RECOGNITION OF PRINTED BENGALI CHARACTERS

by

DECISION THEORETIC METHOD OF PATTERN RECOGNITION

By

MD. ABDUS SATTAR

A Thesis

submitted to the department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

AUGUST,1990.

RECOGNITION OF PRINTED BENGALI CHARACTERS

by

DECISION THEORETIC METHOD OF PATTERN RECOGNITION

A Thesis

By

MD. ABDUS SATTAR

Accepted as satisfactory as to style and contents for partial fulfillment of the requirements for the degree of M.Sc.Engineering in Computer Science and Engineering on 22-8-90.

DR.SYED MAHBUBUR RAHMAN
Associate Professor and Head,
Computer Science and Engineering Dept.
Bangladesh University of Engineering and
Technology, Dhaka, Bangladesh.

Chairman and Supervisor

DR. MD. SHAMSUL ALAM
Associate Professor,
Computer Science and Engineering Dept.
Bangladesh University of Engineering and
Technology, Dhaka, Bangladesh.

Member

DR. MD. ABDUR RASHID
Associate Professor,
Dept. of Applied Physics and Electronics,
University of Dhaka, Dhaka, Bangladesh.

Member (External)

CERTIFICATE OF RESEARCH

Certified that the work presented in this Thesis is the result of the investigation carried out by the candidate under the supervision of Dr. Syed Mahbubur Rahman at the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

22-06-90
Date

Signature of the Candidate

DECLARATION

I do hereby declare that neither this thesis nor any part thereof has been submitted or is being concurrently submitted in candidature for any degree at any other university.

22-06-20

Date

Signature of the Candidate

ACKNOWLEDGMENTS

Keen interest of Dr. Syed Mahbubur Rahman in the field of application of computer facilities in Bangla language has influenced the author to carry out a research work in the field of optical recognition of Bangla characters . The author expresses his deepest gratitude to Dr. Rahman for his support,advice,valuable and constant guidance and his constant encouragement and supervision to make the idea work .

The author also expresses his heartiest and sincerest gratitude to Dr. K. M. Waliuzzaman , Director , Bangladesh Institute of Technology , Rajshahi , Bangladesh for his keen interest , suggestions and encouragements for the development of this research work .

Thanks are also due to my friends who make this otherwise bore research work into an enjoyable job. The author also expresses his gratitudes to the personnels of the Microcomputer Laboratory of BUET specially to Mr. Syed Ahsanul Karim, Lab Instructor and to Mr. H. M. Rezaul Karim,Lab Attendant for their friendly helps.

ABSTRACT

A decision theoretic pattern recognition approach for recognizing Bangla alpha-numerics has been adopted. The approach chosen as the basis for the analysis only recognizes the characters not giving any of their structural description.

The scheme operates on the representation of one character at a time. The representation is in the form of a matrix whose elements are '0' and '1' corresponding to 'black' and 'white' in the original picture. The matrix is obtained by optically scanning the picture of the character.

The scheme uses the "template - matching" technique for recognition. The input pattern (with unknown classification) is compared with a set of templates or prototypes, one for each character, designed previously and stored in machine, and the classification is based on a good match with the template.

An interactive menu driven character recognition software in C language was developed to implement the proposed scheme in IBM PC or compitables.

CONTENT

CHAPTER 1

INTRODUCTION

1.1 GENERAL

The problem of pattern recognition usually denotes a discrimination or classification of a set of processes or events [1]. The set of processes or events to be classified could be a set of physical objects or a set of mental states. The number of pattern classes is often determined by the particular application in mind. For example, in the problem of English character recognition, the problem is of 26 classes. On the other hand, discriminating Bangla characters from English ones is a two class problem.

Pattern recognition is a major area of activity that encompasses the processing of pictorial information obtained from interaction between science and society. Pattern recognition is needed for to communicate with the computing machines in human being's natural mode of communication (the human voice and hand written script) [2]. Pattern recognition is also concerned with the idea of designing and making automata that can hear and understand what human being say and write, the automata that can speak and make people understand, and the automata that can process pictorial information for human use with more and more efficiency. The research in pattern recognition encompasses the field of communication and computer science, mathematics and statistics, acoustics, phonetics, linguistics and psycholinguistics, speech pathology, hematology, radiology, neurophysiology, remote sensing techniques and photogrammetry and similar other many aspects.

Research and developments on pattern recognition and applications may be classified into following major groups :

i) Man- Machine communication

ii) Bio-medical applications and diagnosis of pathological conditions by analyzing X-ray and/or cytological slides

iii) Natural resources estimation and planning in agriculture, forestry, hydrology, geological environment etc.

iv) Scientific and military applications

and v) Detection of crime and criminals.

1.2 NECESSITY OF AUTOMATIC CHARACTER RECOGNITION BY MACHINES

Modern high speed computers can execute millions of instructions per seconds, but their capacity as information processing devices is limited by the type of input data that they can accept. Much of the information that we might like the computers to process is in pictorial form; this includes alpha-numeric characters (printed text, hand writing etc.). Alphanumeric data can be manually converted into computer readable form, but much faster input rates can be achieved if scanning and optical recognition techniques are used.

Some applications of character recognition is shown in Table 1-1

1.3 APPROACHES TO PATTERN/CHARACTER RECOGNITION

The many different mathematical techniques used to solve pattern recognition problems may be grouped into two general approaches : namely,

a) the decision theoretic (or statistical) approach and

b) the syntactic (or linguistic) approach.

TABLE 1 - 1

SOME APPLICATION OF CHARACTER RECOGNITION

Problem	input to recognition system	output of recognition system
Bank checks	magnetic response wave form, optical scanned image	numeric characters, special symbols
Automatic process of documents --- utility bills,credit card charges, sale and inventory documents.	optical scanned image	alphanumeric characters, special symbols
Journal tape reading	optical scanned image	alphanumeric characters, special symbols
Page readers --- automatic type setting,input to computers, reading for the blind	optical scanned image	alphanumeric characters, special symbols
Label readers	optical scanned image	alphanumeric characters, special symbols
Address readers	optical scanned image	letters and numerals combined into zip codes, city and street names, and street addresses
Other readers --- license plate readers, telephone traffic counter reader etc.	optical scanned image	alphanumeric characters, special symbols

1.3.1 DECISION THEORETIC APPROACH

In the decision theoretic approach the classification is based on a set of selected measurements, extracted from the input pattern [1,2]. These selected measurements are called 'feature'. The recognition of each pattern (assignment to a pattern class) is usually made by partitioning the feature space. Once a pattern is transformed through feature extraction, to a point or vector in the feature space, its characteristics are expressed only by a set of numerical values. The information about the structure of the pattern is either ignored or not explicitly represented in the feature space. Most of the developments in the pattern recognition research during the sixties deal with decision theoretic approach. Applications include character recognition, crop classification, medical diagnosis, classification of electrocardiograms etc.

A simplified block diagram for decision theoretic approach is depicted in Figure 1.1.

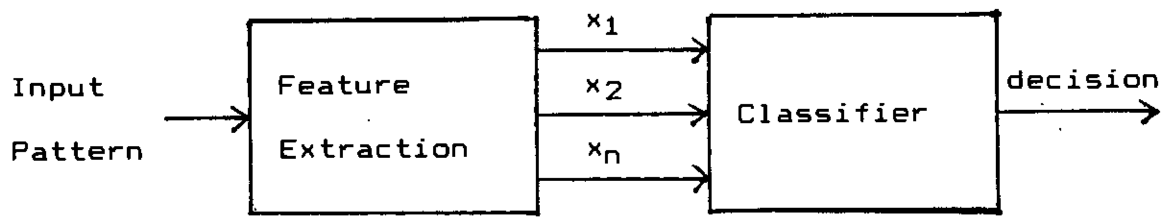

Fig. 1.1 : Block diagram of a pattern recognition system using decision theoretic approach.

1.3.2 SYNTACTIC APPROACH

In some pattern recognition problems, the structural information which describe each pattern is important and recognition process includes not only the capability of assigning the pattern to a particular class (to classify it), but also the capacity to describe specific aspects of the pattern which make it ineligible for assignment to another class [3]. The syntactic approach views patterns as complexes of primitive structural elements, called morphs [4]. A pattern is classified by studying the set of morphs which build up a pattern and studying the relationships among the morphs. This method has been successfully applied in problems of character recognition, chromosome analysis, finger-print classification, X-ray analysis, speech analysis etc.

A simplified block diagram for syntactic pattern recognition method is shown in Figure 1.2.

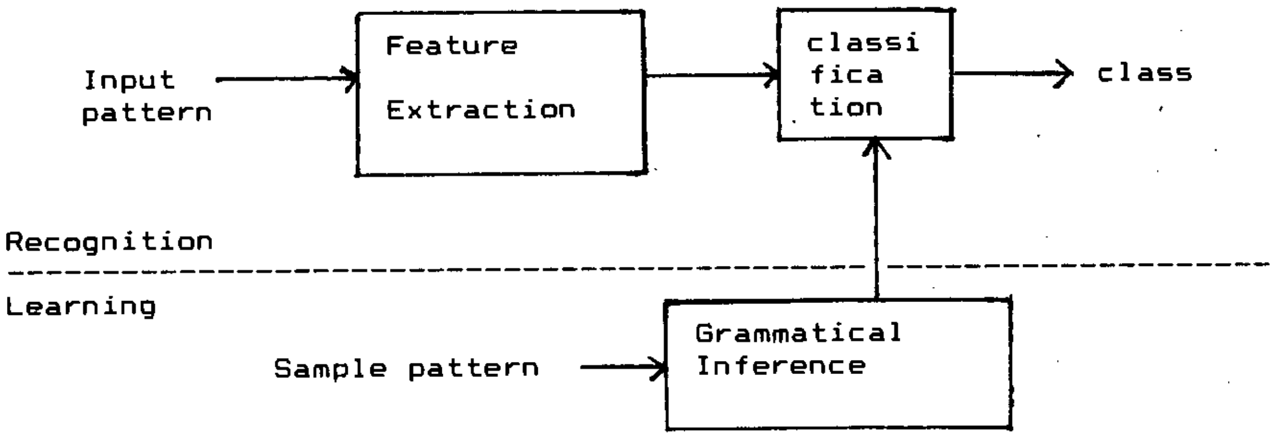

Fig. 1.2 Block diagram of a syntactic pattern recognition problem.

1.4 DICHOTOMY OF SYNTACTIC AND STATISTICAL APPROACHES

In the past much has been made of the apparent difference between two approaches. The stress on the distinction between the two model hides many a similarities:

most of the pre-processing techniques are usefully applied to both the approaches; feature extraction and selection in decision theoretic approach and morphs extraction and selection in syntactic approaches are similar in nature.

The basic difference between the two approaches is that in decision theoretic approach the features are a set of numerical measurements on the pattern/subpattern parameters whereas the morphs in syntactic model are subpatterns themselves. Statistical decision theory focuses entirely on statistical relationships among scalar features ignoring any other structural properties that characterized the pattern i.e. decision theoretic approach only classifies the pattern whereas syntactic approach serves classification as well as description of the pattern.

1.5 A COMPARISON OF ANALOG AND DIGITAL TECHNIQUES FOR PATTERN RECOGNITION

The digital approach to problems in pattern recognition has many advantages [5]. Digital computers provide the user with the capability of performing calculations to essentially any degree of precision with almost infinite flexibility as regards the type and scope of the problem addressed. Due to the universality of most major programming languages and general availability of digital computing facilities, the user also benefits from both ease of programming and the transferability of software. Last but not the least, the digital computer usually offers the user absolute repeatability on each execution of a given program. These are the advantages which led to an almost overwhelming preference for the use of digital computers in carrying out calculations relating to pattern recognition.

Analog computer offers workers using low precision high-speed one or two dimensional linear discriminant analysis a significant advantage in hardware performance (equivalent bits per sec per dollar) over digital computer in certain limited but important areas [5]. These areas include finger print identification, word recognition, chromosome spread detection, earth resources and land use analysis and broad band radar signal analysis. Although at present the analog computer offers significant advantage in certain fields, this advantages will eventually be overcome by digital computer [5].

1.6 LITERATURE SURVEY

Over the years, the field of pattern recognition has attracted workers from a variety of areas such as engineering, system theory, statistics, linguistics, psychology etc., resulting in a vast literature ,containing abstract mathematical approaches as well as highly pragmatic techniques. The literature is scattered in a large number of journals in several fields. At least three IEEE journals (Systems,Man and Cybernetics, Computer and Information Theory) regularly publish pattern recognition papers. Some of the important literatures, relevant to the work, are discussed in this section.

L.N.Kanal in his paper [4] selectively surveyed contributions to major topics in pattern recognition during the period 1968 to 1974. This paper also provides a very useful bibliography about representative books and surveys on pattern recognition published during the above mentioned period. Theoretical models for automatic pattern recognition are contrasted with practical design methodology. He selectively discussed the research contributions to statistical and structural pattern recognition including contributions to error estimation and the experimental design of pattern classifiers. His paper concludes with a representative set of applications of pattern recognition technology.

In his paper [5] Kendall Preston Jr. gave a thorough comparison of analog and digital techniques used for pattern recognition. This paper reviews three major categories (electronic, acoustical and optical) of analog technology used in pattern recognition and predicts the future trends upon the analysis of performance advances which have taken place in both digital and analog fields during the past decade.

Unger [6] observed that for any alphabet there must exist at least one finite set of characteristics that can be used to distinguish amongst the members. He suggested that there must exist a set of yes or no questions such that if these questions are answered with respect to any given figure then there will be only one member of the alphabet to which this figure can belong.

Horwitz and Shelton [7] described a class of techniques for character recognition.These techniques are characterized by the property that the only parameters of the input which are used are those which are independent of the position of the characters, i.e. these techniques are registration invariant.

Perotto [8] described a method, called "morphotopological method for character recognition", that can describe the characters in invariant terms with considerations of topological and morphological type. The method makes it possible of reading english numerical characters having strong variations in form, size and position.

A matrix of simple, identical intercommunicating cells is the core of a pattern classifier described by Glucksman [9]. A test sequencer tests the matrix, on which the pattern is mapped, by following a binary decision tree. The result of each test determines which of two tests will follow. Each test eliminates classes of patterns from the total set of classes. The process lead to one final class at the end of a sequence of such tests.

Yau and Yang [10] suggested simple template-matching pattern recognition technique by using any general purpose associative memory.

A.W.Holt in a paper [11] classifies the character recognition machinery with a minimum reference to the specific components used. According to Holt the job of all such machines is to convert a set of data having high information into a character name having a much lower information content. In his paper he gave a description of single stage, two stage and three stage character recognition machines. He is the first man to use stage concept in classifying character recognition machinery.

Fu, a pioneer in pattern recognition field, in his many papers discusses the different forms of recognition techniques. In a paper [1] he discussed in detail the non-parametric as well as parametric (Bayes) classification methods. The paper also includes a discussion on sequential decision model for pattern classification.

