Recognition of online Handwritten Punjabi numerals

Thesis submitted in partial fulfilment of the requirement for

the award of the degree of

Masters of Science

in

Mathematics and Computing

Submitted by

Shivali

Roll No. – 300903017

Under

the guidance of

Dr. R. K. Sharma

JULY 2011

School of Mathematics and Computer Applications

Thapar University

Patiala – 147004(PUNJAB)

INDIA

CONTENTS

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled "Recognition of online handwritten Punjabi numerals" in partial fulfilment of the requirements for the award of degree of Master of Mathematics and Computing, School of Mathematics and Computer Applications, Thapar University, Patiala is an authentic record of my own work carried out under the supervision of Dr. R. K. Sharma. The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Shivali
(Shivali)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge

(Dr. R. K. Sharma) 15.7.11

Professor
SMCA,Thapar University
Patiala

Countersined By:

Dr. S.S. Bhatia 15/7/11

(Professor and Head)
SMCA,Thapar university
Patiala

Dr. S.K. Mohapatra

(Dean of Academic Affairs)
Thapar University
Patiala

.

ACKNOWLEDGEMENT

"Accomplishment of a task with a desired success calls for dedication towards work and prompting guidance, co-operation and deliberation from seniors". Nothing big can be achieved without the support of others and one must be indebted to them by words and deeds. So coordinated effort of others bridges the gap between where you are and where you want to be.

I would like to express my deep gratitude to **Dr. R.K. Sharma** my project guide for his constant co-operation. He was always there with his competent guidance and valuable suggestion throughout the pursuance of this research project. His views, advices and directions have been invaluable during the course of this project.

I also want to express my regards to Dr. S. S. Bhatia, Head of Department School of Mathematics and Computer Applications, Thapar University, Patiala for providing keen interest, unfailing support, inspiration and necessary research facilities in the school.

Above all no words can express my feelings to my parents, friends all those persons who supported me during my project. Last but not least I would also like to place of appreciation to all the respondents whose responses were of utmost importance for the project. I would also like to thank almighty God for his blessings showered on me during the completion of project report. Mere acknowledgement may not redeem the debt.

Shivali

(Shivali)

List of figures

CHAPTER 1

INTRODUCTION

In this advanced technology age, it is necessary for humans to take help from computers to complete their works efficiently, quickly and accurately. In order to take help from computers, it is required to interact with them without having to learn any extra skill. This interaction is done through input devices namely keyboard, mouse etc. that have their own limitations.

The two famous methods of inputting, other than keyboard and mouse, are speech and handwriting. Speech is more natural to humans (we learn to speak before we learn to write), but it has its own limitations, for example in an environment where several people are working on their own computer terminals such as in a lab, speech input is not the most desirable input method as cross talk becomes the real concern. In contrast, handwriting input is not impeded by noisy environment. So the writing, which is the natural mode of collecting, storing and transmitting the information through centuries, now serves not only for communication among humans, but also, serves for the communication of humans and machines.

As such, online handwriting recognition has a great potential to improve communication between user and computers. The various devices used in online handwriting recognition are PDA, Tablet PCs, Cross Pad etc.

1.1 History of character recognition

The Early age (1900-1980): The origin of character recognition started when the Russian Scientist Tyuring tried to develop an aid for visually handicapped [1]. In the middle of the 1940s, the first character recognizer appeared [2]. During early days, recognition was done on the machine-printed text or on small set of well-distinguished hand-written text or symbols. For handwritten text, low-level image processing techniques have been used on the binary image to extract feature vectors, which are then fed to statistical classifiers. Few successful but constrained algorithms were implemented mostly for Latin characters and numerals. However, some studies on Japanese, Chinese, Hebrew, Indian, Cyrillic, Greek and Arabic characters and numerals in both machine-printed and handwritten cases were also initiated during this period [3], [4], [6], [7].The commercial character recognizers were available in1950s, when electronic tablets capturing the *x-y* coordinates of pen-tip movement were first

introduced. This innovation enabled the researchers to work on the on-line handwriting recognition problem.

Further Developments (1980-1990): There was lack of powerful computer hardware and data acquisition devices up to 1980. Later on CR system developed with the increase in information system [8], [9], [10] and structural approaches were initiated in many systems in addition to the statistical methods. These systems broke the character image into a set of pattern primitives such as lines and curves. The rules were then determined which character most likely matched with the extracted primitives [11], [12], [13]. However, the CR research was focused on basically the shape recognition techniques without using any semantic information. This led to an upper limit in the recognition rate, which was not sufficient in many practical applications. The development in CR, during this period can be found in [14] and [15] for off-line and on-line case, respectively.

The Advancements (1990 onwards): During this period, the real progress on CR systems has been achieved. In the early nineties, Image Processing and Pattern Recognition techniques were efficiently combined with the Artificial Intelligence methodologies. Nowadays, in addition to the more powerful computers and more accurate electronic equipments such as scanners, cameras and electronic tablets, we have efficient, modern use of methodologies such as Fuzzy Set Reasoning, Natural Language Processing, Neural Networks and Hidden Markov Models. The recent systems for the machine-printed off-line [16], [17] and limited vocabulary, user dependent on-line handwritten characters [18], [19], [20] are quite satisfactory for restricted applications.

1.2 Classification of character recognition

In this section, we discuss the classification of character recognition. This has two distinct areas as given in Fig. 1.1.

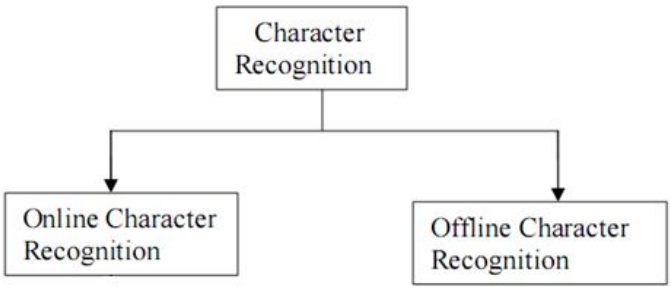


Figure 1.1: Classification of character recognition

1.2.1 Offline Character Recognition

Offline handwriting recognition refers to the process of recognizing words that have been scanned from the surface (such as a sheet of paper) and are stored digitally in grey scale format. After being stored, it is conventional to perform further processing to allow superior recognition.

1.2.2 Online Character Recognition

In this case handwriting is captured and stored in a digital form via different means. Usually a special pen is used in conjunction with an electronic surface. As the pen moves across the surface, the two dimensional coordinates of successive points are represented as a function of time and are stored as a function of time. In the present work, we have dealt only with online character recognition.

1.3 Advantages of Online Recognition over Offline Recognition

a) It is a real time process: Digitizer captures the sequence of co-ordinates points while writing.

b) It is adaptive in real time: The writer gives immediate results to the recognizer for improving the recognition rate as she keeps writing on tablet and observe the results.

c) It captures the dynamic and temporal information of pen trajectory: This information consists of number and order of pen –strokes, the direction and speed of writing for each pen-stroke.

d) Very little pre-processing is required: The operations such as smoothing, de-skewing, detection of line orientations, loops and cusps are easier and faster with the pen trajectory data then on pixel images.

e) Segmentation is easy: Segmentation operations are facilitated by using the temporal and pen lift information, particularly for hand printed characters.

1.3.1 Disadvantages of Online handwritten character recognition

a) Pen and paper are not sufficient, the writer requires special equipments.

b) It cannot be applied to documents printed or written on papers.

c) The available systems for online handwritten character recognition are slow and recognition rates are low for handwriting that is not clean.

1.4 Literature Review

A number of technologies are available in the form of tablets or writing pads for capturing handwritten data. These technologies are based on electronic or electromagnetic or electrostatic or pressure sensitive techniques and the tablets with combination of input and output digitizer or display on same surface are most common in handwriting recognition. The process of online handwritten characters recognition

includes following phases or components: data collection, pre-processing, feature extraction or computation of features, segmentation, recognition and post-processing [22], [23]. During the literature review, it has been noted that segmentation can be performed before or after pre-processing. The output obtained from one phase becomes input to the next phase. These phases are illustrated in Fig. 1.2. Subsection 1.4.1 gives the details of data collection devices and subsections 1.4.2 and 1.4.3 discuss the literature on pre-processing and recognition phases, in detail.

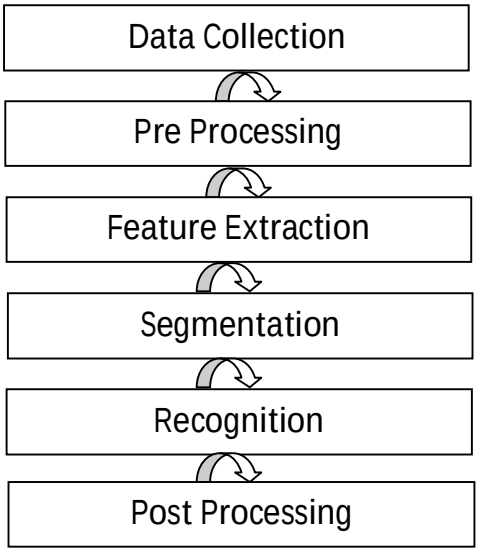


Figure 1.2: Phases of online handwritten recognition

1.4.1 Data Collection

Data collection is the first phase in online handwriting recognition that collects the sequence of coordinate points of the moving pen. It requires a transducer that captures the writing as it is written. The most common of these devices is the electronic tablet or digitizer. These devices uses a pen that is digital in nature. A typical pen includes two actions, namely, PenDown and PenUp. The pen traces are sampled at constant rate, therefore these pen traces are evenly distributed in time and not in space.

The common names of electronic tablet or digitizer are personal digital assistant, cross pad (or pen tablet) and tablet PC. Common personal digital assistants available in the market are Pen Pad PDA 600, Apple, Casio, Z-7000, Omni Go 100, IBM, IBM Work Pad, Motorola, Marco, Envoy, Sharp, Zaurus ZR-5000FX, Zaurus ZR-5800FX, Sony, PIC-1000, Digital, SA-110 Strong ARM, Tungsten, Nokia and HTC etc. The selection of these hardware devices is mainly based on compatibility with operating system in use.

