj R., "Segmentation of Connected Handwritten Numeral Strings," *Pattern Recognition*, vol. 36, no. 3, pp. 625-634, 2003.

[47] Favata J. and S. N. Srihari, "Recognition of General Handwritten Words Using a Hypothesis Generation and Reduction Methodology," *United States Postal Service Advanced Technology Conference*, pp. 237-251, 1992.

[48] Fox A.S. and Tapper C.C., "On-line external word segmentation for handwriting recognition," in *Proc. 3rd International Symposium on Handwriting Computer Applications*, pp. 53-55, 1987.

[49] Fu H.C., Chang H.-Y., Xu Y.Y., and Pao H.-T., "User adaptive handwriting recognition by self-growing probabilistic decision-based neural networks," *IEEE Transactions on Neural Networks*, vol. 11, pp. 1373–1384, November 2000.

[50] Fu K.S., "Syntactic Pattern Recognition and Applications," New Jersey: Prentice-Hall, 1982.

[51] Gader P.D. and Khabou M.A., "Automatic Feature Generation for Handwritten Digit Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 12, pp. 1256-1261, 1996.

[52] Gader P.D., Forester B., Ganzberger M., Gillies A., Mitchell B., Whalen M., and Yocum T., "Recognition of Handwritten Digits using Template and Model Matching," *Pattern Recognition*, vol. 24, no. 5, pp. 421-431, 1991.

[53] Gaillat G., "An on-line recognizer with learning capabilities," in *Proc. 2nd International Joint Conference on Pattern Recognition*, pp. 305-306, 1974.

[54] Gang L., Verma B., and Kulkarni S., "Experimental Analysis of Neural Network Based Feature Extractors for Cursive Handwriting Recognition," in *Proc. of IEEE World Congress on Computational Intelligence*, pp. 2837-2841, 2002.

[55] Garain, U. and Chaudhuri B.B., "Segmentation of touching characters in printed Devnagari and Bangla scripts using fuzzy multifactorial analysis," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol.32, no.4, pp. 449-459, Nov. 2002.

[56] Govindaraju V., Kim G., and Srihari S.N., "Paradigms in handwriting recognition," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, pp. 1498-1503, 1997.

[57] Guerfali W. and Plamondon R., "Normalizing and restoring online handwriting," *Pattern Recognition*, vol. 26, no. 3, pp. 419-431, 1993.

[58] Guillevic D. and Suen C.Y., "Cursive script recognition- a sentence level recognition scheme," in *Proc. of International Workshop on Frontiers in Handwriting Recognition*, pp. 216-223, 1994.

[59] Ha J., Haralick R.M., and Phillips I.T., "Understanding Mathematical Expressions from Document Images," in *Proc. of the Third International Conference on Document Analysis and Recognition*, pp. 956-959, 1995.

[60] Helmers M. and Bunke H., "Generation and use of synthetic training data in cursive handwriting recognition," in *Proc. the 1st Iberian Conference on Pattern Recognition and Image Analysis*, pp. 336-345, 2003.

[61] Hollerbach J.M., "An oscillation theory of handwriting," *Biological Cybernetics*, vol. 39, no. 2, pp. 139-156, January 1981.

[62] Hu J., Lim S.G., and Brown M.K., "Writer Independent On-Line Handwriting Recognition Using an HMM Approach," *Pattern Recognition*, vol. 33, no. 1, pp. 133-147, 2000.

[63] Hu J., Rosenthal A.S., and Brown M.K., "Combining high level features with Sequential local features for online handwriting recognition," in *Proc. 9th International Conference Image Analysis and Processing*, vol. 1311, pp. 647-657, 1997.

[64] Huang B.Q., Zhang Y.B., and Kechadi M.T., "Preprocessing techniques for online handwriting recognition," in *Proc. IEEE 7th International Conference on Intelligent System Design and Application*, pp. 793-798, 2007.

[65] Hull J.J., Commike A., and Ho T.K., "Multiple Algorithms for Handwritten Character Recognition," in *Proc. International Workshop on the Frontiers in Handwriting Recognition*, pp. 117-129, 1990.