In another paper [18] Fu explores the topics on syntactic approach. Here a detailed discussion on selection on pattern primitives, pattern grammar including special grammar and syntax analysis as recognition procedure could be found.

M. A. Sattar and S. M. Rahman [19] in their paper discuss the different problems of recognition of printed

Bangla characters by applying the Template matching method. They discussed about the optimum number of training set for designing the templates and the mismatch threshold to be used to avoid the problem of mis-classification.

1.7 SCOPE OF THE PRESENT WORK

It has already been mentioned that one of the main aspect of pattern recognition is to communicate with the computing machines in the natural mode of communication (the human voice and handwritten or typed script). Research works are going on in different ways of man-machine communication such as speaker recognition, speech analysis, finger print identification, character recognition etc. Character recognition has been receiving considerable attention as the result of the phenomenal growth of the office automation and the need for translating human language into machine language. Most of the workers in the fields of character recognition used the decision theoretic approach. For the present work decision theoretic approach has been chosen which only classifies the patterns/characters not going into its detailed structure. A computer program has been developed to work in steps, first separating the character from the text, second normalizing i.e. registering the input pattern with top and left margin, third comparing the registered pattern with the previously designed "masks" or "templates" and finally taking the decision to properly classify the unknown input pattern.

CHAPTER 2

DECISION THEORETIC METHODS

2.1 PATTERN AND PATTERN CLASSES

A **pattern** is a quantitative or structural description of an object or some other entity of interest , while a **pattern class** is a set of patterns that share some common properties. For example , in character recognition case, each alpha-numeric character represents a **pattern class** whereas each unknown input to the recognition system is a **pattern.**

2.2 ELEMENTS OF A PATTERN RECOGNITION SYSTEM[12]

The principal function of a pattern recognition system is to arrive at decisions concerning the class membership of the patterns with which it is confronted. Several major information translation processes take place between the time a pattern is input and a decision is made by the system. These processes , which are summarized in block diagram form in Fig. 2.1 , extract from the input data the discriminatory information required for classification. The function of the blocks shown in Fig. 2.1 are described briefly as follows :

The sensor is simply the measurement device that transforms the input patterns into a form suitable for machine manipulation. Although some simple pattern recognition systems operate on the input data directly from the sensor, it is common to follow the sensor with a

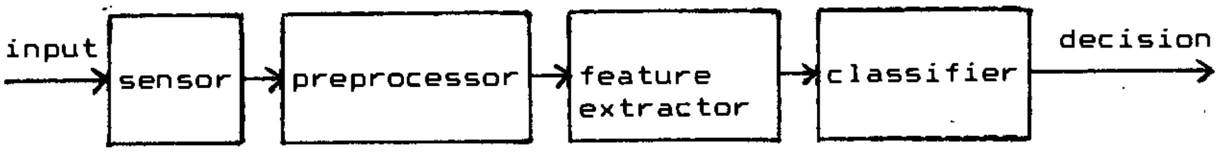

Figure 2.1 Components of a pattern recognition system.

preprocessor and feature extractor. The preprocessor removes unnecessary or corrupting elements from the measured data, while the feature extractor computes from the preprocessed data the features required for classification. Finally, these features are input into the classifier, whose function is to yield a decision concerning the class membership of the pattern being processed.

2.3 DECISION THEORETIC APPROACH TO PATTERN RECOGNITION

In the block diagram of Fig. 2.1 , it has been implicitly assumed that the system "knows" the information processing operations that must be performed on an input pattern in order to arrive at a decision. Although the general form of these operation is specified by the system designer, in most cases each specific operation is characterized by variable parameters that must be adapted to a given pattern recognition problem. The adjustment of these parameters is usually carried out by utilizing sample patterns in what is called a **learning** or **training** process.

Machine learning techniques may be subdivided into two principal categories : (1) **supervised** and (2) **unsupervised**. In a supervised learning situation, the system parameters are estimated by algorithms that utilize training sample patterns whose class membership is specified externally by the system designer. In this manner, the unknown parameters are adjusted to fit a situation where the pattern classes are specified and characterized by representatives samples. Hence, the ultimate success of this approach is dictated by the quality of the sample set used to **train** the pattern recognition system.

The unsupervised learning approach is used when there is little or no *a priori* knowledge about the pattern classes of a given problem. In essence, this approach attempts to extract the pattern classes present in a set of data for which the classification of the available sample patterns is not completely known.

As mentioned earlier, the principal function of a pattern recognition system is to yield decisions concerning the class membership of the pattern with which it is confronted. In order to accomplish this task, it is necessary to establish some rules upon which to base the decisions. One important approach to this problem is the use of decision function . Followings are the discussions of some of the main decision function used in decision theoretic approach of pattern recognition system.

2.3.1 NON-PARAMETRIC DECISION THEORETIC CLASSIFICATION METHODS

The concept of pattern classification may be expressed in terms of the partition of feature space (or mapping from feature space to decision space). Suppose that N features are to be measured from each input pattern . Each set of N features can be considered as a vector **X**, called a feature (measurement) vector, or a point in the N-dimensional feature space W_x. The problem of classification is to assign each possible vector or point in the feature space to a proper pattern class. This can be interpreted as a partition of the feature space into mutually exclusive regions and each region will correspond to a particular pattern class.

Mathematically, the problem of classification can be formulated in terms of "discriminant functions". Let $w_1, w_2, \ldots, w_m$ be designated as the m possible pattern classes to be recognized, and let

$$X = [\ x_1 \quad x_2 \quad \ldots\ldots\ldots \quad x_N\ ]$$

be the feature (measurement) vector where x_i represents the ith feature measurement. Then the discriminant function $D_j(X)$ associated with pattern class w_j, $j = 1,\ldots,m$, is such that if the input pattern represented by the feature vector X is in class w_i, denoted as $X \sim w_i$, the value of $D_i(X)$ must be largest. That is for all $X \sim w_i$

$$D_i(X) > D_j(X), \quad i,j = 1,\ldots,m, \qquad i \neq j. \tag{2.1}$$

Thus, in the feature space W_x the boundary of partition, called the decision boundary, between regions associated with class w_i and class w_j, respectively, is expressed by the following equation.

$$D_i(X) - D_j(X) = 0. \qquad (2.2)$$

Many different forms satisfying condition (2.1) can be selected for $D_i(X)$. Several important discriminant functions are discussed in the following.

2.3.1.1 LINEAR DISCRIMINANT FUNCTION

In this case, a linear combination of the feature measurements $x_1, x_2, \ldots, x_N$ is selected for $D_i(X)$, i.e.,

$$D_i(X) = \sum w_{ik} x_k + w_{i,N+1}, \qquad i = 1, \ldots, m \ .$$

The decision boundary between regions in W_x associated with w_i and w_j is in the form of

$$D_i(X) - D_j(X) = \sum w_k x_k + w_{N+1} = 0 \qquad (2.3)$$

with $w_k = w_{ik} - w_{jk}$ and $w_{N+1} = w_{i,N+1} - w_{j,N+1}$. Equation (2.3) is the equation of a hyperplane in the feature space W_x. If $m = 2$, on the basis of equation (2.3), $i,j = 1,2$ $(i \neq j)$, a threshold logic device, as shown in figure 2.2 can be employed as a linear classifier (a classifier using linear discriminant functions).

From fig.2.2 ,let $D(X) = D_1(X) - D_2(X)$,

if output = +1, i.e., $D(X) > 0$, then $X \sim w_1$

and if output = -1, i.e., $D(X) < 0$, then $X \sim w_2$

For the number of pattern classes more then two, i.e. $m > 2$, several threshold logic devices can be connected in parallel so that the combinations of the outputs from, say, M threshold logic devices will be sufficient for distinguishing m classes, i.e., $2^M >= m$.

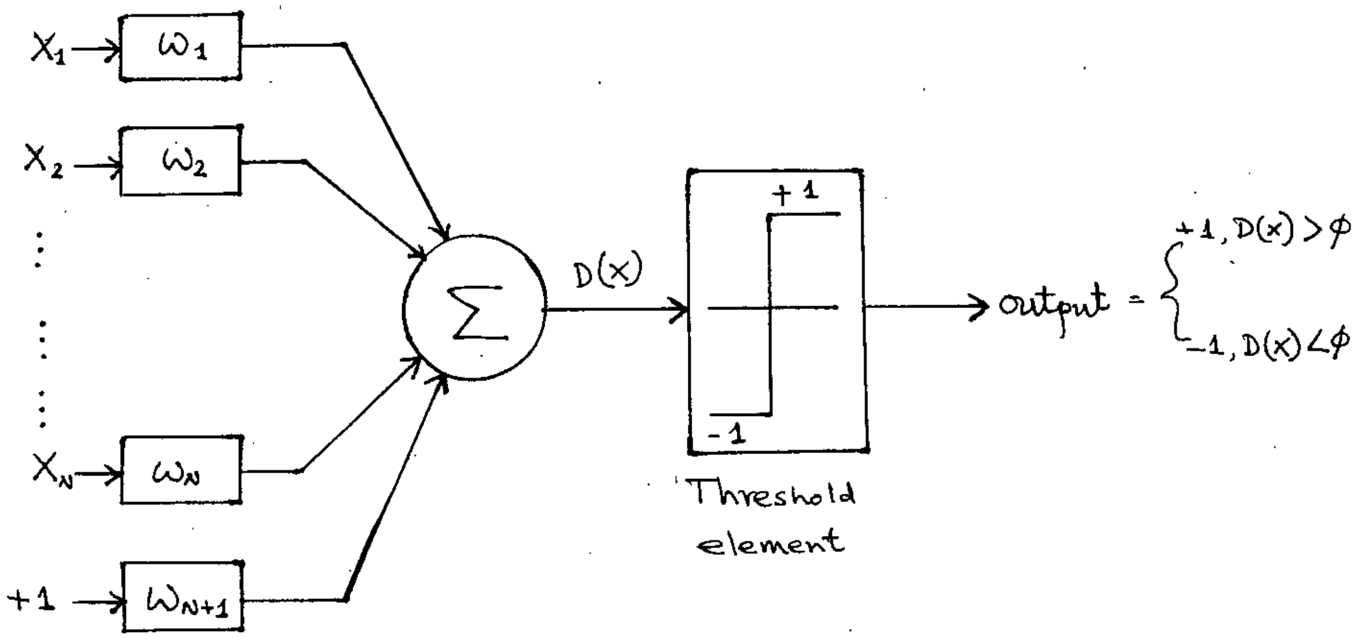

Fig. 2.2 A linear two-class classifier .

2.3.1.2 MINIMUM DISTANCE CLASSIFIER

An important class of linear classifier is that of using the distance between the input pattern and a set of reference vectors or prototype points in the feature spaces as the classification criterion. Suppose that m reference vectors $R_1, R_2, \ldots, R_m$, are given with R_j associated with the pattern class w_j. A minimum-distance classification scheme with respect to $R_1, R_2, \ldots, R_m$ is to classify the input X as from class w_i, i.e.,

$$X \sim w_i \quad \text{if } | X - R_i | \text{ is the minimum.}$$

where $|X-R_i|$ is the distance defined between X and R_i. For example, $|X - R_i|$ may be defined as

$$| X - R_i | = \sqrt{(X - R_i)^T (X - R_i)} \;, \qquad (2.4)$$

where the superscript T represents the transpose operation to a vector. From equation (2.4)

$$|X-R_i|^2 = X^T X - X^T R_i - X R_i^T + R_i^T R_i.$$

Since $X^T X$ is not a function of i, the corresponding discriminant function for a minimum-distance classifier is essentially

$$D_i(X) = X^T R_i + X R_i^T - R_i^T R_i, \qquad i = 1, \ldots, m$$

which is linear. Hence, a minimum distance classifier is, of course, dependent upon an appropriately selected set of reference vectors.

2.3.1.3 NEAREST NEIGHBOR CLASSIFICATION

The concept adopted in Subsection 2.3.1.2 can be extended to the case of minimum-distance classification with respect to sets of reference vectors. Let $R_1, R_2, \ldots, R_m$ be the m sets of reference vectors associated with classes $w_1, w_2, \ldots, w_m$, respectively, and let reference vectors in R_j be denoted as $R_j^{(k)}$, i.e.,

$$R_j^{(k)} \in R_j, \qquad k = 1 \ldots u_j.$$

where u_j is the number of reference vectors in set R_j. Define the distance between an input feature vector X and R_j as

$$d(X, R_j) = \min_{k=1,\ldots,u} |X - R_j^{(k)}|$$

That is, the distance between X and R_j is the smallest of the distances between X and each vector in R_j. The classifier will assign the input to a pattern class which is associated with the closest vector set. If the distance between X and $R_j^{(k)}$, $|X - R_j^{(k)}|$, is defined as Equation (2.4), then the discriminant function used in this case is essentially

$$D_i(X) = \max_{K=1,\ldots,u} \{ X^T R_i^{(k)} + (R_i^{(k)})^T X - (R_i^{(k)})^T R_i^{(k)} \} \quad (2.5)$$

where $i = 1,\ldots,m$

Let

$$D_i^{(k)} = X^T R_i^{(k)} + (R_i^{(k)})^T X - (R_i^{(k)})^T R_i^{(k)}$$

Then

$$D_i(X) = \max_{K=1,\ldots,u} \{ D_i^{(k)}(X) \} ; \qquad i = 1,\ldots,m \qquad (2.6)$$

It is noted that $D_i^{(k)}(X)$ is a linear combination of features, hence, the class of classifiers using (2.5) or (2.6) is often called the Nearest Neighbor classifiers.

2.3.1.4 POLYNOMIAL DISCRIMINANT FUNCTIONS

An rth-order polynomial discriminant function can be expressed as

$$D_i(x) = W_{i1} f_1(x) + W_{i2} f_2(x) + \ldots + W_{iL} f_L(x) + W_{i,L+1}$$

where $f_j(X)$ is of the form

$$X_{K_1}^{n_1} X_{K_2}^{n_2} \cdots X_{K_r}^{n_r} \quad \text{for} \quad \begin{array}{l} K_1, K_2, \ldots, K_r = 1, \ldots, N \text{ and} \\ n_1, n_2, \ldots, n_r = 0 \text{ } \varepsilon' \text{ } 1. \end{array}$$

The decision boundary between any two classes is also in the form of an rth-order polynomial. Particularly, if $r = 2$, the discriminant function is called a quadric discriminant function . In this case,

$$f_i(x) = x_{k_1}^{n_1} \, x_{k_2}^{n_2} \quad \text{for } k_1, k_2 = 1, \ldots, N \text{ and } n_1, n_2 = 0 \,\&\, 1.$$

and

$$L = \frac{1}{2} N(N+3).$$

Typically,

$$D_i(X) = \sum_{k=1}^{N} W_{kk} X_k^2 + \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} W_{jk} X_j X_k + \sum_{j=1}^{N} W_j X_j + W_{L+1}$$

In general, the decision boundary for quadric discriminant functions is a hyper-hyperboloid. Special cases include hypersphere, hyperellipsoid and hyperellipsiodal cylinder.