1.4.2 Pre-processing

Noise or distortions present in input text due to some limitations is removed using the pre-processing [24], [25]. These noise or distortions include irregular size of text, missing points during pen movement collections, jitter present in text, left or right bend in handwriting and uneven distances of points from neighbouring positions. In online handwriting recognition, pre-processing includes five common steps, namely, size normalization and centering, interpolating missing points, smoothing, slant correction and resampling of points [22].

Size normalization depends on how user moves the pen on writing [26]. There are missing points (due to high speed of writing) that can be interpolated using various techniques including Bezier interpolation [27]. Jitter present in the writing that is removed using Smoothing [28]. It averages a point with its neighbours. The shape of input handwritten character is disturbed as most of the writers handwriting is bend to left or right directions, it can be solved using slant correction [29], [30], [31]. Handwritten words are usually slanted or italicized due to the mechanism of handwriting and the personality. Resampling of points refers to the points in the list are to be equidistant from neighbouring points as far as feasible. It means that new data points are calculated on the basis of the original points of list. Pre-processing techniques have been discussed in [32].

Common steps in pre- processing phase are:

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

We have implemented size normalization, interpolation of missing points, smoothing and resampling of points in the present work.

1.4.3 Recognition

There are four types of recognition methods statistical, syntactical and structural, neural network and elastic matching [33], [34]. These methods of recognition are discussed below.

Statistical Recognition

In statistical approach, each pattern is viewed as a point in *d*-dimensional space and is represented in terms of *d* features. This involves selection of features that all pattern vectors belonging to different categories or classes to occupy disjoint region in a *d*-dimensional space [34]. The statistical methods assume variations in natural handwriting as stochastic in nature and are based on prior probabilities of classes. Parametric and non-parametric methods are the two classifications of statistical methods. In parametric methods, handwriting samples are characterized by a set of parameters and also they are random variables from distribution. The selection of parameters is based on training data. Hidden Markov Model (HMM) is an example of parametric methods.

Non-parametric methods are based on direct estimation from training data. The common non-parametric methods are the *k*-nearest neighbours.

Introduction to HMMs has been given by Rabiner [35]. In this tutorial, use of HMMs has been explained with respect to speech and cursive handwriting. The reference of these tutorials has been noted in most of the research work in HMMs after 1990. Later on, HMM is used in isolated online handwritten characters and an average error rate obtained is 6.9%. HMM is used in online handwriting recognition as this reduces memory usage and further improves the recognition rate [36].

Structural and Syntactical Methods

In handwritten patterns where structures and grammar are considered, we use structural and syntactical methods. Structural recognition describes the construction of a pattern from the primitives.

In case of syntactical methods, a formal analogy is drawn between the syntax of a language and the structure of patterns. The sentences are generated according to a grammar. The primitives are viewed as the alphabet of the language and the patterns are viewed as sentences belonging to a language. The grammar for each pattern class must be inferred from the available training samples [34]. A structural approach is proposed for recognizing online handwriting in [37]. A good introduction is given related to the problems in structural and syntactical methods in [38].

Neural Network Methods

Neural networks have the ability to learn complex nonlinear input-output relationships, use sequential training procedures and adapt themselves to the data [34]. Another popular network is the self-organizing map or kohonen network [39].

A recognition system for cursive handwriting based on a variant of the self-organizing map algorithm is developed in [40]. Three sophisticated neural network classifiers that include multiple multilayer perceptron classifiers and solve complex pattern recognition problems have been described in [41]. A

small scale four-layered neural network model for simple character recognition is developed in [42] that can recognize the patterns transformed by affine conversion.

Elastic Matching Methods

Elastic matching is used to determine the similarity between two entities. It is a generic operation in pattern recognition. Considering all allowable changes, the stored template is matched against the pattern to be recognized. The similarity measure can be optimized based on available training set.

An online character recognizer based on elastic matching is given in [43]. These authors have further compared linear matching with elastic matching in [44]. An online handwritten recognition system based on elastic matching was developed in [45]. An online handwritten note where dissimilarity of an input curve and template curve were measured was proposed in [46].

Now elastic matching has also been used in recognizing handwritten characters and images [47] and [48].

1.5 Overview of Gurmukhi numerals

Gurmukhi script is used across the world. It is written in top-down approach and in left-to-right direction. There are three horizontal zones, namely upper zone, middle zone and lower zone in which the words of Gurmukhi script are partitioned The region above the head line is upper zone, where some of the vowels and sub-parts of some other vowels reside, while the area below the head line is middle zone where the consonants and some sub-parts of vowels are present. The area below the middle zone where some vowels and certain half characters lie in the foot of consonants is lower zone. Fig.1.3 depicts distinct digits that are followed in Gurmukhi script. The present study is focused on recognizing these numerals in online handwriting.

Symbol	Illustration
	ਬਿੰਦੀ (BINDI)
	ਇੱਕ (IKK)
	ਦੋ (DO)
	ਤਿੰਨ (TINN)

੪	ਚਾਰ (CHAAR)
੫	ਪੰਜ (PANJ)
੬	ਛੇ (CHHE)
੭	ਸੱਤ (SAT)
੮	ਅੱਠ (ATtH)
੯	ਨੌਂ (NAU)

Figure 1.3: Gurmukhi numerals and their names

This dissertation is divided into four chapters. First chapter (the ongoing chapter) focuses on introducing the problem and also contains a very brief literature review on this topic. Chapter 2 contains the illustration on data collection and pre-processing phase. We have implemented four pre-processing phases in this work. Chapter 3 introduces the concept of support vector machines and also illustrates its usage for the present problem on handwritten Gurmukhi numerals. Chapter 4 concludes the work and also throws some light on future work that can be carried out in this direction.

CHAPTER 2

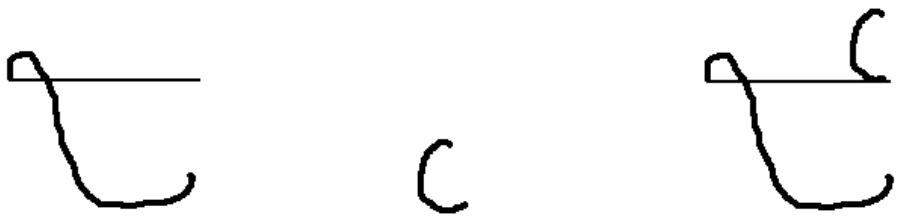
DATA COLLECTION PRE PROCESSING

Data collection and pre-processing are the two important phases required to be applied before recognition in online handwritten recognition systems. This chapter contains the data collection phase and the pre-processing algorithms.

2.1 DATA COLLECTION PHASE

The characters are inputted with the help of mouse or written on writing pads by a digital pen. The sequential positions of the writing device, i.e., the dynamic positions are noted. The time sequential signal from the pen movements on the writing pad, collected by sensor attached to the pad, is described as a sequence of consecutive points on the *x–y* plane. When we write slowly, the sample points are located densely, whereas fast handwriting produces sparsely located points. The points generated by these pen movements are stored in a list.

In Gurmukhi, each character is a group of one or more strokes as given in Fig. 2.1 for the numeral ੯ (‘9’). A stroke is a list that includes recorded points stored in sequential order. Start and end of a stroke depends on PenDown and PenUp function of the input device in use. The data collected in the data collection phase is sent to the pre-processing phase, the next phase of recognition system.



Stroke 1 Stroke 2 Numeral 9

Figure 2.1: Strokes of a character

Fig. 2.1 contains two strokes, stroke 1 and stroke 2. Combining these strokes will give us the complete numeral ੯ (‘9’).

2.2 Data collection

The given window represents the points when we move the pen on the writing pad. This helps in the collection of input pen movements and stores these pen movements in a list and also in a text file. We can use the list storage for pre-processing and maintain the text file storage for later stages.

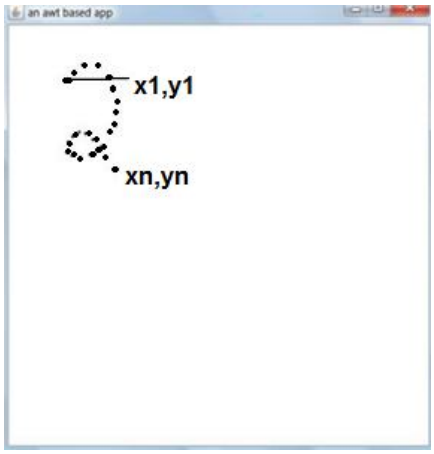


Figure 2.2: Points of pen movements

The co-ordinates *x* and *y* are stored in a .txt file. The further operations are applied to the points taken from that file. The .txt file is shown in Fig 2.3. Here 1 represents the co-ordinate x_1 and 2 represents the co-ordinate y_1. Similarly 3 represents the co-ordinate x_2, 4 represents the co-ordinate y_2 and so on.

2.3 Pre-processing

The main aim of pre-processing is to normalize words and remove variations that would otherwise create complications in recognition. It discards the irrelevant information. Generally the irrelevant information includes the duplicated points, jitter etc. The main steps of pre processing are size normalization, interpolation, smoothing, resampling of points and slant correction that are shown in Fig. 2.4.

2.3.1 Size normalization of a stroke

Size of the input stroke depends on the movements of a pen by the user. Strokes are not of same size, when we move the pen again and again. Size normalization normalizes every stroke to the same size. The algorithm used for size normalization is given below. Fig. 2.5 represents the stroke after normalization.