[66] Hussain A.B.S., Toussaint G.T., and Donaldson R.W., "Results Obtained Using a Simple Character Recognition Procedure on Munson's Hand printed Data," *IEEE Transactions on Computers*, vol. C-21, no. 2, pp. 201-205, 1972.

[67] Impedovo S., Ottaviano L., and Occhinegro S., "Optical Character Recognition - A Survey," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 5, no. 1, pp. 1-24, 1991.

[68] Jaeger S., Liu C. L. and Nakagawa M., "The state of the art in Japanese on-line handwriting recognition compared to techniques in Western handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 6, no. 2, pp. 75-88, 2003.

[69] Jaeger S., Manke S., Reichert J., and Waibel A., "Online handwriting recognition: the Npen++ Recognizer," *International Journal of Document Analysis and Recognition*, vol. 3, no. 3, pp. 169-180, 2001.

[70] Jain A. K., Duin R.P.W., and Mao J., "Statistical pattern recognition: a review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1.pp. 4-37, 2000.

[71] Javed S.T. and Hussain S. "Improving Nastalique Specific Pre-Recognition Process for Urdu OCR," in *Proc. of 13th IEEE International Multitopic Conference*, pp. 1-6, 2009.

[72] Jhajj P. and Sharma D., "Recognition of Isolated Handwritten Characters in Gurmukhi script," *International Journal of Computer Applications*, vol. 4, no. 8, pp. 9-17, 2010.

[73] Jindal M.K., Sharma R.K., and Lehal G.S., "A Study of Touching Characters in Degraded Gurmukhi Text," in *Proc. of World Academy of Science, Engineering and Technology (PWASET)*, vol. 4, pp. 121-124, 2005.

[74] Jindal M.K., Sharma R.K., and Lehal G.S., "Segmentation of Horizontally Overlapping Lines in Printed Indian Scripts," *International Journal of Computational Intelligence Research, Research India Publications*, vol. 3, no. 4, pp. 277-286, 2007.

[75] Jinhai C, and Liu Z.Q., "Integration of structural and statistical information for unconstrained handwritten numeral recognition," *IEEE Transaction Pattern Analysis Machine Intelligence*, vol. 21, no. 3, pp. 263–270, 1999.

[76] Jose O.Jr., Carvalho J.M., Freitas C., and Sabourin R., "Feature sets evaluation for handwritten word recognition," in *Proc. 8th International Workshop on Frontiers in Handwriting Recognition*, pp. 446-551, 2002.

[77] Joshi N., Sita G., Ramakrishnan A.G., Deepu V., and Madhvanath S., "Machine recognition of online handwritten Devanagari characters," in *Proc. International Conference of Document Analysis and Recognition*, pp. 1156-1160, 2005.

[78] Jung K. and Kim H.J., "Online recognition of cursive Korean characters using graph representation," *Pattern Recognition*, vol. 33, pp. 399-412, 2000.

[79] Kavallieratou E., Fakotakis N., and Kokkinakis G., "An unconstrained handwriting recognition system," *International Journal of Document Analysis and Recognition*, vol. 4, no. 4, pp. 226-242, 2002.

[80] Keeler J.D., Rumelhart D.E., and Leow W.K., "Integrated segmentation and recognition of hand-printed numerals," in *Proc. the conference on Advances in Neural Information Processing Systems*, vol. 3, pp. 557-563, 1991.

[81] Kim G. and Govindaraju V., "A lexicon driven approach to handwritten word recognition for real-time applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 366-379, 1997.

[82] Kim H.Y. and Kim J.H., "Hierarchical random graph representation of handwritten character and its application to Hangul recognition," *Pattern Recognition*, vol. 34, no. 2, pp. 187-201, 2001.

[83] Kim I., Jung K., Park S.H., and Kim H.J., "Support Vector Machine-based Text Detection in Digital Video," *Pattern Recognition*, vol. 34, no. 2, pp. 527-529, 2001.

[84] Kim J., "Baseline drift correction of handwritten text," Technical Report, vol. 25, no. 10, IBM Tech. Disclosure Bull., 1983.