2.4 TEMPLATE MATCHING TECHNIQUE

When a pattern class is characterized by a roster of its members, the design of a pattern recognition system may be based on the member-ship-roster concept. Characterization of a pattern class by a roster of its member suggests automatic pattern recognition by **Template matching** [1,13]. Matching or correlation plays a major role in the recognition of characters that have known shapes. Simple template like features are appropriate for classifying patterns of known shape, such as characters [14]. This concepts lead to the design of inexpensive recognition schemes[14].

The template-matching approach can be regarded as a special case of decision theoretic approach when patterns are represented by their raw data or corresponding feature vectors.

Template-matching recognition technique works as follows : Each character in the character set is assigned a template or mask, which is a matrix of 0's and 1's representing black and white points. To classify a given character sample, it is digitized onto a matrix and compared to all templates; the comparison process consists of comparing the matrix elements of the sample to that of the templates. Classification is achieved if one of the templates provides a sufficiently good match to the character sample. The principal design efforts in this technique is in designing the masks or templates; the object is to maximize the probability of correct classification of each character and to minimize the probabilities of incorrect and/or no classification.

Mathematically, if $X = \{ x_1, \ldots, x_i, \ldots, x_N \}$ be a rectangular array obtained by scanning a character that is to be recognized and if $T = \{ t_1, \ldots, t_i, \ldots, t_N \}$ be the rectangular array for a template, then the pattern X is assigned to the class T when

$$\sum_{i=1}^{N} (x_i \oplus t_i)$$

is minimal, where $x_i \oplus t_i$ is the exclusive OR of x_i and t_i.

Template-matching approach may be interpreted as a special case where "features" are stored in the template and a special classification criterion (matching) is used for the classifier[1].

CHAPTER 3

DEVELOPMENT OF THE CHARACTER RECOGNITION SYSTEM

3.1 INTRODUCTION

Fu [1] and Tou and Gonzalez [13] observed in the mentioned papers that characterization of a pattern class by a roster of its member suggests automatic pattern recognition by **Template matching.** Rosenfeld [14] in his paper observed that matching or correlation plays a major role in the recognition of characters that have known shapes. In the paper he concludes that simple template like features are appropriate for classifying patterns of known shape, such as characters. He also points out that the template matching concept leads to the design of inexpensive recognition system.

So, to develop the first Bangla character recognition system, the template matching method/technique was adopted.

3.2 METHODOLOGY OF THE RECOGNITION PROCESS

To design a character recognition system the following methodological procedures are required :

-- **digitization :** To recognition a given text first it is to be digitized onto a matrix i.e., to be transformed into binary form for ease of handling by computer.

-- **separation** : To separate characters from the text as all the individual characters must be isolated to recognize them properly.

-- **normalization** : To register the character onto a common reference. Here it is registered tangent to its left and top margin.

-- **taking decision** : Depending on its previous 'learning' the machine takes the decision to the class belongings of the character. To classify, the classifier fetches a template from the template dictionary and calculates the amount of mismatch between the fetched template and the character. If the amount of mismatch is within the tolerable limit then the classifier gives the decision that the character belongs to the template class, otherwise the classifier fetches the next template and repeats the process until either a decision about the class belongings of the character is reached or the end of the template dictionary is reached in which case the decision is taken that the character can not be recognized and it belongs to rejection class.

A schematic diagram of the proposed methodology is shown in Figure 3.1.

In template matching technique the main design effort is spend on developing the decision taking algorithm. In the subsequent sections the design procedure of each step mentioned above is discussed.

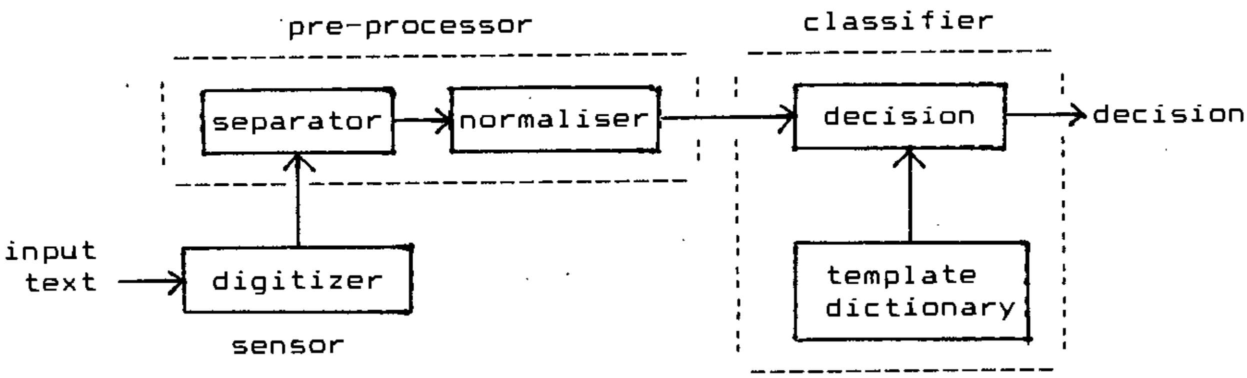

Figure 3.1 Diagram showing the methodology and the different components of the developed pattern recognition system.

3.3 THE ELEMENTS OF THE RECOGNITION SYSTEM DEVELOPED

The dotted enclosures in Figure 3.1 above shows the different components of the recognition system developed. The arrangements of the different components into different groups was based on the suggestion of Gonzalez and Thomason [12]. A brief discussion on the different components of the above diagram follows.

3.3.1 SCANNER

The digitizer is a device which converts a physical sample to be recognized into *pattern vector*.

It is often convenient, for recognition purpose, to arrange the pattern/character in the form of a *pattern vector*:

$$X = (x_1, x_2, x_3, \ldots, x_n)$$

where n is the number of measurements and component x_i of the vector X assumes the value 1 or 0 , depending on the state of the *i*th position for a particular input[9].

The digitizer used was a hand held scanner capable of scanning images of 10.5 cm (4.13 inches) wide. The resolution of the scanned image is 8 dots per mm (200 dots per inch) in both directions which is equivalent to 840 dots per line. Though the scanner offers four encoding mode: three half-tone encoding and one black and white encoding, the black and white option was used to scan the characters. This mode converts the image into binary image ----- 0 for the presence of a black spot, 1 for the absence of any black spot. Figure 3.2 shows a binary image of a character. The brightness of the scanned image can be controlled by the user. The scanner is software controlled. The output of the scanner i.e., the binary image obtained by scanning the characters were stored for further use.

3.3.2 PREPROCESSOR

3.3.2.1 CHARACTER SEPARATOR

Normally the classifier is designed to take decision on a sample presented to it. The sample characters are fed to the classifier one character at a time. But, in real life, stand alone characters carry very little message. Real life documents consists various combinations of characters. So,

```
111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111
111111111000001111111111111111111111111111111111
111110000000000000000000000000111111111111111111
111100000000000000000000000000000000000001111111
111100000000000000000000000000000000000001111111
111100000000000000000000000000000000000000111111
111100000000000000000000000000000000000000111111
111110000000000000000000000000000000000001111111
111111000111111111111111111111110000000111111111
111111111111111111111111111111111100001111111111
111111111111111111111111111111110000001111111111
111111111111111111111111111111000000001111111111
111111111111111111111111111100000000001111111111
111111111111111111111111110000000000011111111111
111111111111111111111111110000000000011111111111
111111111111111111111110000000011100011111111111
111111111111111111111100000000011100011111111111
111111111111111111111100000111111110011111111111
111111111111111111110000001111111110011111111111
111111111111111110000000011111111110011111111111
111111111111111100000000111111111110011111111111
111111111111111000000011111111111100011111111111
111111111111110000000011111111111100001111111111
111111111111100000000001111111111100001111111111
111111111111100000000000111111111100001111111111
111111111111110000000000111111111100001111111111
111111111111110000000000011111111100001111111111
111111111111111000000000001111111100001111111111
111111111111111100000000000111111100001111111111
111111111111111111100000000011111000011111111111
111111111111111111110000000011111000011111111111
111111111111111111111100000001111000011111111111
111111111111111111111111000000111000011111111111
111111111111111111111111000000000000011111111111
111111111111111111111111100000000000011111111111
111111111111111111111111111000000000001111111111
111111111111111111111111111000000000001111111111
111111111111111111111111111100000000011111111111
111111111111111111111111111100000000011111111111
111111111111111111111111111100000000011111111111
111111111111111111111111111100000000011111111111
111111111111111111111111111111000000001111111111
111111111111111111111111111111000000001111111111
111111111111111111111111111111100000001111111111
111111111111111111111111111111110000011111111111
111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111
```

Figure 3.2 Binary image of a sample character as obtained from the output of the scanner.

an algorithm was developed to separate the characters from words. The same algorithm also differentiates between the consequitive lines and also the words in a line. This separated characters were then fed to the normaliser for normalization. In English (European and American) language characters are well separated in a word. But in Bangla this is not the case. Sometimes they are separated from one another but in most cases they are connected in the top by a horizontal line known as "Matra" (মাত্রা). It was observed that this horizontal line is always less than one-fifth of the over-all height of the characters. [Part of the data is depicted in Table 3-1]. This information was applied in separating the characters joined by the top horizontal line. After being separated from a word the character was normalized before feeding to the classifier for decision. A flow-chart of the algorithm of character separation is shown in Figure 3.3 . [For source coding refer to **separate function** in Appendix A].

TABLE 3-1

Part of the data set used to determine

the height of the **MATRA**

name of character	total height in line	height of MATRA in line	height of MATRA in %
অ sample #1	44	6	13.64
অ sample #2	39	6	15.38
ক sample #1	42	7	16.67
ক sample #2	42	6	14.29
ঘ sample #1	38	5	13.16
ঘ sample #2	41	5	12.20
ঝ sample #1	42	5	11.90
ঝ sample #2	40	6	15.00
ব sample #1	44	7	15.91
ব sample #2	43	6	13.96
য sample #1	44	7	15.91
য sample #2	43	7	16.28
র sample #1	48	7	14.58
র sample #2	46	8	17.39
ষ sample #1	44	6	13.64
ষ sample #2	42	5	11.90
য় sample #1	58	6	10.34
য় sample #2	58	7	12.07

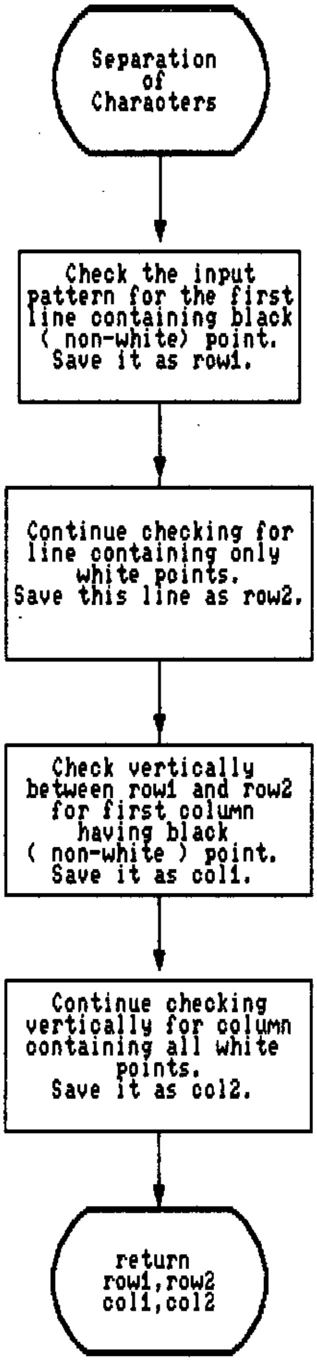

Figure 3.3 Flow-chart for the algorithm of separation of characters.

3.3.2.2 NORMALISER

It is often necessary to "preprocess" the given pictures in order to improve the reliability of subsequent matching. An important type of preprocessing is *NORMALIZING*, in which the picture is brought into a standard form, in order to make feature values independent of, or at least insensitive to, various transformations which may have affected the original picture. For the present work the digitized binary version of a printed character was normalized tangent to the top and left-hand margins of a rectangle field as "preprocessing is the necessary processing of the input image before the subsequent steps can be applied" [4].

A flow-chart of the algorithm for normalizing the input character tangent to the top and to the left-hand margins is shown in Figure 3.4. [For source coding refer to **normalize function** in Appendix A].

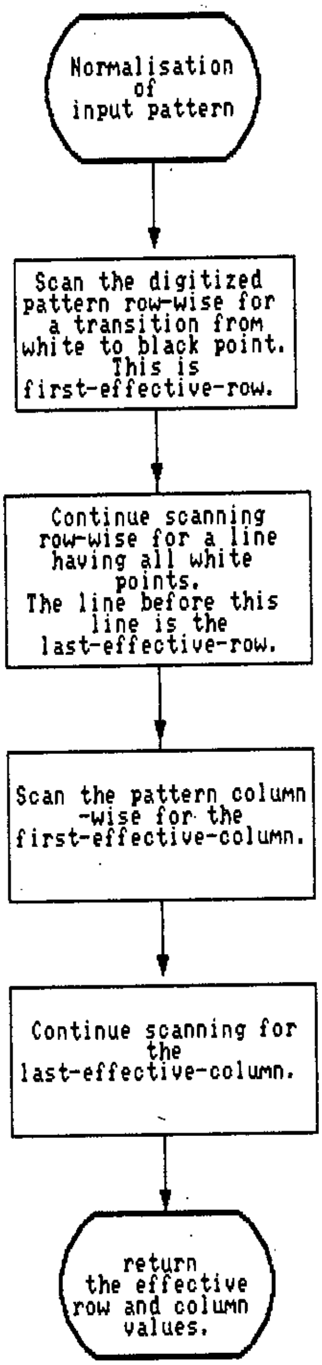

Figure 3.4 Flow-chart of normalising the input pattern

3.3.3 THE CLASSIFIER

The classifier classifies the input pattern to a class to which it belongs to i.e. it takes decision as regard to the belonging of the input pattern to a certain class. The character represented by the pattern matrix after being separated and normalized is now fed to the classifier. A dictionary of templates is searched to find the best matching template corresponding to the input character . The technique of developing the template has been described later in Section 3.3.

The dictionary of templates is arranged according to the frequency of occurrence of the characters because this will lessen the comparison time by finding the most frequent alphabets with less trials. The frequency table is given in Appendix B [11].