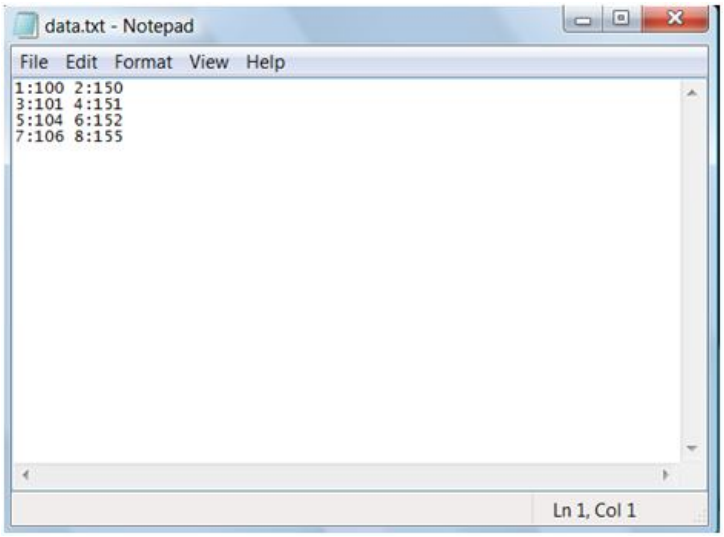


Figure 2.3: data collected in .txt file

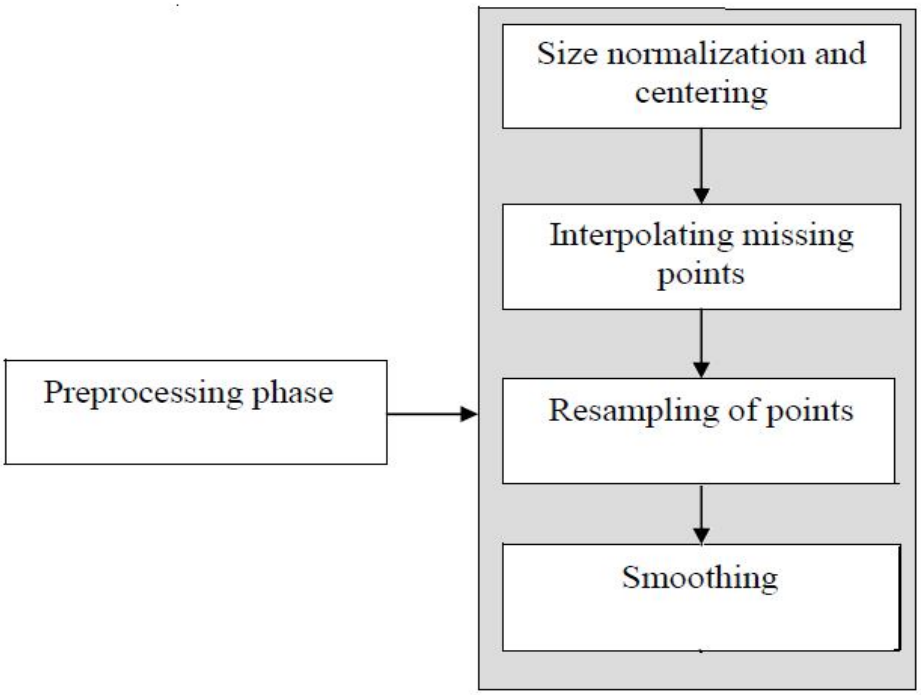


Figure 2.4: Pre processing steps

In this algorithm, origin of the frame of reference is taken as (x_0, y_0) and the set of pixels in which a Gurmukhi numeral is drawn is given by $\{(x, y): 0 \leq x \leq l_x , l_y \leq y \leq 0\}$ where l_x, l_y are the lengths in x and y directions. It may be noted that there are n pixels in a Gurmukhi character.

Algorithm 1: Size Normalization

1. Set L_x = 200 (pixels), L_y = 200 (pixels)

2. $P_{ix} = P_{ix} * L_x \frac{}{l_x}$, $P_{iy} = P_{iy} * \frac{L_y}{} l_y$ for all points P_i in list, i = 1, 2, . . . , n.

3. $P_{ix} = P_{ix} + x_0$, $P_{iy} = P_{iy} + y_0$ for all points P_i in list i = 1, 2, . . . , n.

This algorithm normalizes the stroke in size (A code in Java is given in Appendix).

In Fig. 2.6 we have shown the window that is divided into two parts. First part is the writing area (the space where we move the pen) and the next part is the normalization area (where character is shown after normalization).

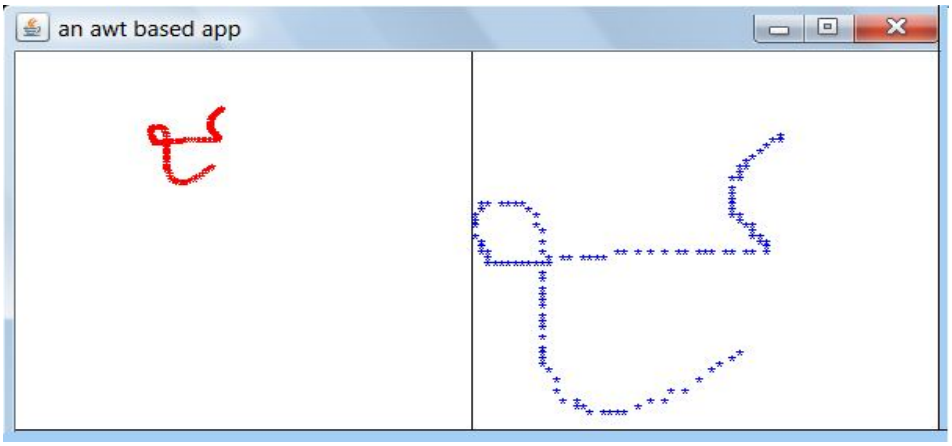


Figure 2.5: Handwritten stroke after normalization

2.3.2 Missing point interpolation

When a character is drawn with high speed, it will have some points missing in it. These missing points can be interpolated using various techniques such as Bezier and B-Spline. In this work, cubic Bezier interpolation has been used. In this piecewise interpolation technique, a set of consecutive four points is considered for obtaining the Bezier curve. The next set of four points gives the next Bezier curve.

We have applied Bezier interpolation between the points where distance is greater than one. Firstly, we find the distance between the pair of consecutive points and call Bezier where distance between points greater than one. For example, let us take say P_2 and P_3 as the points whose distance is greater than one. Now we call Bezier (function) for the set of four points (P_1, P_2, P_3, and P_4) for obtaining the curve. But we plot only those interpolated points that are between P_2 and P_3.

Algorithm 2 Interpolating missing points

a. If distance between points P_i and P_{i+1} is greater than 1.

CALL Bezier (P_i, P_{i+1}, P_{i+2}, P_{i+3}) for all points P_i, i = 1, 2, . . . , m-3, where m is the total no. of points in the stroke.

Function Bezier (P_i, P_{i+1}, P_{i+2}, P_{i+3})

1. u is the variable such that $0 \leq u \leq 1$
2. set $u = 0.2$ and $\Delta u = 0.2$
3. repeat steps 4 and 5 until $u \leq 1$
4. calculate x co-ordinate of new point as

   $$P_{ix} * (1-u)^3 + P_{(i+1)x} * 3 * u * (1-u)^2 + P_{(i+2)x} * 3 * u^2 * (1-u) + P_{(i+3)x} * u^3$$

   and the y co-ordinate as

   $$P_{iy} * (1-u)^3 + P_{(i+1)y} * 3 * u * (1-u)^2 + P_{(i+2)y} * 3 * u^2 * (1-u) + P_{(i+3)y} * u^3$$

5. set $u = u + \Delta u$
6. return

Fig. 2.7 contains the character after Bezier interpolation. This figure consists of three partitions, first partition shows the shape of moving pen. The second partition shows the character after normalization. We take normalized points and find distance between the pair of consecutive points and call Bezier where distance is greater than one. The third partition shows the character after Bezier point interpolation.

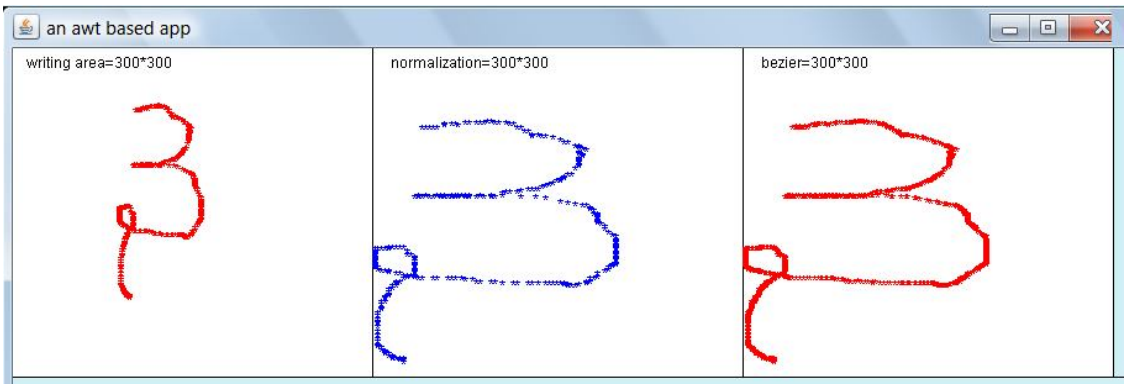


Figure 2.6: Bezier interpolation

2.3.3 Resampling of points

In this step of pre-processing, we filter the stroke points to a fix number of points and retain the shape of the stroke. The shape of the character is maintained after removing some points. Let us consider an example which shows the method to remove the points.

Suppose that we have 18 points and we want to take only 5 points from them. It means that we have to remove 13 points. For removing the points, we follow the following steps:

a. Dividing 18 by 5, we get 3.6, by taking floor and ceil value of 3.6 we get 3 and 4, that means we have to select one point and leave 3 points then select one point and leave 4 points and so on till the last point. Consider points:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

In this, first of all 1 is selected and then we have to leave 2, 3 and 4 (3 points) for the selection of another point. After leaving 3 points we select 5^{th} point, and now leave 4 points (6, 7, 8, 9) and select 10^{th} point and so on. In the last after the selection of 14 if we leave 4 points then our requirement of 5 points will not be completed, so we leave 3 points instead of 4.

Another example for resampling of points:

Suppose we have 19 points and we have to select 7 points, then 19/7= 2.3 imply we have to leave 2 or 3 points.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

We first select 1 and then leave 2 points. Then select 4 and leave 3 points and so on. After the selection of 18 we have to leave 3 points but we have no choice so we take 19 and thus get 7 points after resampling.

Fig. 2.8 gives the Gurmukhi numeral ੪ (‘4’) after resampling of points. In this figure, points after Bezier interpolation were 418 and after resampling they are reduced to 60 (A code in Java is given in Appendix).