[85] Kobayashi M., Masaki S., Miyamoto O., Nakagawa Y., Komiya Y., and Matsumoto T., RAV (reparametrized angle variations) algorithm for online handwriting Recognition," *International Journal of Document Analysis and Recognition*, vol. 3, no. 1, 181-191, 2001.

[86] Koerich A. L., Sabourin R., and Suen C.Y., "Lexicon-driven HMM decoding for large vocabulary handwriting recognition with multiple character models," *International Journal of Document Analysis and Recognition*, vol. 6, no. 2, pp. 126-144, 2003.

[87] Koerich A.L., Sabourin R., and Suen C.Y., "Recognition and Verification of Unconstrained Handwritten Words," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1509-1522, 2005.

[88] Kubatur S., Sid-Ahmed M., and Ahmadi M., "A neural network approach to online Devanagari handwritten character recognition," in *Proc. of International Conference on High Performance Computing and Simulation (HPCS)*, pp. 209-214, 2012.

[89] Kumar A., Balasubramanian A., Namboodiri A., and Jawahar C.V., "Model-based annotation of online handwritten datasets," in *Proc. International Workshop on Frontiers in Handwriting Recognition*, pp. 9-14, 2006.

[90] Lajish V.L. and Kopparapu S.K., "Fuzzy Directional Features for unconstrained on-line Devanagari handwriting recognition," in *Proc. National Conference on Communications*, pp. 1-5, 2010.

[91] Lecolinet E. and Crettez J.P., "A grapheme-based segmentation technique for cursive script recognition," in *Proc. International Conference on Document Analysis and Recognition*, pp. 740-748, 1991.

[92] Lee S., "Off-line recognition of totally unconstrained handwritten numerals using multilayer cluster neural network," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 6, pp. 648-652, 1996.

[93] Lehal G.S. and Singh C., "A Gurmukhi Script Recognition System," in *Proc. 15th International Conference on Pattern Recognition*, vol. 2, pp. 557-560, 2000.

[94] Lehal G.S. and Singh C., "A Technique for Segmentation of Gurmukhi Text," *in Proc. 9th International Conference Computer Analysis of Images and Patterns*, LNCS, vol. 2124, pp. 191-200, 2001.

[95] Liu C.L., Jaeger S. and Nakagawa M., "Online recognition of Chinese characters: The state-of-the-art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 198-213, 2004.

[96] Liu C.L., Nakashima K., Sako H., and Fujisawa H., "Handwritten digit recognition using state-of-the-art techniques," in *Proc. Eighth International Workshop on Frontiers in Handwriting Recognition*, pp. 320-325, 2002.

[97] Liu J., Cham W.K., and Chang M.M.Y., "Online Chinese character recognition using attributed relational graph matching," *IEE Proc. on Vision, Image and Signal Processing*, vol. 143, no. 2, pp. 125-131, 1996.

[98] Lu Y. and Shridhar M., "Character segmentation in handwritten words – an overview," *Pattern Recognition*, vol. 29, pp. 77-96, 1996.

[99] Madhvanath S., Kim G., and Govindaraju V., "Chain code contour processing for handwritten word recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 9, pp. 928-932, 1999.

[100] Madhvanath S., Vijayasenan D., and Kadiresan T.M., "LipiTk: a generic toolkit for online handwriting recognition," in *Proc. International Workshop on Frontiers in Handwriting Recognition*, pp. 349-354, 2006.

[101] Maier M., "Separating Characters in Scripted Documents," in *Proc. International Conference on Pattern Recognition*, pp. 1056-1058, 1986.

[102] Malaviya A., Leja C., and Peters L., "Multi-script handwriting recognition with FOHDEL," in *Proc. Biennial Conference of the North American*, pp.147-151, 1996.

[103] Malaviya A., Peters L., and Camposano R., "A Fuzzy Online Handwriting Recognition System: FOHRES," in *Proc. Second International Conference on Fuzzy Theory and Technology*, pp. 1-15, 1993.

[104] Malik S. and Khan S.A., "Urdu Online Handwriting Recognition," in *Proc. IEEE 5th International Conference on Emerging Technologies*, pp. 27-31, 2005.