3.3.3.1 DECISION TAKING PROCESS

The decision procedure consists only of determining whether any of the templates affords a sufficiently good fit for an unknown sample. Determination technique works as follows:

Unknown sample matrix is compared with pre-stored template matrix. If there are p elements in the unknown sample matrix, m is the total mismatching found, t is the tolerance threshold in percent , then the unknown sample US belongs to $class\ M$ when

$$m \quad <= \quad p * t$$

The flow-chart for decision taking process is depicted in Figure 3.5.

3.4 DEVELOPMENT OF TEMPLATE

The principle design efforts in template-matching technique, as mentioned earlier, is in designing the templates. In designing a template for the individual characters, two objectives are paramount:

1) The template must fit the design character.
2) A certain minimum mismatch level must be maintained against all other character.

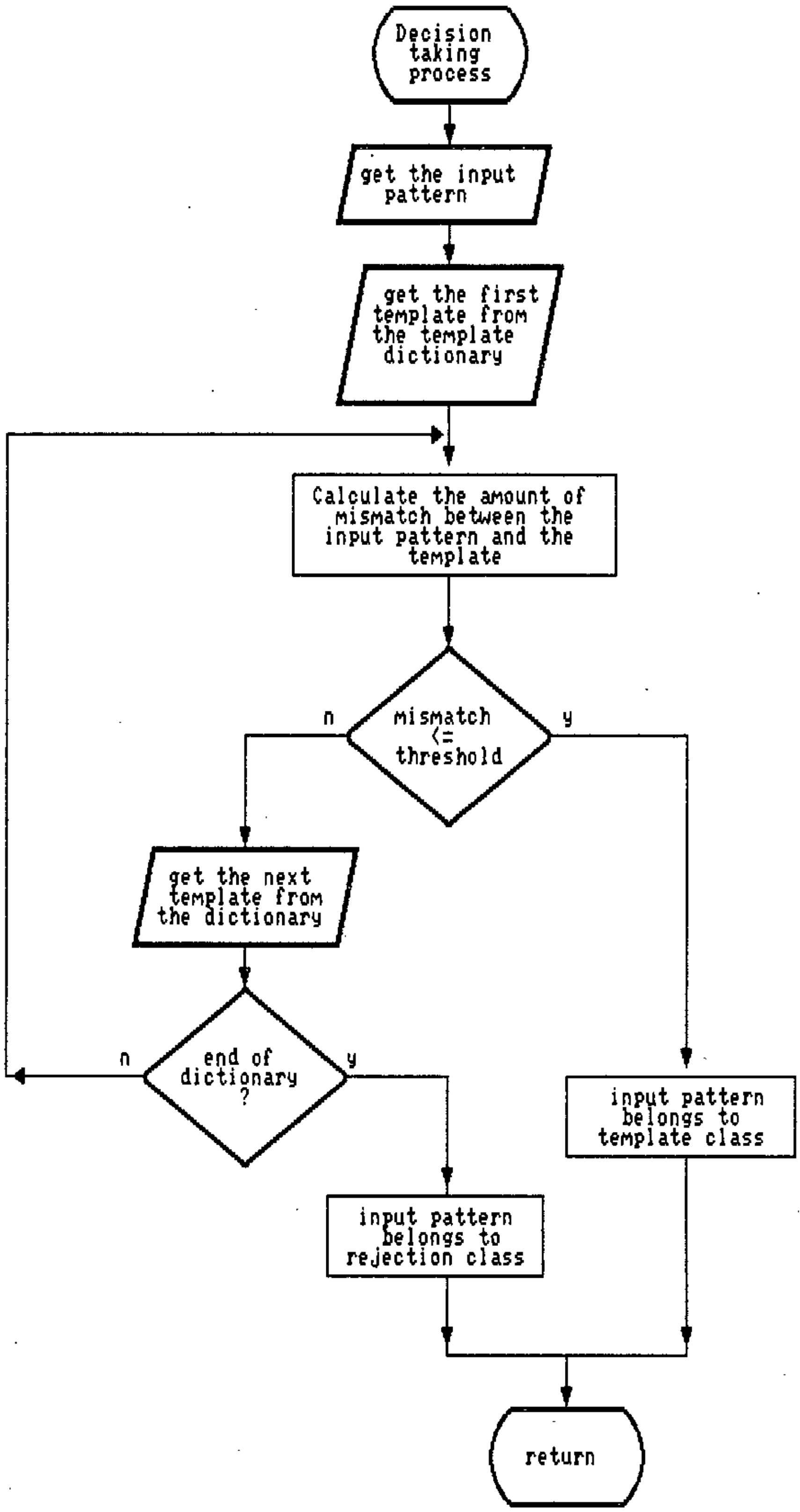

Figure 3.5 Flow-chart of decision taking process.

Templates are constructed from compressed version of sample characters called "skeletons".

3.4.1 SKELETON CHARACTERS

When the binary version of a printed character is registered tangent to top and left-hand margins of a rectangular field, some points are found to be black for most characters in the same class, while other points are usually white. There is also a region of uncertainty, distributed mainly along the border of the characters. Since the template designing procedure requires knowledge of the stable points, i.e., those which are reliably black or reliably white for a given character, so called "skeleton" characters are derived through a computer program (Appendix C) as follows:

A fixed numbers of samples of each character are read from the secondary storage, where the scanned binary version of the printed samples are being stored, and registered in the field. The number of black points in each bit position is counted. If this count is equal to or more than a predetermined threshold, the point is designated as a "stable black point", and if it is equal to or less then another predetermined threshold, the point is designated as a "stable white point". A point which fails to satisfy either criterion is labeled "unstable". The flow-chart is shown in Figure 3.6.

3.4.2 DESIGNING OF TEMPLATES

Samples and skeletons of characters belonging to a single class constitute the input to the template designing program. A template for a given character is composed of stable points selected from the skeleton of that character.

3.5 SELECTION OF NUMBER OF OPTIMUM TRAINING SET IN DESIGNING TEMPLATE

The effect of variation of number of training set in making template is depicted in Figure 3.7 . It is seen from the Figure 3.7 that the number of "reliable" points decreases as the number of training set increases. This is

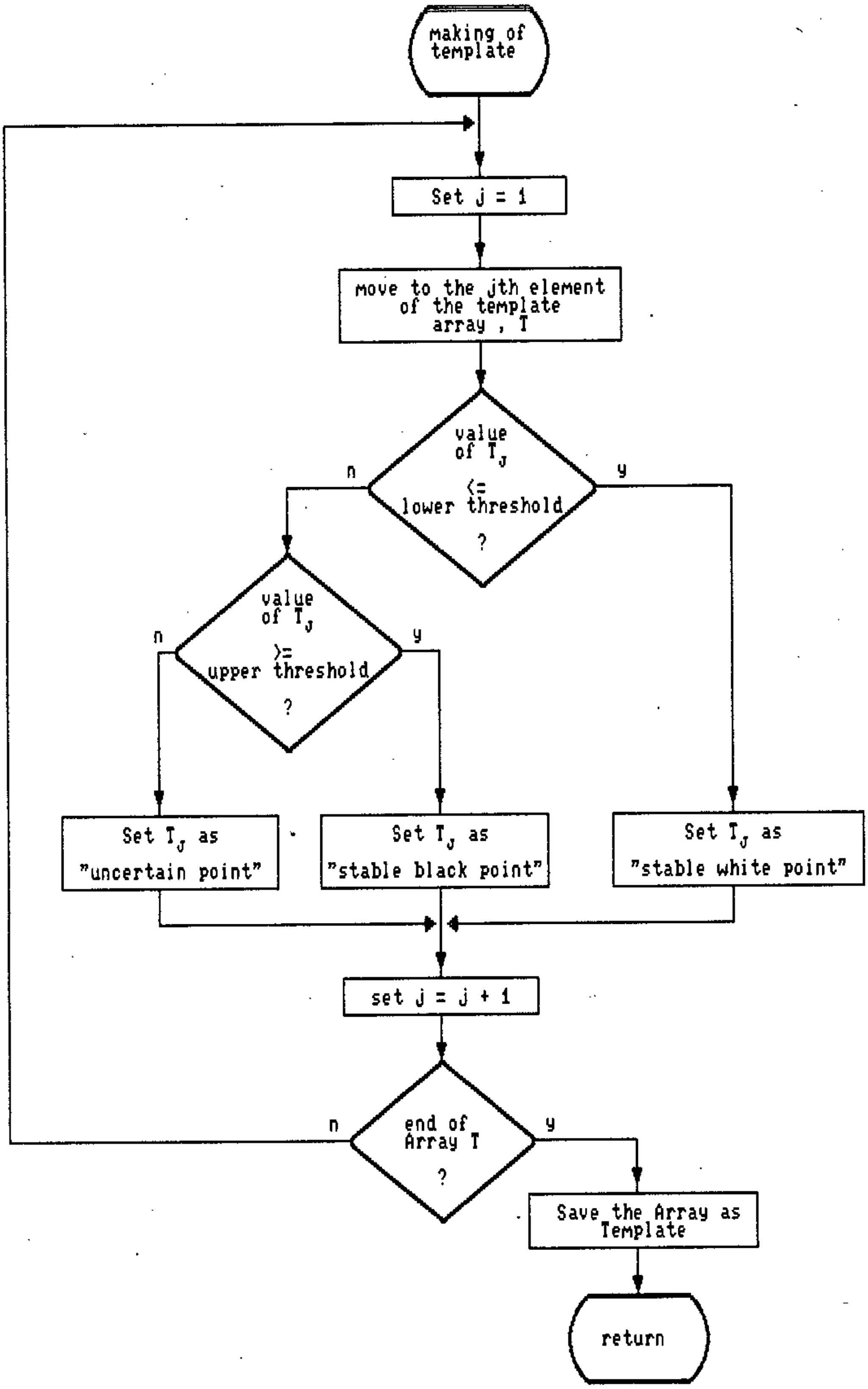

Figure 3.6 Flow-chart of making template.

because with the addition of more training sets the ambiguous points are being dropped out ---- the template contains more and more "reliable" points. From the figure it is also clear that while there is significant difference in templates with less number of training sets, a further increase in training set does not appreciably increase the performance. The fact can be seen in Figure 3.8. From Figure 3.7 it is observed that the curves become almost horizontal after point 13 or 14. Applying t-test of statistics it was found that the curve from point 15 can be taken as horizontal. So, the optimum value for the number of training set for making template was chosen as 15.

3.6 SELECTION OF MISMATCHING THRESHOLD

In selecting the mismatch threshold special emphasis was given to avoid the chances of misclassification between the characters. To find the optimum mismatching threshold, the recognition process was run on almost 900 characters [each alphabet has nearly 20 different samples.] for different percentage of mismatch tolerance. A summary of the result obtained is shown in Table 3-2 and in Figure 3-9.