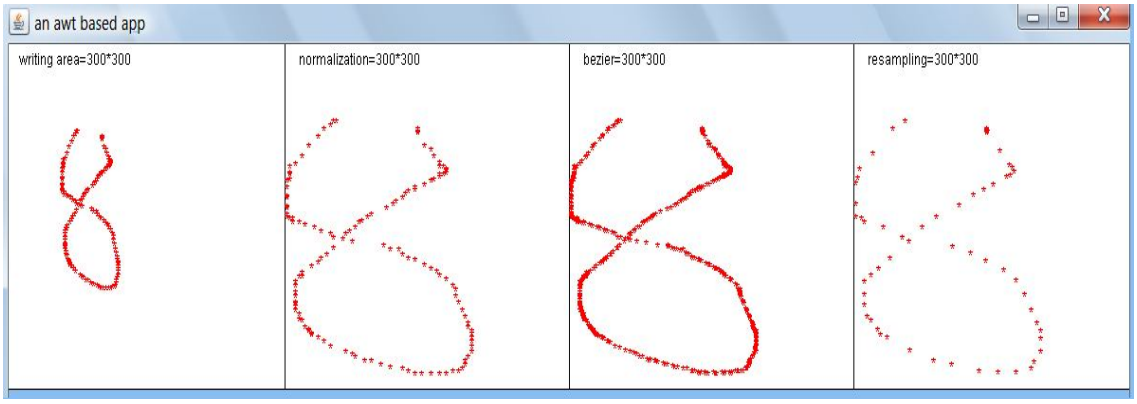


Figure 2.7: Resampling of the numeral ৪ (‘4’)

2.3.4 Smoothing

Sharpness of edges is present in handwriting because of individual handwriting style. This sharpness can be removed by modifying each point of the list with mean value of the neighbours and the angle subtended at the k^{th} position from each end [28]. Fig. 2.9 depicts how 2-neighbors from each side are used for this purpose. In this figure, five points of the list, generated in the previous step, have been used for smoothing the stroke. The point P_i has been modified with the help of points $P_{(i-2)}$, $P_{(i-1)}$, $P_{(i+1)}$, $P_{(i+2)}$. If we consider more than five points then we tend to lose the nature of stroke in terms of sharp edges. Algorithm for smoothing is given below (A code in Java is given in Appendix).

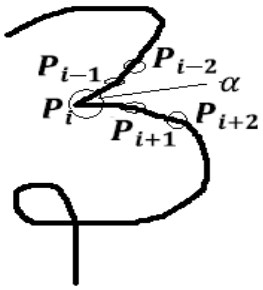


Figure 2.8: Smoothing of a numeral

Algorithm 3: Smoothing

1. Repeat step 2 for each stroke
2. (a). Calculate m the total no. of points in a stroke.

   (b). Repeat steps (c) and (d) for all points P_i, i= 3, 4, . . . , m-2

   (c). Calculate α = $\angle P_{i-2} P_i P_{i+2}$

(d). Set $P_{ix} = (P_{(i-2)x} + P_{(i-1)x} + \alpha * P_i ix + P_i(i+1)x + P_i(i+2)x)/(2 * 2 + \alpha)$.

Set $P_i iy = (P_i(i-2)y + P_i(i-1)y + \alpha * P_i iy + P_i(i+1)y + P_i(i+2)y)/(2 * 2 + \alpha)$.

3. Exit.

Fig. 2.10: contains the Gurmukhi numeral ੪ (‘4’) after smoothing.

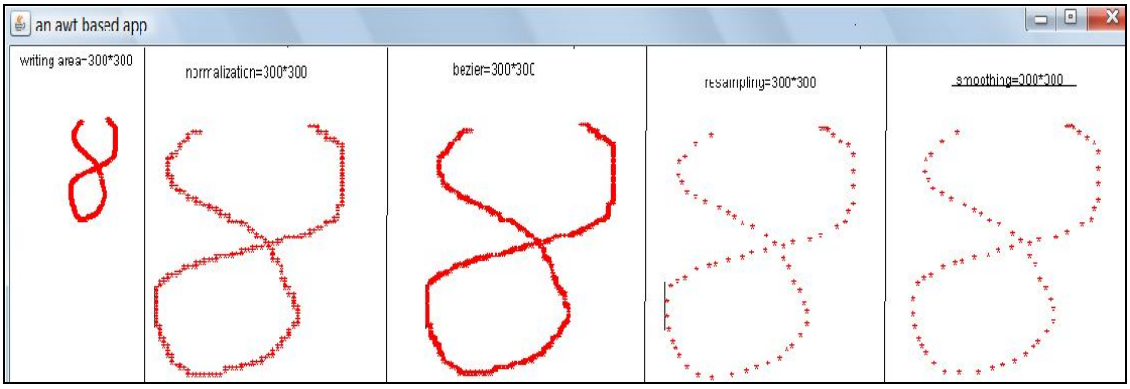


Figure 2.9: Smoothing of a numeral ੪(‘4’)

CHAPTER 3

RECOGNITION PROCESS

3.1 INTRODUCTION TO SVM

Support Vector Machines (SVM) are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in Fig. 3.1. In this example, the objects belong either to class WHITE or RED. The separating line defines a boundary on the right side of which all objects are WHITE and to the left of which all objects are RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as WHITE (or classified as RED should it fall to the left of the separating line).

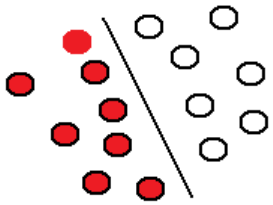


Figure 3.1: Linear classification

The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (WHITE and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). This situation is shown in Fig 3.2. Compared to the previous schematic, it is clear that a full separation of the WHITE and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.

The Fig. 3.3 shows the basic idea behind SVM's. Here we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels.

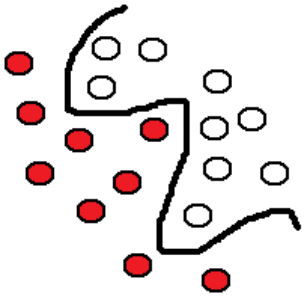


Figure 3.2: Complex classification

The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the WHITE and the RED objects.

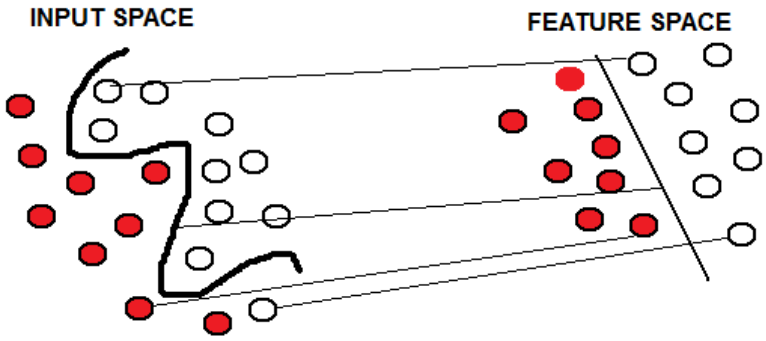


Figure 3.3: Idea behind SVM

SVM is primarily a method that performs classification tasks by constructing hyper planes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. For categorical variables a dummy variable is created with case values as either 0 or 1. Thus, a categorical dependent variable consisting of three levels, say (A, B, C), is represented by a set of three dummy variables.

3.2 SVM APPLICATIONS

SVM has been used successfully in many real world problems.

- Text(and hypertext) categorization
- Image classification
- Bioinformatics(protein classification, cancer classification)
- Hand-written character recognition

We have used SVM in handwritten character recognition in this work.

3.3 RECOGNITION OF NUMERALS

The process has been implemented on the handwritten Punjabi numeral data collected in this work. The numerals are written by a single writer. These numerals as given in an earlier chapter are ੦, ੧, ੨, ੩, ੪, ੫, ੬, ੭, ੮, ੯. The writer was asked to write 20 samples of each Punjabi numeral. As such, we have 200 samples in all that are created during this work. A part of the data collected is given in Fig. 3.4 as points in .txt file. All these samples undergo the pre processing phase and we finally get the resampled points that can be fed to the recognition phase. Since we can control the number of points in a resampled image, we have set up the experimentation on such a way that we may get 40, 50, 60 or 70 points in the resampled image. This is the first level of experimentation in this work.

In the second level of experimentation, we have implemented different strategies of partitioning the collected patterns in training and testing patterns, in this work. Three strategies have been adopted here. These are: a) 80% data in training, 20% data in testing; b) 90% data in training, 10% data in testing and; c) 95% data in training, 5% data in testing.

In this work, we have used SVM for the recognition process and as the third level of experimentation, we have obtained the results using three kernels in SVM. These kernels considered in this study are linear kernel, polynomial kernel and radial basis function kernel.

Linear kernel

Linear SVM is the newest extremely fast machine learning (algorithm for solving multi class classification problems from ultra large data sets that implements an original proprietary version of a cutting plane algorithm for designing a linear support vector machine. Linear SVM is a linearly scalable routine meaning that it creates an SVM model in a CPU time which scales linearly with the size of the training data set.

Features

a. Efficiency in dealing with extra large data sets (say, several millions training data pairs).

b. Solution of multiclass classification problems with any number of classes.

c. Working with high dimensional data (thousands of features, attributes) in both sparse and dense format.

d. No need for expensive computing resources (personal computer is a standard platform).