[105] Man G.M.T. and Poon J.C.H., "An enhanced approach to character recognition by Fourier descriptor," in *Proc. Communications on the Move*, vol. 2, pp. 558-562, 1992.

[106] Mandler E., "Advanced preprocessing technique for on-line script recognition of non-connected symbols," in *Proc. 3rd International Symposium on Handwriting Computer Application*, pp. 64-66, 1987.

[107] Martin G.L., Rashid M., and Pittman J.A., "Integrated Segmentation and Recognition through Exhaustive Scans or Learned Saccadic Jumps," in *Proc. Advances in Pattern Recognition Systems Using Neural Network Technologies*, vol. 7, pp. 187-203, 1993.

[108] Marwala T., Unathi Mahola U., and Chakraverty S., "Fault Classification in Cylinders Using Multilayer Perceptrons Support Vector Machines and Guassian Mixture Models," *Computer Assisted Mechanics and Engineering Sciences*, vol. 14, no. 2, pp. 1-10, 2007.

[109] Matan O., Burges J.C., Cun Y.L., and Denker J.S., "Multi-Digit Recognition Using A Space Displacement Neural Network," in *Proc. the conference on Advances in Neural Information Processing Systems*, vol. 4, pp. 488-495, 1992.

[110] Morasso P. and Ivaldi F.A.M., "Trajectory formation and handwriting: A computational model," *Biological Cybernetics*, vol. 45, no. 2, pp. 131-142, 1982.

[111] Nagra J.S., "Panjabi Made Easy: Book 1," UK: Nagra Publication, March 1988.

[112] Nagy G., "Twenty years of document image analysis in pattern analysis and machine intelligence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 38-62, 2000.

[113] Nakagawa M. and Matsumoto K., "Collection of on-line handwritten Japanese character pattern databases and their analysis," *International Journal on Document Analysis and Recognition*, vol. 7, no. 1, pp. 69-81, 2004.

[114] Namboodiri A. M., Narayanan P. J. and Jawahar C. V., "On using classical poetry structure for Indian language post-processing," in *Proc. International Conference on Document Analysis and Recognition*, pp. 1238-1242, 2007.

[115] Namboodiri A.M., "On-line Handwritten Document Understanding," Ph. D. Thesis, Department of Computer Science, Michigan State University, 2004.

[116] Nataneli G. and Faloutsos P., "Robust classification of strokes with SVM and grouping," in *Proc. 3rd International Symposium of Advances in Visual Computing*, LNCS, vol. 4841, pp. 76-87, 2007.

[117] Negi, A., Swaroop K.S., and Agarwal A., "A correspondence based approach to segmentation of cursive words," in *Proc. of International Conference on Document Analysis and Recognition*, pp. 1034-1037, 1995.

[118] Nicchiotti G. and Scagliola C., "A simple and effective cursive word segmentation method," in *Proc. the 7th International Workshop on Frontiers of Handwriting Recognition*, pp. 499-504, 2000.

[119] Okamoto M. and Yamamoto K., "On-line handwritten character recognition method using directional features and clockwise/counterclockwise direction-change features," in *Proc. International Conference on Document Analysis and Recognition*, pp. 491-494, 1999.

[120] Pal U., Sharma N., Wakabayashi T., and Kimura F., "Off-Line Handwritten Character Recognition of Devnagari Script," in *Proc. of Ninth International Conference on Document Analysis and Recognition*, vol. 1, pp. 496-500, 2007b.

[121] Pal U., Wakabayashi T., and Kimura F., "A system for off-line Oriya handwritten character recognition using curvature feature," in *Proc. 10th International Conference on Information Technology*, pp. 227-229, 2007a.

[122] Pardeep J., Srinivasan E., and Himavathi S., "Diagonal based Feature Extraction for Handwritten Alphabets Recognition System using Neural Network," *International Journal of Computer Science and Information Technology*, vol. 3, no. 1, pp 27-38, 2011.