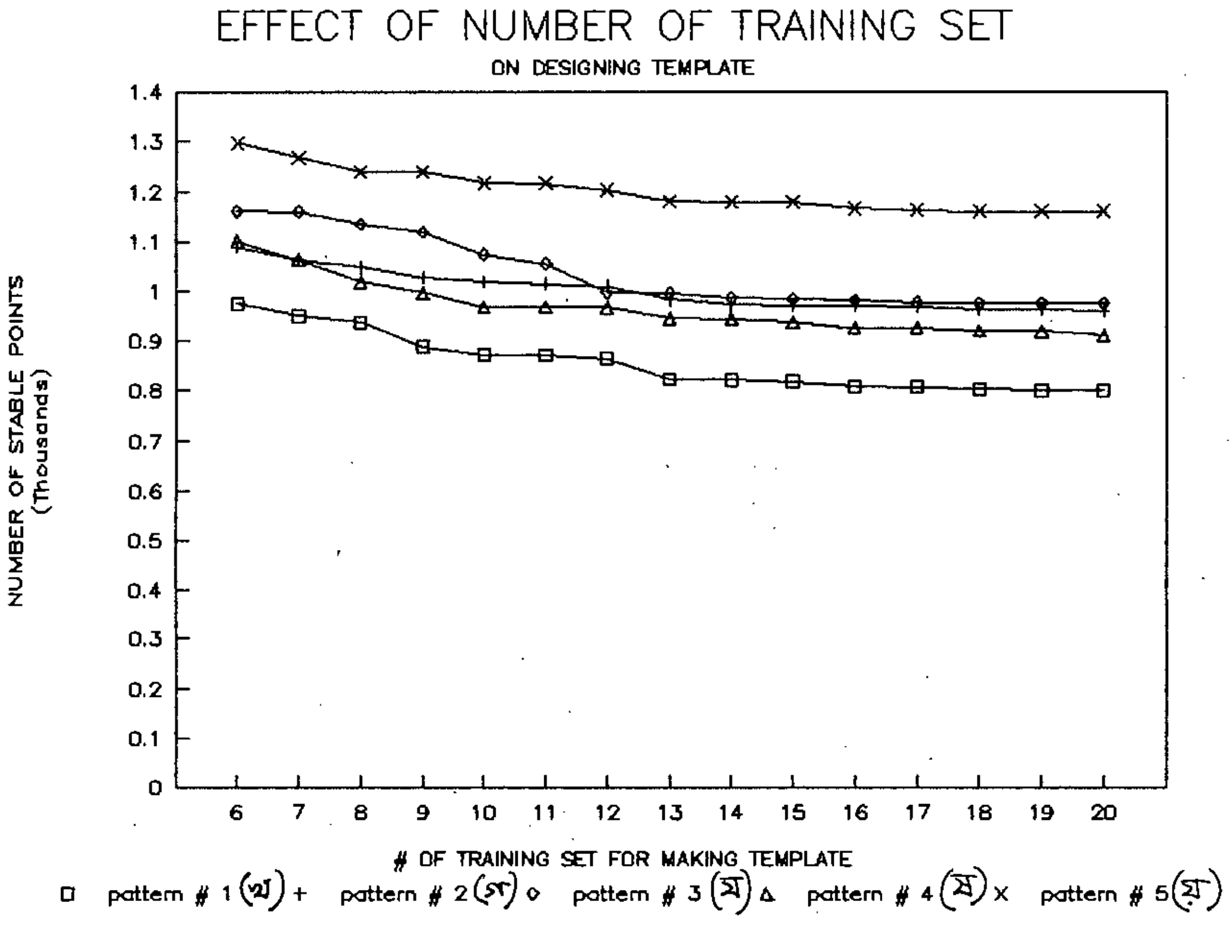

Figure 3.7 Effect of number of training set on designing template

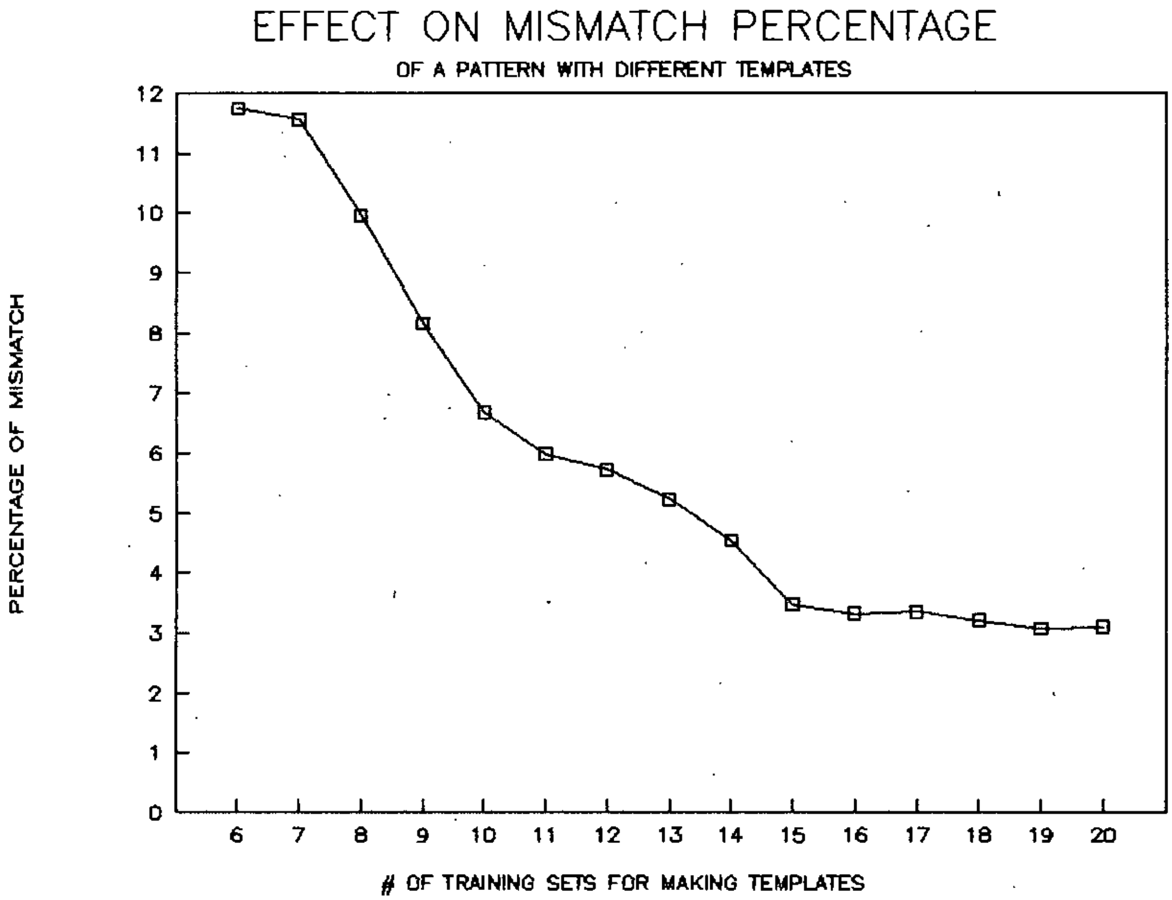

Figure 3.8 Effect on mismatch percentagee of a pattern
with different templates

From the Table 3-2 it is observed that with the increase of percentage of mismatch tolerance initially the number of properly classified characters increases, but after a certain percentage the number of properly classified characters began to decrease. On the other hand, the number of wrongly classified characters increases gradually. The reason for decreasing the number of properly classified characters with the increase in mismatch tolerance is that, with the increase in tolerance more and more characters begin to match with other then its own template and are being misclassified which were earlier classified properly.

From Table 3-2 and also from the graph of Figure 3.9, it is found that the maximum number of proper classification of characters is obtained for 11% mismatch tolerance. Hence, 11% was chosen as the mismatch threshold.

TABLE 3 - 2

A SUMMARY OF THE TEST OF PERFORMANCE OF THE

RECOGNITION SYSTEM ON 896 CHARACTERS

allowable mismatch tolerance	characters classified properly		characters classified wrongly		characters rejected	
in %	in #	in %	in #	in %	in #	in %
6	684	0.763	5	0.006	207	0.249
7	721	0.787	8	0.009	167	0.204
8	766	0.837	9	0.010	121	0.153
9	783	0.856	23	0.026	90	0.118
10	792	0.866	31	0.035	73	0.099
11	798	0.873	38	0.042	59	0.085
12	777	0.849	70	0.078	49	0.072
13	752	0.822	91	0.101	53	0.075
14	737	0.805	108	0.120	51	0.075
15	700	0.766	133	0.146	63	0.088

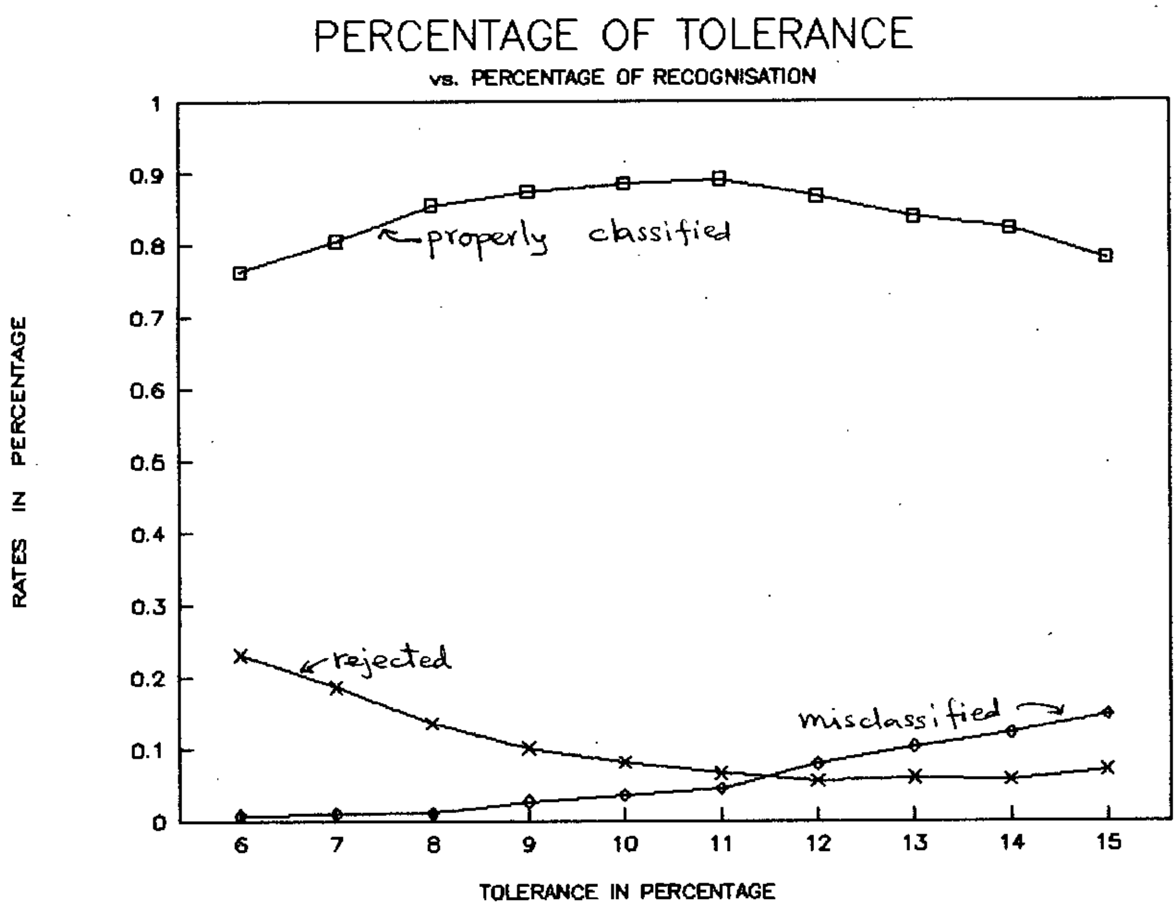

Figure 3.9 Effect on classification for different percentage of mismatch tolerences.

CHAPTER 4

DISCUSSIONS

4.1 GENERAL REMARKS

Recognition of characters is a very involved task as characters of allowable writing styles as well as variations in quality of the digitizer output have to be taken into consideration. The program developed has been successfully tested with a number of characters from different sources. The tests were designed to consider two aspects :

(i) To see whether the program develops consistent codes for characters of the same font i.e., whether the two instances of the same character from the same source yield the same code .

(ii) To see whether the program work for characters from different sources .

Sources in the above statements relates to different styles of writing, the result of the test was positive. To see whether the program developed gives consistent code for characters of same font i.e., whether the program can recognize the characters properly a test run on all characters each with 20 different samples was performed. The result of the test is shown in Table 4-1. Out of total 896 samples the program recognizes 798 times correctly, 39 times incorrectly and in 59 cases it fails to give a decision i.e., it rejects the input pattern. So, from Table 4-1 it is seen that Test (i) gives 89.06% accuracy and Test (ii) gives consistent accuracy within allowable writing styles.

The program developed for the present analysis operates on a representation of one character at a time. The representation is in the form of a matrix whose entries have level '1' or '0' corresponding to white or black pixels in the original pattern. Figure 3.2 shows one such binary form of a character.

4.2 GENERATION OF DATA

The present work concentrates on the analyzing of digitized characters. For this work, the digitized character in binary form was obtained from the output of the scanner used which has already been described in Section 3.3.1. This binary data was transferred to a matrix through software. Characters of different writing styles were considered and at the same time the errors which are quite natural to the practical digitizer output were taken into consideration. Though no "smoothing" operation has been used on the input

TABLE 4-1

SUMMARY OF THE RESULT OBTAINED BY RUNNING THE RECOGNITION PROGRAM ON A TOTAL OF 896 CHARACTERS.

pattern name	# of samples	properly classified	wrongly classified	rejected
অ	20	20	–	–
আ	20	19	1	–
ই	20	20	–	–
ঈ	20	19	–	1
উ	20	19	–	1
ঋ	20	15	–	5
এ	20	18	–	2
ঐ	20	18	–	2
ও	20	20	–	–
ঔ	20	18	–	2
ক	20	19	–	1
খ	20	18	–	2
গ	20	20	–	–
ঘ	20	19	–	1
ঙ	20	18	–	2
চ	20	19	–	1
ছ	20	20	–	–
জ	20	14	–	6
ঝ	20	18	–	2
ঞ	20	20	–	–
ট	20	19	–	1
ঠ	20	19	–	1
ড	20	14	6	–
ঢ	20	18	–	2
ণ	20	20	–	–
ত	20	19	–	1
থ	20	12	7	1
দ	20	20	–	–
ধ	20	18	–	2
ন	20	20	–	–
প	20	18	1	1
ফ	18	18	–	–
ব	20	20	–	–
ভ	20	17	–	3
ম	20	7	13	–
য	20	17	2	1
র	20	18	–	2

continuation of TABLE 4-1

pattern name	# of samples	properly classified	wrongly classified	rejected
ল	20	18	–	2
শ	18	15	–	3
ষ	20	10	9	1
স	20	19	–	1
হ	20	20	–	–
ড়	20	18	–	2
ঢ়	20	15	–	5
য়	20	18	–	2
	896	798	39	59

data, this could be used for characters having distortion like gaps and holes. This would lower the rejection rate.

4.3 ISOLATION OF CHARACTERS FROM TEXT

In Bangla language, even in printed text, the characters are not isolated from one another i.e. from the previous or the next character ---- most of the time they form a continuous figure. First requirement of the recognition problem is that the characters must be alone by itself. Hence, the individual characters must be isolated from the text. As the top horizontal line, known as *MATRA*(মাত্রা), forms a continuous line, so the isolation work was not an easy task. However, it was observed that if the presence of the *matra* can be ignored then the characters seem to be isolated from one another. Hence, the recognition of the presence of *matra* plays a significant role in isolating the characters. Though the height of the *matra* varies depending on the quality of printed text and on some other factors related to scanner [discussed in detailed in Appendix D], on close observation it was found that the height of the *matra* is well within 20% of the total height of the character. [Part of the data used is shown in Table 3.1]. Through the software this 20% top portion of the image was processed and the presence of any *matra* was ignored to appear the characters isolated from each other. However, this algorithm fails to isolate the conjunctive characters as well as the 'ি'-kar and 'ু'-kar overlapped with the characters.

4.4 DESIGNING OF TEMPLATES

A wide variety of templates from different number of sample sets with different threshold levels for "stable points" were designed and tried. It was observed that the performance of the template designed with 15 sample sets and 80% threshold level was satisfactory. 80% threshold level means that to be stable point a point must be white or black in more than or equal to 80% of the total cases. With templates produced using more sample sets, the performance increases but the design time, storage requirement outweights the performance. Decreasing the threshold level increases the risk of misclassification among the characters.

4.5 SELECTION OF MISMATCH THRESHOLD

As it has already been mentioned that the object of a recognition system is to maximize the probability of correct classification of each character and to minimize the probability of incorrect or misclassification, so, to avoid misclassification, observing the resemblance between the characters and applying the intuition, all the similar appearing characters were grouped together. List of such grouping is given in Table 4-2. To verify the validity of such groupings, a number of samples of each character of the group were taken, all the samples were then checked against the templates of the individual member of the group. A table was formed with the mismatch number. Two samples of such table is shown in Table 4-3 and 4-4. Table 4-3 was obtained by comparing the template र (RA) with the other members of group # 7 of Table 4-2 whereas Table 4-4 was obtained by comparing the template of घ (GHA) with the other members of group #10. For each member 20 samples were compared. First column of the table indicates the sample number. Column 2 (total point) shows the total number of points of the template that were compared with the corresponding point

TABLE 4-2

GROUPING OF BANGLA CHARACTERS

Group # 1	:	অ	আ	ত	ভ		
Group # 2	:	ই	হ				
Group # 3	:	ঈ					
Group # 4	:	উ	ঊ	ড	ড়		
Group # 5	:	এ	ঐ	ঞ			
Group # 6	:	ও	ঔ				
Group # 7	:	ঋ	ক	ঝ	ধ	ব	র
Group # 8	:	খ	থ				
Group # 9	:	গ	প				
Group #10	:	ঘ	য	ষ	য়		
Group #11	:	ঙ	ড	ড়			
Group #12	:	চ	ছ	ট			
Group #13	:	জ					
Group #14	:	ঠ	ট				
Group #15	:	ঌ					
Group #16	:	ড	ড়				
Group #17	:	ঢ	ঢ়	ট			
Group #18	:	ণ					
Group #19	:	দ					
Group #20	:	ন					
Group #21	:	ফ	য				
Group #22	:	ম					
Group #23	:	ল					
Group #24	:	শ					
Group #25	:	স					

of the sample. The next two columns show the number of mismatch found during comparison. Column 3 shows the mismatching in numbers whereas column 4 shows the same information in percentage form calculated on the basis of total number of points.