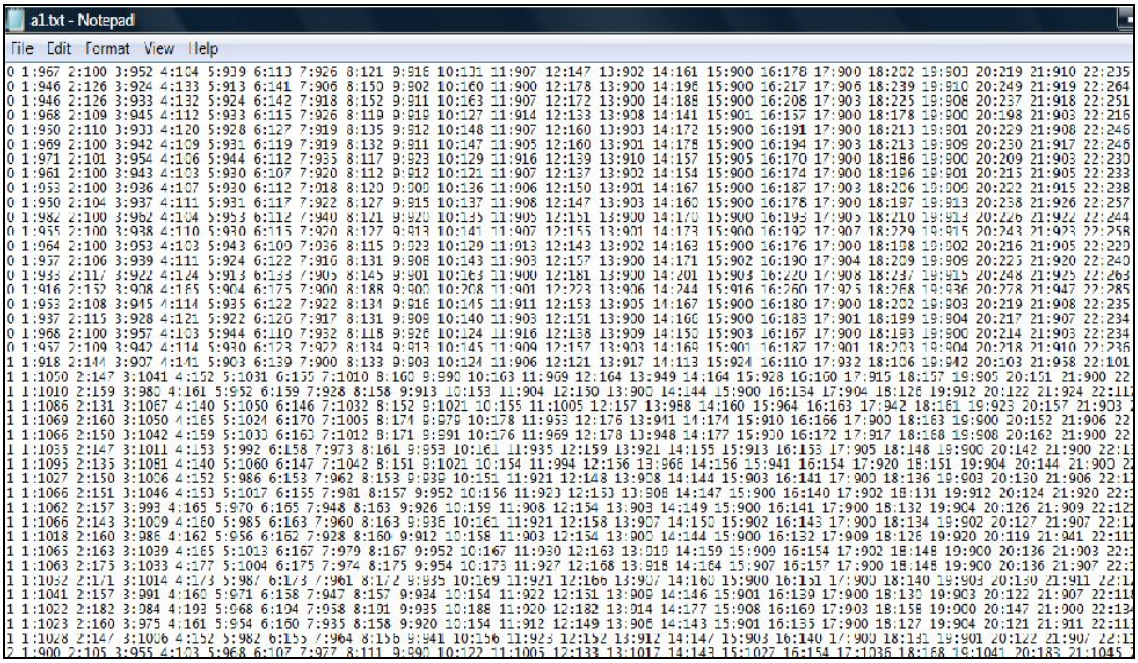


Figure 3.4: Points in .txt file

Polynomial kernel

The Polynomial kernel is a non-stationary kernel. Polynomial kernels are well suited for problems where all the training data is normalized.

$$K(x, y) = (\alpha x^T y + c)^d$$

In this type of kernel, the adjustable parameters are the slope alpha, the constant term ***c*** and the polynomial degree *d*.

Radial basis function kernel

It is defined on the interval [−1, 1]. It is given by the recursive formula:

$$K(x, y) = B_{2p-1}(x - y)$$

where $p \in \mathbf{N}$ with $B_{i+1} = B_i \otimes B_0$ also $K(x, y) = \prod_{p=1}^{d} B_{2n+1}(x_p - y_p)$

3.3.1 Results and Discussions

a. Recognition with 40 resampled points

We have first of all used 40 co-ordinates for each of the 200 data points collected from the user. As such we have a 80 dimension feature vector that is fed to SVM. The patterns are partitioned into three strategies and in each strategy, three kernel functions of SVM are used. The results obtained by this recognition are shown in table 3.1 and Fig. 3.5 depicts these results graphically.

Table 3.1 Recognition accuracy with 40 resampled points

PARTITIONING STRATEGY / KERNEL FUNCTIONS	80% TRAINING 20% TESTING	90% TRAINING 10% TESTING	95% TRAINING 5% TESTING
Linear	97.5%	80%	20%
Polynomial	87%	70%	20%
RBF	20%	20%	10%

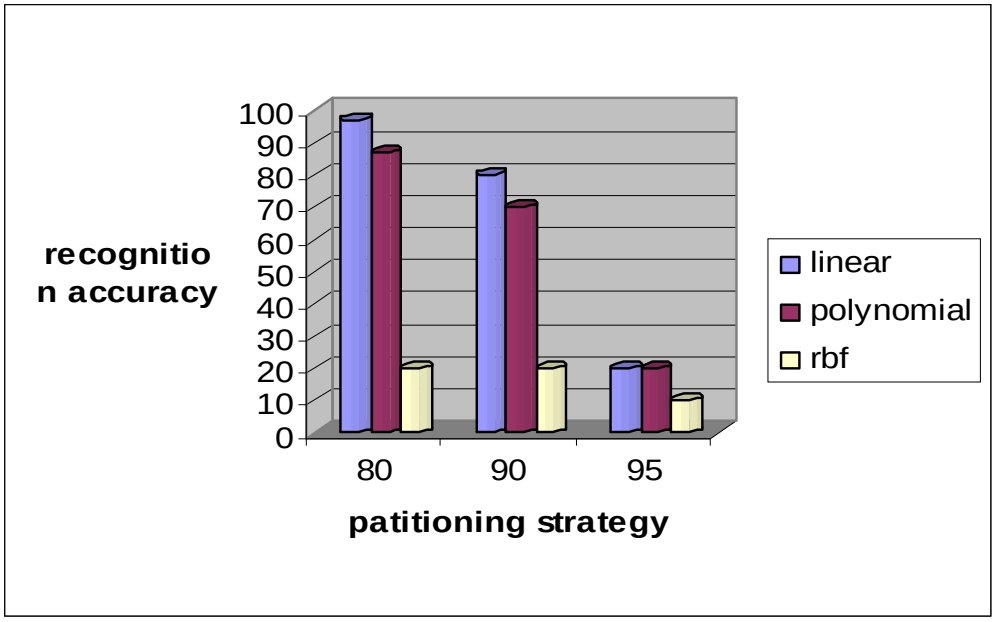


Figure 3.5: Recognition of numerals with 40 resampled points

b. Recognition with 50 resampled points

In this case, we have used 50 co-ordinates for each of the 200 data points collected from the user. As such we have a 100 dimension feature vector that is fed to SVM. The patterns are partitioned into three

strategies and in each strategy, three kernel functions of SVM are used. The results obtained by this recognition are shown in table 3.2 and Fig. 3.6 depicts these results graphically.

Table 3.2: Recognition accuracy with 50 resampled points

PARTITIONING / KERNEL FUNCTIONS / STRATEGY	80% TRAINING 20% TESTING	90% TRAINING 10% TESTING	95% TRAINING 5% TESTING
Linear	70.7317%	60%	50%
Polynomial	65%	50%	40%
RBF	9%	10%	10%

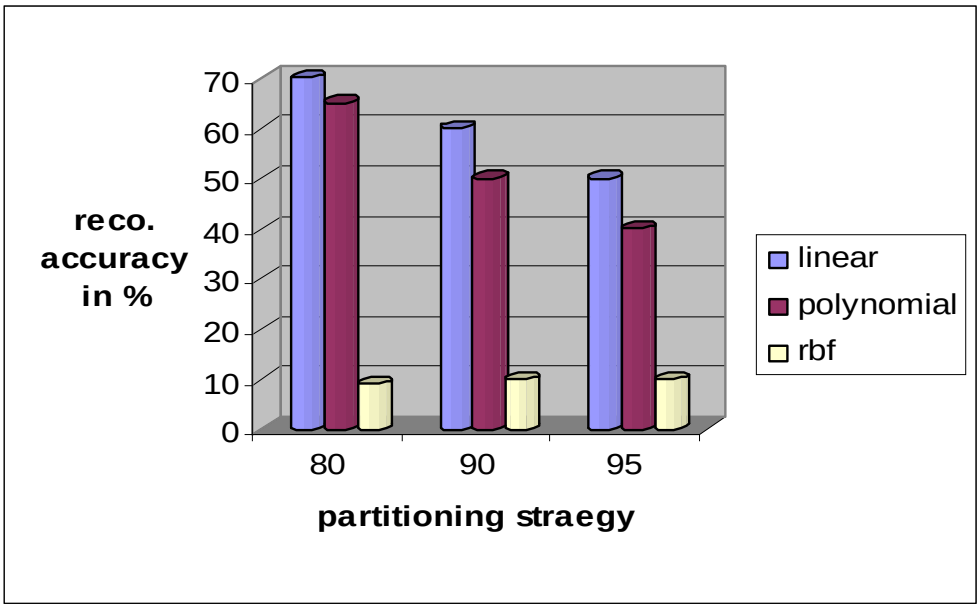


Figure 3.6: Recognition of numerals with 50 resampled points

c. Recognition with 60 resampled points

In this case, we have used 60 co-ordinates for each of the 200 data points collected from the user. As such we have a 120 dimension feature vector that is fed to SVM. The patterns are partitioned into three strategies and in each strategy, three kernel functions of SVM are used. The results obtained by this recognition are shown in table 3.3 and Fig. 3.7 depicts these results graphically.

Table 3.3: Recognition accuracy with 60 resampled points

PARTITIONING / KERNEL FUNCTIONS / STRATEGY	80% TRAINING 20% TESTING	90% TRAINING 10% TESTING	95% TRAINING 5% TESTING
Linear	68.2927%	60%	50%
Polynomial	60%	50%	40%
RBF	10%	9%	10%

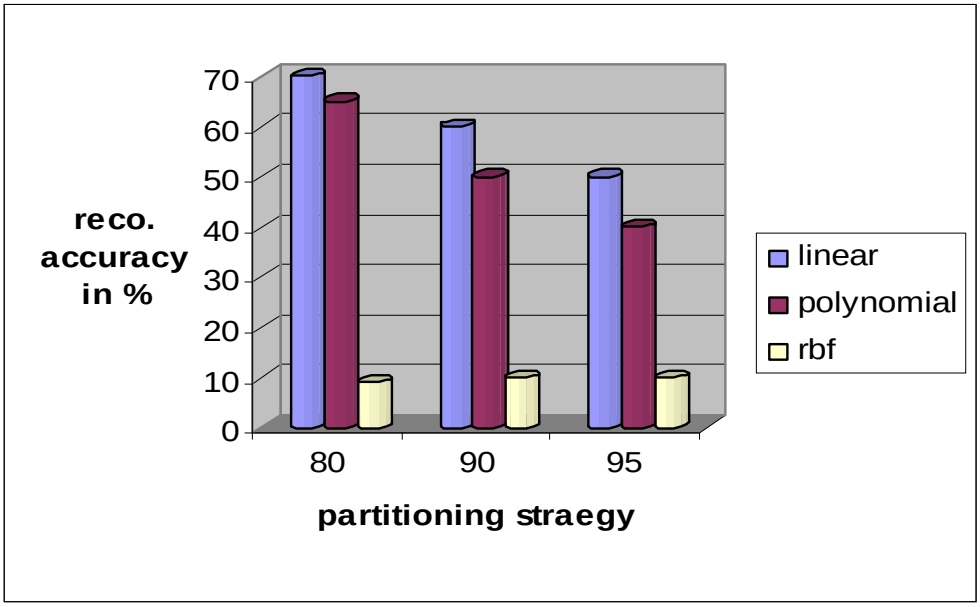


Figure 3.7: Recognition of numerals with 60 resampled points

d. Recognition with 70 resampled points

In this case, we have used 70 co-ordinates for each of the 200 data points collected from the user. As such we have a 140 dimension feature vector that is fed to SVM. The patterns are partitioned into three strategies and in each strategy, three kernel functions of SVM are used. The results obtained by this recognition are shown in table 3.3 and Fig. 3.7 depicts these results graphically.