[123] Parui S.K., Guin K., Bhattacharya U., and Chaudhuri B.B., "Online handwritten Bangla character recognition using HMM," in *Proc. of 19th International Conference on Pattern Recognition*, pp.1-4, 2008.

[124] Pastor M., Toselli A., and Vidal E., "Projection Profile Based Algorithm for Slant Removal," in *Proc. Image Analysis and Recognition*, LNCS, vol. 3212, pp. 183-190, 2004.

[125] Pavlidis I., Singh R., and Papanikolopoulos N.P., An online handwritten note recognition method using shape metamorphosis," in *Proc. of 5th International Conference on Document Analysis and Recognition*, vol. 2, pp. 914-918, 1997.

[126] Perlovsky L.I., "Conundrum of Combinatorial Complexity," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 20, no. 6, pp. 666-670, 1998.

[127] Pitrelli J.F. and Perrone M.P., "Confidence modeling for verification post-processing for handwriting recognition," in *Proc. Eighth International Workshop on Frontiers in Handwriting Recognition*, pp.30-35, 2002.

[128] Plamondon R. and Guerfali W., "The generation of handwriting with delta-lognormal synergies," *Biological Cybernetics*, vol. 78, no. 2, pp. 119-132, 1998.

[129] Plamondon R. and Maarse F.J., "An Evaluation of Motor Models of Handwriting," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 5, pp. 1060-1072, 1989.

[130] Plamondon R. and Srihari S.N., "Online and offline handwriting recognition-a comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63-84, 2000.

[131] Prasad G.K., Khan I., Chanukotimath N.R., and Khan F., "On-line handwritten character recognition system for Kannada using Principal Component Analysis Approach: For handheld devices," in *Proc. of World Congress on Information and Communication Technologies (WICT)*, pp.675-678, 2012.

[132] Prasanth L., Jagadeesh V., Sharma R.R, Rao G.V.P., and Mandalapu D., "Elastic Matching of Online Handwritten Tamil and Telugu Scripts Using Local Features," in *Proc. 9th International Conference on Documentation Analysis and Research*, vol. 2, pp. 1028-1032, 2007.

[133] Rajashekararadhya S.V. and Ranjan P.V., "Efficient zone based feature extraction algorithm for handwritten numeral recognition of four popular south Indian scripts," *Journal of Theoretical and Applied Information Technology*, vol. 4, no. 12, pp. 1171-1181, 2008.

[134] Rampalli R. and Ramakrishnan A.G., "Fusion of complementary online and offline strategies for recognition of handwritten Kannada characters," *Journal of Universal Computer Science*, vol. 17, no. 1, pp. 81-93, 2011.

[135] Rocha J. and Pavlidis T., "A shape analysis model with applications to a character recognition system," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 4, pp. 393-404, 1994.

[136] Roy K., Vajda S., Pal U. and Chaudhuri B.B., "A system towards Indian postal automation," in *Proc. Ninth International Workshop on Frontiers in Handwriting Recognition*, pp. 580-585, 2004.

[137] Sabourin R., El-Yacoubi A., Gilloux M. and Suen C.Y., "An HMM-based approach for off-line unconstrained handwritten word modeling and recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 8, pp. 752-760, 1999.

[138] Sachan M.K., Lehal G.S., and Jain V.K., "A Novel Method to Segment Online Gurmukhi Script," in *Proc. Information Systems for Indian Languages Communications in Computer and Information Science*, vol. 139, pp. 1-8, 2011.

[139] Sanguansat P., Asdornwised W., and Jitapunkul S., "Online Thai handwritten character recognition using hidden Markov models and support vector machines," in *Proc. IEEE International Symposium on Communications and Information Technology*, vol. 1, pp. 492-497, 2004.

[140] Shankar G., Anoop V., and Chakravarthy V.S., "LEKHAK [MAL]: A System for Online Recognition of Handwritten Malayalam Characters," in *Proc. of National Conference on Communications (NCC 2003)*, pp. 463-467, 2003.

[141] Sharma A., Kumar R. and Sharma R. K., "Online Handwritten Gurmukhi in Character Recognition Using Elastic Matching," in *Proc. Congress on Signal and Image Processing*, vol. 2, pp. 391-396, 2008.