From Table 4-3 it is observed that র (RA) has a dis-similarity of 52.91 to 59.73% with ঋ (vowel #7), 28.82% to 37.00% with ক (KA), 27.05% to 32.23% with ঝ (JHA), 39.92% to 56.03% with ধ (DHA) and 14.00% to 36.33% with ব (BA) whereas the amount of dis-similarities between the samples of র (RA) itself ranges from 0.22% to 8.27% only.. So, it can be easily concluded that though ব (BA) has some resemblance with র (RA) but ঋ, ক, ঝ ও ধ have very little resemblance with র (RA).

From Table 4-4 it is observed that the dis-similarity of samples of ঘ (GHA) between themselves ranges from 0.00% to a maximum of 7.25% [except sample # 18, which has a high value due to high distortion in scanning] whereas that with জ (JA) ranges from a minimum of 10.77% to a maximum of 32.89%, with য from 15.99% to 36.58% and with য় ranges from 35.5% to 57.72%.

The same type of results were observed regarding the other groups of Table 4-2. As the amount of dis-similarity is strikingly high in most of the cases, so the hypothesis of grouping the characters according to Table 4-2 was ultimately discarded.

TABLE 4-3

TABLE USED TO VERIFY THE VALIDITY OF GROUPING OF CHARACTERS.
TABLE OBTAINED BY COMPARING THE TEMPLATE OF র (RA)
WITH THE OTHER MEMBER OF ITS GROUP.

sample #	total point	mismatch in #	mismatch in %	sample #	total point	mismatch in #	mismatch in %
PATTERN USED : ঋ (VOWEL # 7)							
1	1177	664	56.41	2	1248	715	57.29
3	2003	1156	57.71	4	1278	741	57.98
5	1177	703	59.73	6	1248	708	56.73
7	1218	708	58.13	8	1297	694	53.51
9	1381	749	54.24	10	1151	684	59.43
11	1206	719	59.62	12	1391	761	54.71
13	1177	692	58.79	14	1391	736	52.91
15	1218	699	57.39	16	1319	723	54.81
17	1361	742	54.52	18	1407	752	53.45
19	1218	689	56.57	20	1257	719	57.20
PATTERN USED : ক (CONSONANT # 1)							
1	1254	399	31.82	2	1254	464	37.00
3	1311	448	34.17	4	1213	443	36.52
5	1311	413	31.50	6	1311	418	31.88
7	1356	423	31.19	8	1341	435	32.44
9	1433	413	28.82	10	1283	430	33.52
11	2353	1482	62.98	12	1283	423	32.97
13	1283	371	28.92	14	1387	424	30.57
15	1327	433	32.63	16	1254	403	32.14
17	1356	425	31.34	18	1387	413	29.78
19	1327	400	30.14	20	1254	397	31.66
PATTERN USED : ঝ (CONSONANT # 9)							
1	1176	360	30.61	2	1160	356	30.69
3	1283	347	27.05	4	1186	361	30.44
5	1241	362	29.17	6	1150	337	29.30
7	1297	391	30.15	8	1098	328	29.87
9	1241	342	27.56	10	1268	395	31.15
11	1241	383	30.86	12	1203	355	29.51
13	1241	383	30.86	14	1150	358	31.13
15	1213	391	32.23	16	1098	325	29.60
17	1254	377	30.06	18	1125	338	30.04
19	1241	367	29.57	20	1142	363	31.79

continuation of Table 4-3

PATTERN USED : ध (CONSONANT # 19)

1	892	453	50.78	2	852	436	51.17
3	987	553	56.03	4	912	456	50.00
5	1020	559	54.80	6	957	475	49.63
7	987	459	46.50	8	987	510	51.67
9	1010	516	51.09	10	932	506	54.29
11	957	521	54.44	12	894	398	44.52
13	957	506	52.87	14	1017	444	43.66
15	957	508	53.08	16	975	500	51.28
17	998	526	52.71	18	1020	522	51.18
19	953	493	51.73	20	754	301	39.92

PATTERN USED : ब (CONSONANT # 23)

1	1229	172	14.00	2	1241	331	26.67
3	1140	279	24.47	4	1257	187	14.88
5	1203	223	18.54	6	1248	368	29.49
7	1166	228	19.55	8	1341	294	21.92
9	1356	364	26.84	10	1268	227	17.90
11	1268	316	24.92	12	1203	437	36.33
13	1356	265	19.54	14	1241	247	19.90
15	1311	266	20.29	16	1283	404	31.49
17	1268	348	27.44	18	1229	266	21.64
19	1176	333	28.32	20	1327	273	20.57

PATTERN USED : ब़ (CONSONANT # 27)

1	1348	7	0.52	2	1329	108	8.13
3	1306	4	0.31	4	1487	31	2.08
5	1473	21	1.43	6	1564	102	6.52
7	1348	7	0.52	8	1306	14	1.07
9	1306	108	8.27	10	1438	13	0.90
11	1375	77	5.60	12	1473	87	5.91
13	1151	90	7.82	14	1391	12	0.86
15	1381	15	1.09	16	1361	24	1.76
17	1348	71	5.27	18	1407	52	3.70
19	1381	3	0.22	20	1615	112	6.93

TABLE 4-4

TABLE USED TO VERIFY THE VALIDITY OF GROUPING OF CHARACTERS.
TABLE OBTAINED BY COMPARING THE TEMPLATE OF घ (GHA)
WITH THE OTHER MEMBER OF ITS GROUP.

sample #	total point	mismatch in #	mismatch in %	sample #	total point	mismatch in #	mismatch in %
PATTERN USED : घ (CONSONANT # 4)							
1	1116	73	6.54	2	1214	27	2.22
3	1115	50	4.48	4	1214	35	2.88
5	1355	25	1.85	6	1313	11	0.84
7	1433	70	4.88	8	1245	0	0.00
9	1321	30	2.27	10	1433	65	4.54
11	1315	12	0.91	12	1236	22	1.78
13	1355	7	0.52	14	1277	24	1.88
15	1355	11	0.81	16	1393	86	6.17
17	1393	101	7.25	18	1358	474	34.90
19	1313	12	0.91	20	1315	61	4.64
PATTERN USED : य (CONSONANT # 26)							
1	1393	334	23.98	2	1397	422	30.21
3	1438	473	32.89	4	1393	306	21.97
5	1479	471	31.85	6	1393	187	13.42
7	1282	188	14.66	8	1393	306	21.97
9	1245	153	12.29	10	1245	156	12.53
11	1355	185	13.65	12	1355	154	11.37
13	1315	204	15.51	14	1277	142	11.12
15	1288	175	13.59	16	1479	464	31.37
17	1315	202	15.36	18	1355	146	10.77
19	1399	413	29.52	20	1315	225	17.11
PATTERN USED : श (CONSONANT # 30)							
1	1352	302	22.34	2	1245	320	25.70
3	1391	343	24.66	4	1282	205	15.99
5	1308	346	26.45	6	1503	497	33.07
7	1277	209	16.37	8	1277	282	22.08
9	1214	302	24.88	10	1277	248	19.42
11	1236	420	33.98	12	1094	394	36.01
13	1345	492	36.58	14	1352	245	18.12
15	1214	253	20.84	16	1391	278	19.99
17	1163	412	35.43	18	1351	382	28.28
19	1345	430	31.97	20	1313	305	23.23

continuation of Table 4-4

PATTERN USED : ज्ञ (CONSONANT # 36)

1	1845	917	49.70	2	1845	736	39.89
3	1993	810	40.64	4	1693	764	45.13
5	1993	819	41.09	6	1093	489	44.74
7	2053	1185	57.72	8	1364	627	45.97
9	1913	824	43.07	10	1752	735	41.95
11	1873	757	40.42	12	1781	689	38.69
13	1976	972	49.19	14	1674	719	42.95
15	1625	692	42.58	16	1937	856	44.19
17	1898	811	42.73	18	1845	732	39.67
19	1859	660	35.50	20	1625	660	40.62

4.6 DISCUSSION OF THE PROGRAM DEVELOPED

The program is menu driven and during its execution inter-acts with the user. It was designed in modular form. The main routine after displaying a welcome message for the user presents a menu and waits to know the user option. The user has three options to choose :

(1) recognition option : it presents yet another sub-menu.

(2) make/see mask menu : which also shows further sub-menu.

(3) the option to quit this program and return to the operating system.

Recognition menu is a menu with three options :

(1) scan and recognition option : which allows the user to scan a character and present the character to the recognition system for recognition.

(2) pre-scanned character recognition option : which allows the user to use a pre-scanned and stored data file to use for the purpose of recognition.

(3) quit option : which takes the user to the main menu.

Make/see mask menu has also three options :

(1) Make new mask : using this option one can make a new mask from the named pattern sets. This option has further sub-options :

    (1) make mask from pre-scanned character patterns.

    (2) scan and make mask.

    (3) return to mask menu.

(2) see pre-designed mask option : by choosing this option one can see the mask on the screen in graphics mode.

(3) this options takes the user to the main menu.

CHAPTER 5

CONCLUSION AND SUGGESTION FOR FUTURE WORK

5.1 CONCLUSIONS

For the present analysis the basic idea of template matching decision theoretic method ----- i.e. to compare the unknown character with pre-stored templates in order to classify it ---- is utilized. The idea was first proposed and successfully used by Casey and Nagy [17] in 1965 to recognize printed Chinese characters. One major difference between their work and the present work is that Chinese symbols are well separated from one another but this is not the case with Bangla characters. Bangla characters are joined with one another in most of the cases. So, a technique for separating characters from one another into their original form has been suggested.

Though the technique developed here is quite efficient in recognizing Bangla characters, it fails to recognize any conjunctive characters or characters having a 'ৃ'-kar or a 'ু'-kar under it or a 'ি'-kar preceding it.

The technique for separation of characters from a text will no doubt play a significant role in the field of Bangla character recognition; but still it needs further improvement so as to make it able to separate characters

having special vowel or consonant sign under it, such as '', '', '' etc., or the special vowel sign like '', ''
etc.

Like every method the present one naturally has got some limitations as mentioned; but still the performance of the present method in recognizing Bangla characters has found to be quite satisfactory.

5.2 SUGGESTION FOR FUTURE WORK

Further research work in automatic recognition of Bangla character should concentrate on

i) developing an algorithm for designing templates for composite characters.

ii) developing an algorithm for separating overlapping characters.

iii) instead of using the whole mask matrix , whether selected elements of the mask matrix could be used , should be investigated. This would lessen the compare time, increasing the throughput.

iv) modification of the present algorithm for implementing in composite character recognition.

v) Other methods of character recognition such as Syntactic methods should be investigated and the performance should be compared with the present work.

REFERENCES

REFERENCES

1. K. S. Fu : Introduction, Digital Pattern Recognition : ed. by K. S. Fu, pp. 1 - 14, Springer-Verlag, New York, 1976.
2. D. Dutta Majumder : Pattern Recognition Methods and Applications, Recent Developments in Pattern Recognition and Digital Techniques : pp. 150 - 190, Indian Statistical Institute , 1977.
3. K.S.Fu : Introduction to Syntactic Pattern Recognition, Syntactic Pattern recognition applications : ed. by K.S.Fu , pp. 1 - 30 , Springer - Verlag , New York , 1977.
4. Laveen Kanal : Patterns in Pattern Recognition 1968 - 1974 : IEEE Trans. Information Theory , vol. IT - 20, pp. 697-722 , Nov ' 1974.
5. Kendall Preston Jr. : A comparison of analog and digital techniques for Pattern Recognition : Proc. IEEE , vol. 60, pp. 1216 - 1231 , Oct. ' 1972.
6. S. H. Unger : Pattern detection and recognition : Proc. IRE. vol. 47, pp. 1737 - 1752, October' 1959.
7. L. P. Horwitz & G. L. Shelton : Pattern recognition using auto-correlation : Proc. IRE. January '1961, pp. 175-185.

8. P. G. Perotto : A new method for automatic character recognition : IEEE Trans. on Electronic Computers, pp. 521- 526, October'1963.

9. H.A. Glucksman : A parapropagation pattern classifier : IEEE Trans. on Electronic Computers, pp.434-443, June'1965.

10. S. S. Yau & C. C. Yang : Pattern recognition by using Associative memory : IEEE Trans. on Electronic Computers, pp. 944 - 947 ,December ' 1966.

11. A.W.Holt : Comparative religion in Character Recognition machines : IEEE Computer Group News , vol. 2 , pp. 3 - 11 , Nov ' 1968.

12. R.C. Gonzalez and M.G. Thomason : Syntactic pattern recognition , An Introduction : 1982 , Addision - Wesley Publishing Company , Reading , Massachusetts , U.S.A.

13. J.T. Tou and R.C. Gonzalez : Pattern recognition principles : 1981 ,Addision-Wesley Publishing Company, Reading, Massachusetts, U.S.A.

14. Rosenfeld, A. : Digital picture analysis : edited by Rosenfeld A., Springer - Verlag , Berlin , 1976 .

15. Md. Mozammel Huq Azad Khan : Optimal realization of Bengali key-board and character encoding for computer applications : M.Sc. Engg. Thesis, Oct.'1986, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology , Dhaka, Bangladesh.

16. K.S. Fu : Recent developments in pattern recognition : IEEE Trans. on Computers , vol.: C - 29 , October'1980 .

17. Casey and Nagy : Printed Chinese character recognition : IEEE Trans. on Electronic Computers, vol.: EC-15, no. 1, February'1966, pp. 91 - 101.

18. K. S. Fu : Introduction, Digital Pattern Recognition : ed. by K. S. Fu, pp. 95 - 134, Springer-Verlag, New York, 1976.

19. M. A. Sattar and S. M. Rahman : An experimental investigation on Bangla character recognition system : Bangladesh Computer Soceity Journal , vol. - 4, no. 1, Dec'89, pp. 1 - 4 .