Table 3.4: Recognition accuracy with 70 resampled points

PARTITIONING STRATEGY / KERNEL FUNCTIONS	80% TRAINING 20% TESTING	90% TRAINING 10% TESTING	95% TRAINING 5% TESTING
Linear	67.5%	60%	40%
Polynomial	60%	55%	30%
RBF	10%	10%	10%

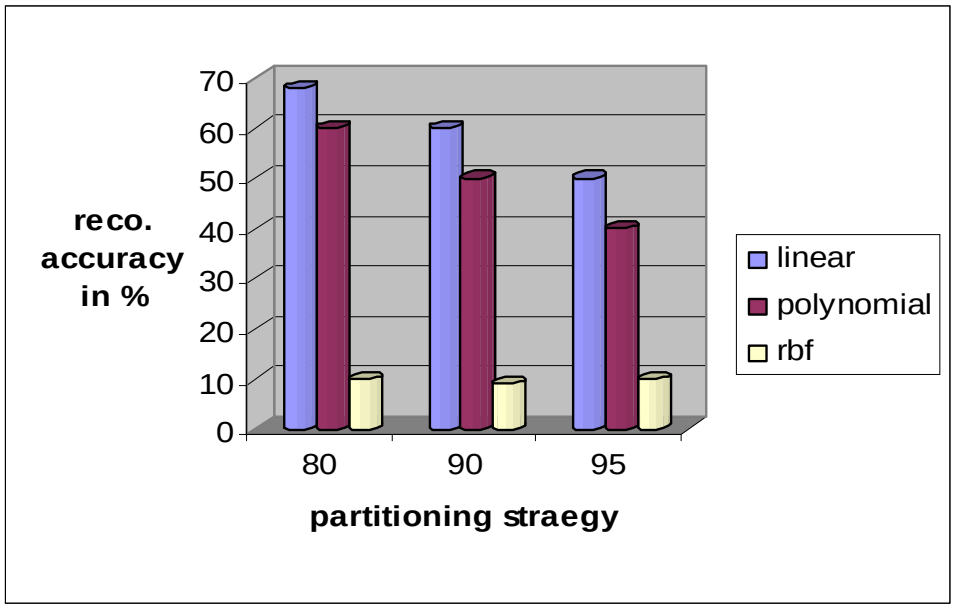


Figure 3.8: Recognition of 70 resampled points

Chapter 4

Conclusions and Future Scope

4.1 Results and Discussions

The main objective of this thesis was to develop an online handwritten Gurmukhi numerals recognition system. We have recognized online handwritten Gurmukhi numerals using three kernel functions of SVM.

In Chapter 1. we have discussed the history of character recognition, advantages and limitations of online handwritten character recognition over offline character recognition, literature review of online handwriting recognition and overview of Gurmukhi numerals. It has helped us in providing the background knowledge of online handwriting recognition.

In chapter 2, we have discussed the procedure of collecting the co-ordinates of input handwritten numeral and various algorithms of pre processing. We have implemented the algorithms of pre processing using Java programming. Algorithm discussed in section 2.3.1 normalizes the character to the size 200×200. It transforms every stroke to the standard size. The algorithm discussed in section 2.3.2 interpolates the missing points between the two consecutive points that have a distance greater than 1. We have used the cubic Bezier curve to interpolate the missing points. This algorithm is necessary in the situations where the hardware is not able to catch up with the speed of writing or cannot capture all the points of the trajectory.

The algorithm discussed in section 2.3.3 accomplishes the resampling of points. It helps in providing the same spatial distance between the captured points. In the last algorithm discussed in section 2.3.4 smoothing of the curve is done. It helps in removing the flickers present in handwriting.

All these algorithms are implemented using Java programming.

In Chapter 3, the data obtained after pre-processing is used for recognition. The recognition is done using three kernel functions of SVM. The recognition percentages obtained on the different strategies of data are shown in the previous chapter. In case of 40 resampled points ,the highest recognition accuracy obtained is 97.5%. It is obtained in the case when we picked 80% data in training and 20% data in testing and is recognised by linear kernel function of SVM. The second highest recognition accuracy is 87%, when same data is recognised by polynomial kernel function of SVM. . In case of 50 resampled points, the highest recognition accuracy obtained is 70.7517%. It is obtained in the case when we picked

80% data in training and 20% data in testing and is recognised by linear kernel function of SVM. The second highest recognition accuracy is 65%, when same data is recognised by polynomial kernel function of SVM. . In case of 60 resampled points, the highest recognition accuracy obtained is 68.2927%. It is obtained in the case when we picked 80% data in training and 20% data in testing and is recognised by linear kernel function of SVM. The second highest recognition accuracy is 60%, when same data is recognised by polynomial kernel function of SVM. . In the last case of 70 resampled points ,the highest recognition accuracy obtained is 67.5%. It is obtained in the case when we picked 80% data in training and 20% data in testing and is recognised by linear kernel function of SVM. The second highest recognition accuracy is 60%, when same data is recognised by polynomial kernel function of SVM. The highest accuracy achieved among all the cases is 97.5%.

4.2 Conclusions and Future scope

The main goal of this thesis was to develop an online handwritten gurmukhi numeral recognition system. This thesis describes the pre-processing steps and the recognition process. Pre-processing is an important phase to increase the efficiency of character recognition system. The main highlights of this project is that all the programming of pre-processing algorithms is done in Java and also the output obtained from pre processing is directly used for recognition. The recognition is done using three kernel functions of SVM and the results obtained are reasonably good with an accuracy of 97.5% when linear kernel is used.

Future Work

The present work can be extended in following directions.

a. One can include the alphanumeric characters in the database for their recognition and thus the online handwritten alphanumeric character recognition can be done. One can also include the Gurmukhi words and thus the recognition of Gurmukhi words can be done in future.

b. A variety of approaches have been discussed by the researchers but no recognition is 100% accurate. The recognition accuracy in this thesis is reasonably good. It can be further improved by introducing new features in Gurmukhi script.

c. The complexities of pre-processing algorithms can further be reduced in order to increase the recognition percentage and to decrease the overall recognition time of the stroke.

d. There is a lot of scope of future enhancements if the implementation of other pre-processing techniques is done i.e word rotation, slant correction, feature extraction etc.

e. The results achieved in the present study motivate to extend the present work with SVM as recognition method. There is always a scope of increase in recognition rate with increase in database when SVM has been used as recognition method.

f. This work has been implemented after collecting the data by one user. As such the results obtained are specific to that user. We can extend the process by collecting the data from different users.

APPENDIX

The code given below helps in data collection, pre-processing (normalization, interpolation, smoothing, resampling of points) and stores the co-ordinates, captured during pen movements in a .txt file.

```
import java.io.*;

import java.awt.*;

import java.awt.event.*;

import java.applet.*;

 // creating a frame window

  public class compare4 extends Frame {

      String mousemsg=" "  ;

      int mouseX, mouseY, t;

      static int i;

      int x[]=new int[500];

      int y[]=new int[500];

      int minX,minY,maxX,maxY;

public compare4() {

     addMouseMotionListener (new MyMouseMotionAdapter(this));

     addMouseListener (new MyMouseAdapter(this));

     addWindowListener( new MyWindowAdapter());

                minX=300;maxX=0;minY=300;maxY=0;

                mousemsg =".";

                i=1;t=1;

}

public void paint(Graphics g) {

 //create rectangles to divide the frame into different parts

 g.drawString("writing area=300*300",20,50);

 g.drawString("bezier=300*300",615,50);
```

```
g.drawString("normalization=300*300",315,50);
g.drawString("resampling=300*300",915,50);
g.drawString("smoothing=300*300",615,400);
 g.drawRect(0,0,300,300);
g.drawRect(300,0,300,300);
g.drawRect(600,400,300,300);
g.drawRect(600,0,300,300);
g.drawRect(900,0,300,300);
  }
//create the window
public static void main ( String args[])   {
      compare4 appwin = new compare4() ;
      appwin.setSize(new Dimension(1200,800));
      appwin.setTitle("an awt based app");
      appwin.setVisible(true);
  } }
class MyMouseMotionAdapter extends MouseMotionAdapter {
   compare4 appwindow;
   public MyMouseMotionAdapter(compare4 appwindow) {
   this.appwindow=appwindow;
   }
  public void mouseDragged(MouseEvent me) {
Graphics g=appwindow.getGraphics();
          appwindow.mouseX=me.getX();
          appwindow.x[appwindow.i]=appwindow.mouseX;
          appwindow.mouseY=me.getY();
          appwindow.y[appwindow.i]=appwindow.mouseY;
```