[142] Sharma A., Kumar R. and Sharma R. K., "Online preprocessing of handwritten Gurmukhi strokes," *International Journal of Machine Graphics and Vision*, vol. 18, no. 1, pp. 115-120, 2009.

[143] Sharma A., Kumar R. and Sharma R. K., "HMM Based Online Handwritten Gurmukhi Character Recognition," *International Journal of Machine Graphics and Vision*, vol. 19, no. 4, pp. 439-449, 2010.

[144] Sharma D. and Jain U., "Recognition of Isolated Handwritten Characters of Gurumukhi Script using Neocognitron," *International Journal of Computer Applications*, vol. 10, no. 8, pp. 10-16, 2010.

[145] Shin J., "On-line Cursive Hangul Recognition that uses DP Matching to Detect Key Segmentation Points," *Pattern Recognition*, vol. 37, no. 11, pp. 2101-2112, 2004.

[146] Shrivastava S.K. and Gharde S.S., "Support Vector Machine for Handwritten Devanagari Numeral Recognition," *International Journal of Computer Applications*, vol. 7, no.11, pp. 9-14, 2010.

[147] Siddharth, Singh K., Dhir R., and Rani R., "Handwritten Gurmukhi numeral recognition using different feature set," *International Journal of Computer Application*, vol. 28, no.2, pp. 20-24, 2011.

[148] Simoncini L. and Kovacs M., "A system for reading USA Census 90 handwritten fields," in *Proc. of International Conference on Document Analysis and Recognition*, vol. 2, pp. 86-91, 1995.

[149] Singh G., "Panjabi Primer Scientifically Designed for Easy Learning," Canada: Canadian Sikh Study and Teaching Society, 1995.

[150] Singh G., Kumar C.J., Rani R., and Dhir R., "Feature Extraction of Gurmukhi Script and Numerals: A Review of Offline Techniques," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 1, pp. 257-263, 2013.

[151] Sinha R.M.K. and Mahabala H.N., "Machine Recognition of Devanagari Script," *IEEE Transaction on Systems Man and Cybernetics*, vol. 9, no. 8, pp. 435-441, 1979.

[152] Sinha R.M.K., "Rule based contextual post-processing for Devanagari text Recognition," *Pattern Recognition*, vol. 20, no. 5, pp. 475-485, 1987.

[153] Slavik P. and Govindaraju V., "Equivalence of different methods for slant and skew corrections in word recognition applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 3, pp. 323-326, 2001.

[154] Subrahmonia J. and Zimmerman T., "Pen computing: challenges and applications," in *Proc. 15th International Conference on Pattern Recognition*, vol. 2, pp. 60-66, 2000.

[155] Suen C.Y., Berthod M., and Mori S., "Automatic Recognition of Handprinted Characters-The State of the Art," *Proc. of the IEEE*, vol. 68, no. 4, pp. 469-487, 1980.

[156] Suen C.Y., Nadal C., Legault R., Mai T.A., and Lam L., "Computer Recognition of Unconstrained Handwritten Numerals," *Proc. of the IEEE*, vol. 80, no. 7, pp. 1162-1180, 1992.

[157] Sundaresan S. and Keerthi S.S., "A study of representations for pen based handwriting recognition of Tamil characters," in *Proc. International Conference on Document Analysis and Recognition*, pp. 422-425, 1999.

[158] Swethalakshmi H., Jayaraman A., Sekhar C.C., and Chakravarthy V.S., "Online handwritten character recognition of Devanagari and Tamil scripts using support vector machines," in *Proc. International Workshop on Frontiers in Handwriting Recognition*, pp. 367-372, 2006.

[159] Tai J.W., Liu Y.J., and Zhang L.Q., "A model based detecting approach for feature extraction of off-line handwritten Chinese character recognition," in *Proc. the Second International Conference on Document Analysis and Recognition*, pp. 826-829, 1993.

[160] Tappert C., "Cursive script recognition by elastic matching," *IBM Journal of Research and Development*, vol. 26, no. 6, pp. 765-771, 1982.