APPENDIX

APPENDIX A

SOURCE CODING OF THE PROGRAM DEVELOPED

```
/*   THE COMPLETE PROGRAM OF THE RECOGNITION SYSTEM DEVELOPED */

#include <stdio.h>
#include <conio.h>
#include <dir.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <math.h>
#include <graphics.h>
#include <io.h>
#include <ctype.h>
#include <mem.h>
#include <alloc.h>

#define MASK1 0x80
#define MASK2 0x01
#define M     3

/* ========================================================== */
/*              DEFINITION OF DIFFERENT MESSAGES              */
/* ========================================================== */

char *msg[] =
{       "********  MAIN MENU  *******",
        "RECOGNITION",
        "DESIGN/SEE TEMPLATE",
        "QUIT",
        "******* RECOGNITION  MENU *******",
        "SCAN AND RECOGNITION",
        "PRE-SCANNED CHARACTER RECOGNITION",
        "*******  DESIGN/SEE  TEMPLATE  MENU  *******",
        "DESIGNING TEMPLATE ",
        "SEE PRE-DESIGNED TEMPLATE ",
        "********** MAKING  TEMPLATE  MENU  *********",
        "MAKE TEMPLATE FROM PRE-SCANNED CHARACTERS",
        "SCAN AND MAKE TEMPLATE ",
        "*******  DESIGNING  TEMPLATE  MENU  *******",
        "MAKING  A  NEW  TEMPLATE",
        "DELETING  AN  OLD  TEMPLATE",
}   ;
/* ==========   END OF MESSAGE DEFINITION  ================= */
```

```
char patternname[20] ;
unsigned long  size;
div_t divide ;

typedef struct  {        int width, height, row1, row2, col1 ;
                         int col2, pix1, pix2 ;
                         int condition_row, condition_col ;
                } SEPE_INFO ;
typedef struct  {        int row1, row2, col1, col2, pix1, pix2;
                } NORM_INFO ;

NORM_INFO *file[15] ;
SEPE_INFO *sepe_file ;


/* ============================================================ */
/*                CURSOR CONTROL ROUTINE.                        */
/*     HIDES, ELONGATES OR SETS THE CURSOR TO NORMAL SHAPE      */
/*     ACCORDING TO USER NEED.                                   */
/*     CALLED BY : WELCOME AND THANKS FUNCTION.                  */
/*     CALLS  ON : NONE .                                        */
/* ============================================================ */

void setcursor ( int x )
#define HIDE       0
#define NORMAL     1
#define ELONGATED 2
{        union REGS regs;
         switch ( x )
         {        case 0 : regs.h.ch = 32; break;
                  case 1 : regs.h.ch = 11; regs.h.cl = 12; break;
                  case 2 : regs.h.ch =  0; regs.h.cl = 13; break;
                  default: exit(0);
         }
         regs.h.ah = 1;
         int86 ( 0x10 , &regs , &regs );
}
/* ===============   END OF CURSOR CONTROL ROUTINE   ========= */


void sing (void ) {  printf ("\a\a");    }


void box ( void )
{
     clrscr();  window ( 25,7,70,18 );
}
```

```
/* ============================================================ */
/*                 SHOW   OPTION   FUNCTION                     */
/*   THIS FUNCTION SHOWS THE OPTIONS AVAILABLE TO THE USER      */
/*   CALLED BY : MAIN , RECOGNITION AND MAKEMASK FUNCTIONS      */
/*   CALLS ON  : NONE.                                          */
/* ============================================================ */

void show_option ( char *s1,char *s2,char *s3,char *s4)
{       clrscr();
        cprintf ( " %s\r\n\n",s1);
        cprintf ( " 1. %s\r\n",s2);
        cprintf ( " 2. %s\r\n",s3);
        cprintf ( " 3. %s\r\n\n",s4);
        cprintf ( " Press your choice.... ");
}

/* ==========   END OF SHOW OPTION FUNCTION   ============== */



/* ============================================================ */
/*                 GET   OPTION   FUNCTION                      */
/*            THIS ROUTINE GETS THE USER OPTION                 */
/*                                                              */
/*      CALLED BY : MAIN , RECOGNITION AND MAKEMASK FUNCTIONS   */
/*      CALLS  ON : NONE .                                      */
/* ============================================================ */

char get_option ( void )
{       char choice;
        while ( ((choice = getche()) <'1' || choice > '3') \
                                    && choice != '\x1b')
        {       printf ( "\a\a");
                textattr ( 0xf0 );
                cprintf ( "\n\n\n\     Wrong option . \
                                  Try again ....    ");
                textattr ( 0x07 );
                gotoxy ( 24,7 );
        }
        return ( choice ) ;
}

/* ===========   END OF GET OPTION ROUTINE   =============== */
```

```
/* ============================================================= */
/*            SHOWS A WELCOME MESSAGE TO THE                     */
/*            USER AFTER EVOKING THE PACKAGE.                    */
/*            CALLED BY : MAIN FUNCTION                          */
/*            CALLS  ON : SETCURSOR  FUNCTION                    */
/* ============================================================= */

void welcome ( void )
{       box();
        setcursor ( HIDE );
        highvideo ();
        textattr ( 0xF0 );
        cprintf ("                                      \r\n");
        cprintf ("                                      \r\n");
        cprintf ("              WELCOME TO              \r\n");
        cprintf ("                                      \r\n");
        cprintf ("           BANGLA CHARACTER           \r\n");
        cprintf ("                                      \r\n");
        cprintf ("          RECOGNITION SYSTEM          \r\n");
        cprintf ("                                      \r\n");
        cprintf ("                                      \r\n");
        textattr ( 0x07 );
        normvideo ();
        sing ();
        cprintf ( " \r\n      Press any key to proceed ... ");
        getch(); setcursor ( NORMAL ) ;
}

/* ========   END OF WELCOME MESSAGE SHOWING ROUTINE   ======= */


/* ============================================================= */
/*        THIS IS A THANKS GIVING ROUTINE. WHEN THE              */
/*        USER WISHES TO RETURN TO DOS A "THANK YOU"             */
/*        MESSAGE IS SHOWN ON THE SCREEN.                        */
/* ============================================================= */

void  thanks ( void )
{       setcursor ( HIDE );
        clrscr ();
        textattr ( 0xf0 );
        cprintf ("\n");
        cprintf ("                                   \r\n");
        cprintf ("                                   \r\n");
        cprintf ("             THANK  YOU            \r\n");
        cprintf ("                                   \r\n");
        cprintf ("                 FOR               \r\n");
        cprintf ("                                   \r\n");
```

```
        cprintf ("      USING  RECOGNITION  SYSTEM      \r\n");
        cprintf ("                                      \r\n");
        cprintf ("                                      \r\n");
        textattr ( 0x07 );
        cprintf ( "\n\n Press any key to return to DOS ...");
        getch ();
        window ( 1,1,80,25);
        clrscr ();
        setcursor ( NORMAL );
}

/* ==========   END OF THANKS GIVING ROUTINE   =============== */



/* ============================================================ */
/*            CHARACTER ISOLATION  ROUTINE                       */
/*       TO ISOLATE CHARACTERS FROM PRINTED TEXT                 */
/*       CALLED BY : RECOGNITION  FUNCTION.                      */
/*       CALLS  ON : NONE .                                      */
/* ============================================================ */

void isolate_character ( char huge *pattern , SEPE_INFO *s  )
{    int h, col, n ;
     unsigned long count ;
     unsigned char data, bit ;

/* ========================================================== */
/* row search for the first transition from white to black   */
/* ========================================================== */
   h = s->row1 ;
   count = 32 + s->width * ( h - 1 )   ;
   while ( h++ < s->height )
   {  col = 0;
      while ( col++ < s->width )
      {  data = pattern[count++];
         if (data != 0xff)     goto label1;
      }
   }    /* h gives the row where transition occurs */
   s->condition_row = -1 ;
   return ;
   label1:       s->row1 = h - 1;
/* ========================================================== */
/* row search for the first transition from black  to white  */
/* ========================================================== */
     col = 0;
     count = 32 + s->width * s->row1 ;
```

```
    while ( col++ < s->width )
    {     data = pattern[count++];
          if (data != 0xff)
          {     col = 0 ;
                count = 32 + s->width * h ;
                h++ ;
                continue ;
          }
    } /*        h  gives the row where transition from black
                to white occurs */
    label2:     s->row2 = h - 1 ;
/* ====================================================== */
/*      column search  from bottom  to top                */
/*      for the first transition from white to black      */
/* ====================================================== */
    col = s->col1 ; h = s->pix1 ;
    while (col++ < s->width )
    {     while ( h++ <= 7 )
          {     n = s->row2 ;
                label5:
                while ( n-- >= s->row1 )
                {     if ( wherey() > 10 )
                      {     cprintf("\n\rPress any key ...");
                            getch() ; clrscr() ;
                      }
                      count = 32 + n * s->width + col - 2 ;
                      data = pattern[count];
                      data <<= ( h - 1 ) ;
                      bit = data & MASK1 ;
                      if (bit == 0)
                            if ( (s->row2 - n) >= 35)
                                  break;
                            else goto label3 ;
                }
          }
          h = 0 ;
    }  /* col gives left margin  */
    s->condition_col = -1 ;
    return ;
    label3:s->col1 = col - 1 ;
    s->pix1 = h - 1 ;
/* ====================================================== */
/*      column search  from top to bottom                 */
/*      for the first transition from black to white      */
/* ====================================================== */
    col = s->col1 ;
    h = s->pix1 + 1 ;
```

```
    while (col++ < s->width )
    {     while ( h++ <= 7 )
          {     n = s->row2 ;
                while ( n-- >= s->row1 )
                {     count = 32 + n * s->width + col - 2 ;
                      data = pattern[count];
                      if ( wherey() > 10 )
                      {     cprintf("\n\rPress any key ... ");
                            getch() ; clrscr() ;
                      }
                      data <<= ( h - 1 ) ;
                      bit = data & MASK1 ;
                      if (bit == 0)
                          goto label4 ;
                }
                s->col2 = col - 1 ;
                s->pix2 = h - 2 ;
                if ( s->pix2 < 0 )
                {     s->col2 -=1 ;
                      s->pix2 = 7 ;
                }
                goto end ;
                label4:
                if ( (s->row2 - n) >= 35 )
                {     s->col2 = col - 1 ;
                      s->pix2 = h - 1 ;
                      if ( s->pix2 < 0 )
                      {     s->col2 -= 1 ;
                            s->pix2 = 7 ;
                      }
                      goto end;
                }
          }
          h = 0 ;
    }  /* col gives right margin  */
end : ;
}

/* ========== END OF CHARACTER ISOLATION  ROUTINE ========== */
```

```
/* ============================================================= */
/*                 NORMALISE  ROUTINE.                            */
/*        THIS ROUTINE TAKES A PATTERN AND FINDS THE ROWS         */
/*        AND  COLUMNS FROM WHERE THE CHARACTER  ACTUALLY         */
/*        STARTS.  IT  THEN STORES THE INFORMATION  IN  A         */
/*        STRUCTURE FOR FURTHER USE.                              */
/*        CALLED BY : RECOGNITION AND MAKEMASK FUNCTIONS          */
/*        CALLS  ON : NONE                                        */
/* ============================================================= */

void normalise ( char huge *pattern , NORM_INFO *s  )
{    int h,col,bit,n,c ;
     unsigned long count ;
     unsigned char data ;


/* ============================================================= */
/*   SEARCHING ROW-WISE FROM TOP TO BOTTOM FOR THE FIRST          */
/*   TRANSITION FROM WHITE POINT TO BLACK POINT.                  */
/* ============================================================= */

 h = sepe_file->row1 ;
 while ( h++ < sepe_file->row2 )
 { col = sepe_file->col1 ;
   while ( col++ < sepe_file->col2 )
   { count = 32 + ( h - 1 ) * sepe_file->width + col - 1 ;
     data = pattern[count++];
     if (data != 0xff)   /* IF THE CONDITION IS TRUE THEN A
                            TRANSITION FROM WHITE POINT TO
                            A BLACK POINT IS FOUND         */
     {     s->row1 = h - 1;    /*   SAVES THE INFORMATION
                                    IN  A STRUCTURE        */
/* ============================================================= */
/*   SEARCHING FROM BOTTOM TO UPWARD FOR A TRANSITION             */
/*   FROM WHITE POINT TO BLACK POINT                              */
/* ============================================================= */

     n = 1;
     h = sepe_file->row2 ;
     while ( h-- > s->row1)
     {     col = sepe_file->col1 ;
           count = size - (sepe_file->height - sepe_file->row2 \
                         - 1 ) *  sepe_file->width ;
           while ( col++ < sepe_file->col2 )
           {  data = pattern[count++];
              if (data != 0xff)  /* IF THE CONDITION IS TRUE
                                    THEN A TRANSITION FROM WHITE
                                  POINT TO BLACK POINT IS FOUND */
```

```
    {     s->row2 = h + 1;          /* SAVES THE INFORMATION IN     */
                                    /* A STRUCTURE                  */
/* ============================================================== */
/*         COLUMN SEARCH FROM TOP TO BOTTOM FOR THE FIRST          */
/*         TRANSITION  FROM  WHITE  POINT TO BLACK 'POINT          */
/* ============================================================== */

          col = sepe_file->col1 ;
          while (col++ < sepe_file->col2 )
          {     n = s->row1 - 1;
                while ( n <= s->row2)
                { count = 32 + n * sepe_file->width + col - 1;
                  data = pattern[count];
                  if (data != 0xff)   /* IF THE CONDITION IS
                                         TRUE THEN A TRANSITION
                                         FROM WHITE POINT TO
                                         BLACK POINT IS FOUND  */
                  { h = 0 ;   c = n ;
                    while (h++ <= 7)
                    {  while ( n <= s->row2)
                       { count = 32 + n * sepe_file->width + \
                                                col - 1;
                         data = pattern[count];
                         data <<= (h - 1);
                         bit = data & MASK1;
                         if (bit == 0)
                         {   s->col1 = col ;
                             s->pix1 = h - 1 ;
/* ============================================================== */
/*         COLUMN SEARCHES  FROM RIGHT TO LEFT FOR THE             */
/*         FIRST TRANSITION FROM WHITE POINT TO BLACK POINT        */