//find min and max of the stroke

```
if(appwindow.x[appwindow.i]<appwindow.minX)

appwindow.minX=appwindow.x[appwindow.i];

if(appwindow.x[appwindow.i]>appwindow.maxX)

appwindow.maxX=appwindow.x[appwindow.i];

 if(appwindow.y[appwindow.i]<appwindow.minY)

  appwindow.minY=appwindow.y[appwindow.i];

  if(appwindow.y[appwindow.i]>appwindow.maxY)

  appwindow.maxY=appwindow.y[appwindow.i];

System.out.println("x,y,maxx,minx,maxy,miny="+appwindow.i+","+appwindow.x[appwindow.i] + ","
+appwindow.y[appwindow.i]+ "," +appwindow.maxX+ "," +appwindow.minX+","
+appwindow.maxY+ "," +appwindow.minY);

appwindow.mousemsg=".";

g.setColor(Color.red);

g.drawString(".",(int)(appwindow.x[appwindow.i]),(int)(appwindow.y[appwindow.i]));

 appwindow.i++;

  }}

class MyMouseAdapter extends MouseAdapter{

        double a[]=new double[5000];

        double b[]=new double[5000];

        double xnew[]=new double[5000];

        double ynew[]=new double[5000];

double a1[]=new double[150];

double b1[]=new double[150];

double a2[]=new double[150];

double b2[]=new double[150];

int k, j=0,flag=1,i,len=0,c,num;

double x1,y1;

String strFilePath = "C://drag//a11.txt";

String strFilePath1 = "C://drag//num.txt";
```

```
    FileOutputStream fos, fos1;
    DataOutputStream  dos ;
    FileInputStream  fin, fin1 ;
    DataInputStream din, din1 ;
    PrintStream pf, pf1;
 compare4 appwindow;
         public MyMouseAdapter(compare4 appwindow)
           {
                this.appwindow=appwindow;
           }
   public void mouseReleased(MouseEvent me)
      {
          Graphics g=appwindow.getGraphics();
           if(flag==1)
           normalize();
  // points after draging and normalization
       for(k=1;k<appwindow.i;k++)
          {
               g.setColor(Color.black);
          g.drawString(".",appwindow.x[k],appwindow.y[k]);
          g.drawString("*",(int)(xnew[k]),(int)(ynew[k]));
         }
// points for bezier
  for(k=1;k<=j;k++)
    g.drawString("*",(int)(a[k]),(int)(b[k]));
// points after resampling and smoothing
     for(i=1;i<=len;i++)
       {
```

```
            g.drawString("*",(int)(a1[i]),(int)(b1[i]));
            g.drawString("*",(int)(a2[i]-300),(int)(b2[i]+400));
      }
 }
 void normalize()
{       double u,c;
        Graphics g=appwindow.getGraphics();
         int p,m=0,need=0,left,n1;
         double dis[]=new double[1000];
         double x;
//normalize the stroke into 200*200
       for(k=1;k<appwindow.i;k++)
         {
xnew[k]=((195.0/(appwindow.maxX-appwindow.minX))*(appwindow.x[k]-
appwindow.minX ));
         xnew[k]=xnew[k]+300;
 ynew[k]=((195.0/(appwindow.maxY-appwindow.minY))*(appwindow.y[k]-  appwindow.minY));
         ynew[k]=ynew[k]+100;
     g.setColor(Color.blue);
     g.drawString("*",(int)(xnew[k]),(int)(ynew[k]));
  //   System.out.println("val of norm="+xnew[k]+","+ynew[k]);
         }
// distances b/w points
for(j=1;j<=appwindow.i-2;j++)
    {
x=Math.abs ( (xnew[j+1]-xnew[j])* (xnew[j+1]-xnew[j]) +(ynew[j+1]-ynew[j])*(ynew[j+1]-ynew[j])) ;
       dis[j]= Math.sqrt (x );
//System.out.println("dis="+j+"="+dis[j]+","+xnew[j]+","+xnew[j+1]+","+ynew[j]+","+ynew[j+1]) ;
```

```
    }
  // bezier point interpolation of the stroke
j=1;
for(k=1;k<=appwindow.i-3;k++)
    {
        a[j]=xnew[k]+300; b[j]=ynew[k];
         // System.out.println("val of j="+j+"="+a[j]+","+b[j]);
          j++;
        if(dis[k]>1)
      {
         if (k==1)
          {
            for(u=0;u<=1;u=u+0.2)
            {
x1=((1-u)*(1-u)*(1-u)*xnew[1]+3*(1-u)*(1-u)*u*xnew[2]+3*(1-u) * u* u* xnew[3] +
u*u*u*xnew[4]);
y1=((1-u)*(1-u)*(1-u)*ynew[1]+3*(1-u)*(1-u)*u*ynew[2]+3*(1-u)* u*u* ynew[3] + u* u* u *
ynew[4]);
              fun();
} }
else
   {
    for(u=0;u<=1;u=u+0.2)
     {
 x1=((1-u)*(1-u)*(1-u)*xnew[k-1]+3*(1-u)*(1-u)*u*xnew[k]+3*(1-u)* u*u* xnew[k+1]+ u* u*u
*xnew[k+2]);
y1=((1-u)*(1-u)*(1-u)*ynew[k-1]+3*(1-u)*(1-u)*u*ynew[k]+3*(1-u)* u*u* ynew[k+1]+ u*
u*u*ynew[k+2]);
          fun();
```

```
        } }}}
a[j]=xnew[appwindow.i-2]+300;
b[j]=ynew[appwindow.i-2];
        j++;
       if (dis[appwindow.i-2]>1)
       {
        for(u=0;u<=1;u=u+0.2)
{
x1=((1-u)*(1-u)*(1-u)*xnew[appwindow.i-4]+3*(1-u)*(1-u)*u*xnew[appwindow.i-3]+3* (1- u)
*u*u*xnew[appwindow.i-2]+u*u*u*xnew[appwindow.i-1]);
y1=((1-u)*(1-u)*(1-u)*ynew[appwindow.i-4]+3*(1-u)*(1-u)*u*ynew[appwindow.i-3]+3*(1-
u)*u*u*ynew[appwindow.i-2]+u*u*u*ynew[appwindow.i-1]);
          fun();
              } }
a[j]=xnew[appwindow.i-1]+300;
b[j]=ynew[appwindow.i-1];
g.setColor(Color.red);
 for(k=1;k<=j;k++)
    {
      g.drawString("*",(int)(a[k]),(int)(b[k]));
     System.out.println("val of k="+k+"="+a[k]+b[k]);
     }
System.out.println("ist case total no of points in bezier="+j);
 flag=0;
System.out.println("enter no of points that u want to take");
 // resampling  of points
  BufferedReader buf= new BufferedReader ( new InputStreamReader(System.in));
 String data =null;
     try
```

```
        {
          data=buf.readLine();
          len=Integer.parseInt(data);
         }
catch(Exception e)
{        }
if(j<len)
System.out.println("points are less than len="+len);
if(j==len)
for(k=1;k<=j;k++)
g.drawString("*",(int)(a[k]+600),(int)(b[k]));
if(j>len)
{
n1=(int)(j/len);
p=1;
 a1[p]=a[1]+300;
 b1[p]=b[1];
 k=1;
 System.out.println("after resam.="+p+"="+a1[p]+","+b1[p]+",index"+k);
System.out.println("need,left"+(len-1)+","+(j-k));
while(k<j)
{
System.out.println("  ");
System.out.println("ist time");
 k=k+n1+1;
 if(k>=j)
break;
 p++;
```

```
need=len-p;
left=j-k;
 if(need==left)
 break;
if(need>left)
     break;
a1[p]=a[k]+300;
 b1[p]=b[k];
    System.out.println(" after resam.="+p+"="+a1[p]+","+b1[p]+", index"+k);
    System.out.println("need,left"+need+","+left);
    System.out.println("    ");
   System.out.println(" 2nd time");
     k=k+n1+2;
     if(k>=j)
      break;
       p++;
     need=len-p;
     left=j-k;
  if(need<left)
{
      a1[p]=a[k]+300;
       b1[p]=b[k];
    System.out.println(" after resam.="+p+"="+a1[p]+","+b1[p]+", index"+k);
    System.out.println("need,left"+need+","+left);
     }
if(need>left)
       {
           k=k-n1-2;
```

```
            k=k+n1+1;
            left=j-k;
     if(need>left)
     break;
 a1[p]=a[k]+300;
 b1[p]=b[k];
System.out.println(" after resam.="+p+"="+a1[p]+","+b1[p]+", index"+k);
System.out.println("need,left"+need+","+left);
break;
     }
 if(need==left)
    break;
}
// need=len-p;
int count=j-need;
if(count==k)
  {
      count++;
      p++;
  }
while(count<=j)
{
   a1[p]=a[count]+300;
   b1[p]=b[count];
System.out.println(" after resam.="+p+"="+a1[p]+","+b1[p]+",index"+count);
    p++;
   count++;
}}
```

```
 smoothing();
 file();
void file()
{
    int m=1,n=2;
    try
 {
    //create FileInputStream object
          fos = new FileOutputStream(strFilePath,true);
  pf=new PrintStream(fos);
  fin = new FileInputStream(strFilePath);
  din = new DataInputStream(fin);
 fin1 = new FileInputStream(strFilePath1);
  din1 = new DataInputStream(fin1);
    if(din1.available() !=0)
          num=Integer.parseInt(din1.readLine());
           fos1 = new FileOutputStream(strFilePath1);
           pf1=new PrintStream(fos1);
c=num/10;
 num++;
              din1.close();
              pf1.print(num);
              pf.print(c);
for(int i=1;i<=len;i++)
     {
       pf.print(" ");
       pf.print(m);
       pf.print(':');
```

```
          pf.print((int)a1[i]);
          pf.print(" ");
          pf.print(n);
          pf.print(':');
         pf.print((int)b1[i]);
           m=m+2;
           n=n+2;
       }
pf.print('\n');
             pf.close();
             pf1.close();
System.out.print( "points start");
while(din.available() !=0)
{
  //print file to screen
    System.out.println(din.readLine());
}
 din.close();
}
catch(FileNotFoundException fe)
  {
     System.out.println("FileNotFoundException : " + fe);
  }
 catch(IOException ioe)
    {
      System.out.println("IOException : " + ioe);
    }}
void smoothing()
```

```
{
  double m1,m2,b,ang;
  Graphics g=appwindow.getGraphics();
   for(j=3;j<=len-2;j++)
       {
          if((a1[j]-a1[j-2])==0)
                  m1=(b1[j]-b1[j-2])/0.01;
          else
                  m1=(b1[j]-b1[j-2])/(a1[j]-a1[j-2]);
            if((a1[j]-a1[j+2])==0)
                  m2=(b1[j]-b1[j+2])/0.01;
           else
                  m2=(b1[j]-b1[j+2])/(a1[j]-a1[j+2]);
                        System.out.println("ang"+m1+","+m2);
        b=(m1+m2)/(1-m1*m2);
             ang=(Math.atan(b)*180)/Math.PI;
                if(ang<0)
                  ang=360+ang;
       //   System.out.println("ang="+ang);
   a2[j]=(a1[j-2] + a1[j-1] + ang * a1[j] +a1[j+1]+a1[j+2])/(2.0*2.0+ang) ;
   b2[j]=(b1[j-2] + b1[j-1] + ang * b1[j] +b1[j+1]+b1[j+2])/(2.0*2.0+ang) ;
     }
      a2[1]=a1[1]; b2[1]=b1[1];
      a2[2]=a1[2]; b2[2]=b1[2];
              a2[len-1]=a1[len-1]; b2[len-1]=b1[len-1];
             a2[len]=a1[len];     b2[len]=b1[len];
   g.setColor(Color.blue);
   for(k=1;k<=len;k++)
```

```
        {
         //   System.out.println("smoothing points="+a2[k]+","+b2[k]);
        }}
void  fun()
  {
     if (xnew[k+1] >xnew[k])
   {
       if (x1>xnew[k] && x1<xnew[k+1] )
         {  a[j]=x1+300;
           b[j]=y1;
        //   System.out.println("val of j="+j+"="+a[j]+","+b[j]);
           j++;
          }}
if(xnew[k+1]< xnew[k])
        {
         if(x1<xnew[k] && x1>xnew[k+1])
        {    a[j]=x1+300;
             b[j]=y1;
             j++;
        }}
if ( (xnew[k+1]== xnew[k]) && ynew[k+1] > ynew[k])
         {
          if ( y1<ynew[k+1] && y1>ynew[k])
            {
               a[j]=x1+300;
               b[j]=y1;
               j++;
             }}
```

```
 if( (xnew[k+1]== xnew[k]) && ynew[k+1]< ynew[k])
        {
          if(y1 >ynew[k+1] && y1< ynew[k])
            {
             a[j]=x1+300;
             b[j]=y1;
             j++;
             }}}}
class MyWindowAdapter extends WindowAdapter {
public void windowClosing(WindowEvent we) {
System.exit(0);  }}
```