[161] Tappert C., Suen C.Y., and Wakahara T., "The state of the art in online handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 8, pp. 787-808, 1990.

[162] Taylor S. and Cristianini N., "Kernel Methods for Pattern Analysis," Cambridge University Press, 2004.

[163] Trier O.D., Jain A.K., and Taxt T., "Feature Extraction Methods for Character Recognition-A survey," *Pattern Recognition*, vol. 29, no 4, pp. 641-662, 1996.

[164] Uchida S. and Sakoe H., "A survey of elastic matching techniques for handwritten character recognition," *The Institute of Electronics, Information and Communication Engineering Transaction Information and Systems*, vol. E88-D, no. 8, pp. 1781-1790, 2005.

[165] Vamvakas G., Gatos B., Petridis S., and Stamatopoulos N., "An Efficient Feature Extraction and Dimensionality Reduction Scheme for Isolated Greek Handwritten Character Recognition," in *Proc. Ninth International Conference on Document Analysis and Recognition*, vol. 2, pp.1073-1077, 2007.

[166] Verma B., Blumenstein M., and Ghosh M., "A novel approach for structural feature extraction: Contour vs. direction," *Pattern Recognition Letters*, vol. 25, no. 9, pp. 975-988, 2004.

[167] Verma B., Blumenstein M., and Kulkarni S., "Recent Achievements in Off-line Handwriting Recognition," in *Proc. of 2nd International Conference on Computational Intelligence and Multimedia Applications, World Scientific Publishing Company*, pp. 27-33, 1998.

[168] Verma K. and Sharma R.K., "Performance Analysis of Zone Based Features for Online Handwritten Gurmukhi Script Recognition using Support Vector Machine," in *Proc. of the Twenty-Third International Conference on Systems Engineering*, pp. 747-753, 2014.

[169] Vertanen K., "Combining open vocabulary recognition and word confusion networks," *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4325-4328, 2008.

[170] Viard-Gaudin C., Lallican P.M., Knerr S., and Binter P., "The IRESTE On/Off (IRONOFF) dual handwriting database," in *Proc. the Fifth International Conference on Document Analysis and Recognition*, pp. 455-458, 1999.

[171] Vuori V. and Laaksonen J., "A comparison of techniques for automatic clustering of handwritten characters," in *Proc. International Conference on Pattern Recognition*, vol. 3, pp. 168-171, 2002.

[172] Vuurpijl L., Niels R., van Erp M., Schomaker L. and Ratzlaff E., "Verifying the UNIPEN devset," in *Proc. Ninth International Workshop on Frontiers in Handwriting Recognition*, pp. 586-591, 2004.

[173] Wang D.H., Liu C.L., and Zhou X.D., "An approach for real-time recognition of online Chinese handwritten sentences," *Pattern Recognition*, vol. 45, no. 10, pp. 3661-3675, 2012.

[174] Wink A.M. and Roerdink J.B., "Denoising functional MR images: a comparison of wavelet de noising and Gaussian smoothing," *IEEE Transactions on Medical Imaging*, vol. 23, no. 3, pp 374-387, 2004.

[175] Xiao X. and Leedham G., "Knowledge-based English cursive script segmentation," *Pattern Recognition Letters*, vol. 21, no. 10, pp. 945-954, 2000.

[176] Yamada H., Yamamoto K., and Saito T., "A non-linear normalization method for handprinted Kanji character recognition line density equalization," *Pattern Recognition*, vol. 23, no. 9, pp. 1023-1029, 1990.

[177] Yanikoglu B. and Sandon P.A., "Segmentation of off-line cursive handwriting using linear programming," *Pattern Recognition*, vol. 31, no. 12, pp. 1825-1833, 1998.

[178] Yimei D., Fumitika K., Yasuji M., and Malayappan S., "Slant estimation for handwritten words by directionally refined chain code," in *Proc. the Seventh International Workshop on Frontiers in Handwriting Recognition*, pp. 53-62, 2000.

[179] Zanibbi R., Blostein D., and Cordy J.R., "Recognizing mathematical expressions using tree transformation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 11, pp. 1455-1467, 2002.