References

[1] J. Mantas, "An Overview of Character Recognition Methodologies", Pattern *Recognition*, vol.19, no.6, pp. 425-430, 1986.

[2] L. D. Earnest, "Machine Reading of Cursive Script", in Proc. IFIP Congress, pp.462-466, Amsterdam, 1963.

[3] K. Mori, I. Masuda, "Advances in recognition of Chinese characters", in Proc. 5th Int. Conf. Pattern Recognition, Miami, pp.692-702 1980.

[4] El-Sheikh, R. M. Guindi, "Computer Recognition of Arabic Cursive Scripts", Pattern Recognition, vol.21, no.4, pp.293-302, 1988.

[5] P. D. Gader, M. A. Khabou, "Automatic Feature Generation for Handwritten Digit Recognition", IEEE Trans. Pattern Analysis and Machine Intelligence, vol.18, no.12, pp.1256-1262, 1996.

[6] S. Mori, K. Yamamoto, M. Yasuda, "Research on Machine Recognition of Handprinted Characters", IEEE Trans. Pattern Analysis, Machine Intelligence, vol.6, no.4, pp.386-404, 1984.

[7] C. Y. Suen, M. Berthod, S. Mori, "Automatic Recognition of Handprinted Characters - The State of the Art", Proc. of the IEEE, vol. 68, no. 4, pp. 469 - 487, 1980.

[8] V. K. Govindan, A. P. Shivaprasad, "Character recognition- A review", Pattern Recognition vol.23, no.7, pp.671-683, 1990.

[9] R. M. Bozinovic, S. N. Srihari, "Off-line Cursive Script Word Recognition", IEEE Trans. Pattern Analysis and Machine Intelligence, vol.11, no.1, pp.68-83, 1989.

[10] Q. Tian, P. Zhang, T. Alexander, Y. Kim. "Survey: Omnifont Printed Character Recognition", Visual Communication and Image Processing 91: Image Processing, pp. 260 - 268, 1991.

[11] M. Shridhar, A. Badreldin, "High Accuracy Syntactic Recognition Algorithm for Handwritten Numerals", IEEE Trans. Systems Man and Cybernetics, vol.15, no.1, pp.152 - 158, 1985.

[12] M. Tayli, A I. Ai-Salamah, "Building Bilingual Microcomputer System" *Communications of the ACM*, vol.33, no.5, pp.495-504, 1990.

[13] A. Belaid, J.P. Haton, "A Syntactic Approach for Handwritten Mathematical Formula Recognition", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.6, pp. 105-111, 1984.

[14] S. Mori, C. Y. Suen, K. Yamamoto, "Historical Review of OCR Research and Development", *IEEE-Proceedings*, vol.80, no.7, pp.1029- 1057, 1992.

[15] C. Y. Suen, C. C. Tappert, T. Wakahara, "The State of the Art in On- Line Handwriting Recognition", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, no. 8, pp.787-808, 1990.

[16] I. Bazzi, R. Schwartz, J. Makhoul, "An Omnifont Open Vocabulary OCR System for English and Arabic", *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol.21, no.6, pp.495-504, 1999.

[17] H. I. Avi-Itzhak, T. A. Diep, H. Gartland, "High Accuracy Optical Character Recognition Using Neural Network with Centroid Dithering", *IEEE Trans. Pattern Analysis and Machine Intelligence,*vol.17, no.2, pp.218-228, 1995.

[18] A. Meyer, "Pen Computing: A Technology Overview and A Vision", *SIGCHI Bulletin*, vol.27, no.3, pp.46-90, 1995.

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

[20] R. Plamandon, D. Lopresti, L. R. B. Schomaker, R. Srihari, "On-line Handwriting Recognition", *Encyclopedia of Electrical and Electronics Eng*., J. G. Webster, ed., vol.15, pp.123-146, New York, Wiley, 1999.

[21] Nafiz Arica, Student Member, IEEE and Fatos T. Yarman-Vural, Senior Member, IEEE on An overview of character recognition focused on offline handwriting.

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

[23] Suen, C. Y., Koerich, A. L. and Sabourin, R., 2003. 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.

[24] Govindaraju, V., Srihari, S. N., 1997. Paradigms in handwriting recognition. Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, pp. 1498 - 1503.

[25] Subrahmonia, J., Zimmerman, T., 2000. Pen computing challenges and applications. Proceedings of 15th International conference on Pattern Recognition , on vol 2, pp. 60-66.

[26] Beigi, H., Nathan, K., Clary, G. J., and Subhramonia, J., 1994. Size normalization in unconstrained online handwritng recognition, Proceedings ICIP, pp. 169-173.

[27] Unser, M., Aldroubi, A., Eden, M., 1993. B-Spline signal processing: part II - efficient design and applications. IEEE Transactions on Signal Processing, vol. 41, no. 2, pp. 834-848.

[28] Kavallieratou, E., Fakatakis, N., Kolkkinakis, G., 2002. An unconstrained handwriting recognition system. International Journal of Document Analysis and Recognition, vol. 4, no. 4, pp. 226-242.

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

[30] Slavik, P., Govindaraju, V., 2001. 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.

[31] Yimei, D., Fumitika, K., Yasuji, M., Malayappan, S., 2000. Slant estimation for handwritten words by directionally refined chain code. Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition, pp. 53-62.

[32] Guerfali, W and Plamondon, R, 1993. Normalizing and restoring online handwriting. Pattern Recognition, vol. 26, no. 3, pp. 419.

[33] Bellegarda, E. J., Bellegarda, J. R., Namahoo, D. and Nathan K. S., 1993. A probabilistic framework for online handwriting recognition. Proceedings of IWFHR III, pp. 225-234.

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

[35] Ragot, N. and Anquetil, E., 2003. A generic hybrid classifier based on hierarchical fuzzy modeling: experiments on online handwritten character recognition. Proceedings of International Conference on Document Analysis and Recognition, pp. 963-967.

[36] Funanda, A., Muramatsu, D., Matsumoto, T., 2004. The reduction of memory and the improvement of recognition rate for HMM on-line handwriting recognition. Proceedings of IWFHR, pp. 383-388.

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

[38] Basu, M., Bunke, H. and Bimbo, A. D., 2005. Guest editors' introduction to the special section on syntactic and structural pattern recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 7, pp. 1009-1012.

[39] Schomaker, L., 1993. Using stroke or character based self-organizing maps in the recognition of online, connected cursive script. Pattern Recognition, vol. 26, no. 3, pp. 443-450

[40] Morasso, P. G., Limoncelli, M. and Morchio, M., 1995. Incremental learning experiments with SCRIPTOR: an engine for online recognition of cursive handwriting. Machine Vision and Applications, 8, pp. 206-314.

[41] Cho, S., 1997. Neural-network classifiers for recognizing totally unconstrained handwritten numerals. IEEE Transactions on Neural Networks, vol. 8, no. 1, pp. 43-53

[42] Shintani, H., Akutagawa, M., Nagashino, H., Kinouchi, Y., 2005. Recognition mechanism of a neural network for character recognition. Proceedings of Engineering in Medicine and Biology 27th Annual Conference, pp. 6540-6543.

[43] Tappert, C. C., 1982, Cursive script recognition by elastic matching. IBM Journal of Research and Development, vol. 26, no. 6, pp. 765-771.

[44] Tappert, C. C., 1991. Speed, accuracy, and flexibility trade-offs in online character recognition. International Journal of Pattern Recognition and Artificial Intelligence, vol. 5, No. 1/2, pp. 79-95,

[45] Scattolin, P., 1995, Recognition of handwritten numerals using elastic matching, Master's thesis, Concordia University, Canada.

[46] Pavlidis, I., Singh, R. and Papanikolopoulos, N. P., 1997. An online handwritten note recognition method using shape metamorphosis. Proceedings of fifth International Conference on Document Analysis and Recognition, vol. 2, pp. 914-918.

[47] Flusser, J., Boldy, J. and Zitova, B., 2003. Moment forms invariant to rotation and blur in arbitrary number of dimensions. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 2. pp. 234-245.

[48] Stefano, L.D., Mattoccia, S. and Tombari, F., 2005. ZNCC-based template matching using bounded partial correlation. Pattern Recognition Letters, vol. 26, no. 14, pp. 2129-2134.