/* ============================================================== */

            col = sepe_file->col2 ;
            while (col-->s->col1)
            { n = s->row1 - 1;
              while ( n < s->row2)
              { count = 32 + n * sepe_file->width + col;
                data = pattern[count];
                if (data != 0xff)  /* IF THE CONDITION IS TRUE
                                      THEN A TRANSITION FROM
                                      WHITE POINT TO BLACK POINT
                                      IS FOUND   */
                { c = n ;
                  h = 0 ;
                  while (h++ <= 7)
                  { while ( n <= s->row2)
                    { count = 32 + n * sepe_file->width + col;
```

```
                              data = pattern[count];
                              data >>= (h - 1);
                              bit = data & MASK2;
                              if (bit == 0)
                              { s->col2 = col + 1;
                                s->pix2 = 7 - h + 1;
                                goto end ;  /* ALL THE ROWS AND COLUMNS
                                           IS FOUND , SO TERMINATE THE
                                                SEARCHING    */
                              }
                              n++;
                            }
                            n = c;
                          }
                        }
                        n++;
                      }
                    }  /* col gives right margin   */
                   }
                   n++;
                  }
                  n = c;
                 }
                }
                n++;
               }
              }    /* col gives left margin   */
             }
            }
            n++;
           } /* height gives the last effective row */
          }
         }
        }    /* h gives first effective row */
end:;
}

/* ============   END OF NORMALISE ROUTINE   =============== */
```

```
/* ============================================================ */
/*                        COMPARE ROUTINE                       */
/*        THIS ROUTINE COMPARES THE UNKNOWN PATTERN WITH        */
/*        DIFFERENT  TEMPLATES AND CALCULATES THE AMOUNT        */
/*        OF MISMATCHING BETWEEN THE TWO. IF THE AMOUNT OF      */
/*        MISMATCHING IS LESS THEN THE MISMATCH THRESHOLD       */
/*        THEN THE TEMPLATE CODE IS RETURNED OTHERWISE THE      */
/*        FUNCTION RETURNS NULL                                 */
/*   CALLED BY : RECOGNITION  FUNCTION                          */
/*   CALLS ON  : NONE                                           */
/* ============================================================ */

int compare (char huge *pattern, NORM_INFO *s )
{    FILE *mask , *order ;
     char *sequence_buffer ;
     int pattern_height,pattern_width,mask_width ;
     int yes,no,total,n,counter,c,col,h , bit, headerAddress ;
     int pattern_counter , sequence_counter ;
     unsigned int SequenceFileLength  ;
     long  templateAddress , filePointer ;
     unsigned char pattern_data, mask_data , template_code ;
     char huge *c_mask ;


  if((order = fopen("check.seq","rb"))==NULL)
        { perror ("check.seq"); exit(1);  }
  SequenceFileLength = (int) filelength(fileno(order));
  if (( sequence_buffer = (char *)calloc(SequenceFileLength ,\
                           sizeof(char))) == NULL)
  { cprintf("\n\rSequence buffer allocation failure");
    cprintf("\n\rProgram is aborting ... ") ;
    fcloseall();   exit(1);
  }
  if ( SequenceFileLength != (fread(sequence_buffer,\
              sizeof(char),SequenceFileLength,order)))
  {     cprintf("\n\rCheck order file read error");
        cprintf("\nProgram is aborting ... ") ;
        fcloseall(); exit(1);
  }
  fclose(order);

  if((mask=fopen("template.lib" ,"rb"))==NULL)
  {  perror ("TEMPLATE.LIB");
     cprintf("Program is aborting ... ") ;
     exit(1);
  }
  size = filelength(fileno(mask));
```

```
if ((c_mask = (char *) farcalloc(size, \
            (unsigned)sizeof(char))) == NULL )
{       cprintf("\nTemplate buffer allocation failure");
        cprintf("Program is aborting ... ") ;
        fcloseall();    exit(1);
}
filePointer = 0L ;
do
{       mask_data = fgetc(mask) ;
        c_mask[filePointer++] = mask_data ;
}       while ( ! (feof(mask)) ) ;
fclose(mask);
sequence_counter = 0 ;
while (sequence_counter < SequenceFileLength )
{ template_code = sequence_buffer[sequence_counter++] ;
  headerAddress = ( template_code - 1 ) * 4 ;
  templateAddress =  c_mask[headerAddress] + \
                     256 * c_mask[headerAddress++] +\
                     256*256*c_mask[headerAddress++]+ \
                     256*256*256*c_mask[headerAddress++] ;
  mask_width = c_mask [templateAddress++];
  templateAddress++ ;
  total = 0; yes = 0 ; no = 0 ;
  pattern_height = s->row1 - 1;
  while (pattern_height++ < s->row2 )
  { pattern_counter = 32 + (pattern_height - 1) * \
                        sepe_file->width + s->col1-1 ;
    n = 0 ;    counter = 1;
    pattern_width = s->col1;
    while ( pattern_width++ <= s->col2)
    { pattern_data = pattern[pattern_counter++];
      n++;
      if ( (n == 1) && ( s->pix1 != 0))
        { col = s->pix1; h= 7; pattern_data <<= col; }
      else if ( (n==(s->col2-s->col1+1) &&(s->pix2 != 7)))
        { col = 0 ;   h = s->pix2 ;   }
      else    { col = 0 ; h = 7 ; }
      for ( c = col ; c <= h ; c++ )
      { mask_data = c_mask[templateAddress++];
        counter++;
        bit = pattern_data & MASK1 ;
        if ( bit == 128) bit = 1;
        if ( mask_data == 255) ;
        else if ( mask_data == bit )
             { yes++; total++; }
        else  { no++ ; total++ ; }
        pattern_data <<= 1;
      }
```

```
      }
      while ( counter++ <= mask_width )
        c_mask[templateAddress++];
    }
    n = total*15/100 ;
    if ( no <= n  )  /*  n is  percentage of  mismatch  */
        return ( (int)template_code );
  }
  return ( (int) NULL ) ;
}

/* =============   END OF COMPARE ROUTINE   ================= */


/* ======================================================== */
/*                  RECOGNITION   ROUTINE                   */
/*          CALLED BY : MAIN  FUNCTION                      */
/*          CALLS  ON : ISOLATE_CHARACTER , NORMALISE       */
/*                      AND  COMPARE  FUNCTIONS             */
/*                                                          */
/*          FORMAT OF THE INPUT PATTERN :                   */
/*          4TH AND 5TH BYTE GIVES THE WIDTH OF THE PATTERN */
/*          6TH AND 7TH BYTE GIVES THE HEIGHT OF THE PATTERN */
/*          IN BOTH THE CASES THE FIRST BYTE IS OF LOW ORDER */
/*          PICTURE INFORMATION STARTS FROM 32ND BYTE       */
/*          BLACK SPOT IS REPRESENTED BY   0   AND          */
/*          WHITE POINT IS REPRESENTED BY  1                */
/* ======================================================== */

void    recognise( void )
{    FILE *pattern , *ocr ;
     char ch ;
     char huge *pattern_buffer ;
     NORM_INFO *coor;
     int  a, b, c ;
     unsigned char low_byte , high_byte ;
     long bufferPointer ;
     char drive[3] , dir[25] , file[9] , ext[5] , ocrname[30] ;

  clrscr();
  do
  { show_option (msg[4],msg[5],msg[6],msg[3]);
    ch = get_option ();
    switch ( ch )
    { case '1' : if((spawnl ( 0 , "c:\\scan\\scan.exe" ,\
                               "c:\\scan\\scan.exe" )) < 0 )
                 {   perror("Sorry!"); exit ( 1 ) ;  }
```

```
case '2' : clrscr();
  cprintf("\n\r  Name the pattern file to be recognised");
  cprintf ( "\n\r  (with full pathname) ...");
  cprintf ( "\n\r  ");
  scanf ( "%s" , patternname );
  if (( pattern = fopen ( patternname, "rb" )) == NULL )
  {  perror ( patternname );
     cprintf("Program is aborting ... ") ;
     exit (1) ;
  }
  size = filelength (fileno(pattern));
  if ( (pattern_buffer = (char *)farcalloc ( size , \
                  sizeof(char)))        == NULL)
  { cprintf("\r\nbuffer memory allocation failure");
    cprintf("\r\nProgram is aborting ... ") ;
    exit(1);
  }
  bufferPointer = ØL ;
  do
  {
     pattern_buffer[bufferPointer++] = fgetc(pattern) ;
  }     while ( ! (feof(pattern)) ) ;
  fclose(pattern);

  fnsplit ( patternname , drive , dir , file , ext ) ;
  fnmerge ( ocrname , drive , dir , file , ".ocr" ) ;
  if (( ocr = fopen ( ocrname, "wb" )) == NULL )
  { perror ( ocrname );
    cprintf("Program is aborting ... ") ;
    exit (1) ;
  }
  if ( ( coor = calloc ( sizeof(NORM_INFO) , \
                  sizeof ( char ))) == NULL)
  {      cprintf("\r\nstructure memory allocation failure");
         cprintf("\r\nProgram is aborting ... ") ;
         exit(1);
  }

  if ( ( sepe_file = calloc ( sizeof(SEPE_INFO),\
                  sizeof ( char ))) == NULL)
  {      cprintf("\r\nstructure memory allocation failure");
         cprintf("\r\nProgram is aborting ... ") ; exit(1);
  }
  a = 4;
  low_byte = pattern_buffer[a++];
  high_byte = pattern_buffer[a++];
  sepe_file->width = ( high_byte * 256 + low_byte )/8 ;
             /* width in byte */
```

```
low_byte = pattern_buffer[a++];
high_byte = pattern_buffer[a];
sepe_file->height = high_byte * 256 +  low_byte ; /* height */
a = 0 ; b = 0 ;
sepe_file->condition_col = 0 ;
sepe_file->condition_row = 0 ;
for ( ;; )
{    if ( sepe_file->condition_row < 0 )
        break;
  /* ================================================ */
  /*          initiallising the structure             */
  /* ================================================ */

     if ( a == 0 )
     {
      sepe_file->row1 = 1 ;
      sepe_file->col1 = 1 ;
      sepe_file->pix1 = 0 ;
     }
     else if ( b == 0 )
     {
      sepe_file->col1 = sepe_file->col2 ;
      sepe_file->pix1 = sepe_file->pix2 + 1 ;
     }
     else
     {
      sepe_file->row1 = sepe_file->row2 + 1 ;
      sepe_file->col1 = 1 ;
      sepe_file->pix1 = 0 ;
      b = 0 ;
     }
     isolate_character ( pattern_buffer , sepe_file );
     normalise ( pattern_buffer , coor ) ;
     c = compare ( pattern_buffer , coor );
     if ( c == NULL )
       { cprintf ( " (?)  "); fputc ( '?' , ocr ) ; }
     else
     {      cprintf ( " (%02X)    " , c ) ;
         fputc( c , ocr ) ;
     }

     a++ ;
     if ( sepe_file->condition_col < 0 )
     {         b++ ;
      sepe_file->condition_col = 0 ;
      continue ;
     }
   }
```

```
      farfree ( (char * )pattern_buffer ) ;
      fclose ( ocr ) ;

    case '\x1B':
    case '3' :   break;
   }
  } while ( ch == '1' || ch == '2' );
}

/* ===============   END OF RECOGNITION ROUTINE   ============ */


/* ============================================================ */
/*                 DRAW MASK ROUTINE                            */
/*          THIS ROUTINE DRAWS THE TEMPLATE ON                  */
/*          THE  SCREEN  IN  GRAPHICS MODE                      */
/*          CALLED BY : MAKEMASK  FUNCTION                      */
/*          CALLS  ON : NONE                                    */
/* ============================================================ */

void drawmask ( void )
{     FILE *fp;
      int driver, mode, x = 1, y = 1 ;
      char *buffer ;
      int width, code, counter, length  ;
      long int offset ;

      clrscr();
      cprintf("\nGive the code of the template, please...");
      scanf("%d",&code);
      if ( (fp = fopen ("template.lib","rb")) == NULL )
      {       perror("template.lib");
              cprintf("Program is aborting ... ") ;
              exit(1);
      }
      counter = ( code - 1 ) * 4 ;
      if ( ! ( fseek ( fp, counter, 0 ) ) )
      {    cprintf("\r\nERROR !!!") ;
           cprintf("\r\nProgram is aborting ...") ;
           fcloseall() ;   exit(1) ;
      }
      offset =  fgetc(fp) + 256 * fgetc(fp) + \
                256*256*fgetc(fp) + 256*256*256*fgetc(fp) ;
      if ( offset == 0 )
      {  cprintf("\r\nNo such TEMPLATE in TEMPLATE LIBRARY.");
         return ;
      }
```

```
if ( ! ( fseek ( fp, offset, 0 ) ) )
{  cprintf("\r\nERROR !!!") ;
   cprintf("\r\nProgram is aborting ...") ;
   fcloseall() ;
   exit(1) ;
}
length = fgetc(fp) + 256 * fgetc(fp) ;
if ( ( buffer = calloc( length , sizeof(char)) ) == NULL)
{  cprintf("\nMomory buffer allocation failure.");
   cprintf("Program is aborting ... ") ;
   fcloseall();
   exit(1);
 }
 if ( length != (fread(buffer,sizeof(char),length,fp) ))
 {       cprintf("\nTemplate file read error");
         cprintf("Program is aborting ... ") ;
         fcloseall();
         exit(1);
 }
 fclose ( fp ) ;
 counter = 0 ;
 width = buffer[counter++] ;
 counter++ ;

 for ( code = 10 ; code >= 1 ; code-- )
 {       driver = code ;
         switch ( code )
         {       case 10 : mode = 0 ; break ;
                 case 9  : mode = 2 ; break ;
                 case 8  : mode = 5 ; break ;
                 case 7  : mode = 0 ; break ;
                 case 6  : mode = 1 ; break ;
                 case 5  : mode = 3 ; break ;
                 case 4  : mode = 1 ; break ;
                 case 3  : mode = 1 ; break ;
                 case 2  : mode = 5 ; break ;
                 case 1  : mode = 4 ; break ;
         }
         initgraph(&driver,&mode,"");
         if ( graphresult() == 0 )   break ;
 }
 while ( counter < length )
 {       if ( !(buffer[counter++]) )
                 putpixel (x+320,y+150,1);
         x++;
         if ( x > width) { x = 1; y++; }
 }
```

```
        outtextxy ( 250,340,"Press any key to exit ...");
        getch(); closegraph();
}

/* ===============   END OF DRAW MASK ROUTINE   ============== */


/* ============================================================ */
/*                     DELETE MASK ROUTINE                      */
/*           THIS ROUTINE UPDATES THE  TEMPLATE  BY             */
/*           DELETING A TEMPLATE ON USER OPTION                 */
/*      CALLED BY : MAKEMASK  FUNCTION                          */
/*      CALLS  ON : NONE                                        */
/* ============================================================ */

void deletemask ( void )
{    FILE *fp ;
     char ch ;
     unsigned char mask_data ;
     long filesize, offset, masklength , i, filePointer ;
     ldiv_t findremainder ;
     int code, x, y, n ;
     char huge *buffer ;

        clrscr() ;
        cprintf("Type in the code of the template \
                      to be deleted ...") ;
        scanf("%d", &code ) ;
        cprintf("\r\n\nAre you sure ( Y / N ) ...") ;
        x = wherex() ; y = wherey() ;
        while ( ( ch = toupper(getche() ) ) != 'N' && ch != 'Y' )
        {  printf("\a\a") ;
           cprintf("\r\n\n   Wrong option !!!!  Try again ... ") ;
           gotoxy ( x, y ) ;
        }
        if ( ch == 'N' ) ;
        else
        {  if (( fp = fopen ( "template.lib" , "r+b" ) )== NULL)
           {  perror("template.lib") ;
              cprintf("Program is aborting ... ") ;  exit(1) ;
           }
           filesize = filelength ( fileno ( fp ) ) ;
           if (( buffer = (char*) farcalloc ( filesize , \
                              sizeof ( char) ) ) == NULL)
           {  perror("buffer allocation failure") ;
              cprintf("Program is aborting ... ") ;
              exit(1) ;
           }
```

```
/*  reading the TEMPLATE.LIB  into memory  */
          filePointer = 0L ;
          do
          {    mask_data = fgetc(fp) ;
               buffer[filePointer++] = mask_data ;
          }    while ( ! ( feof ( fp ) ) ) ;
/* reading ends here */
          filePointer = ( code - 1 ) * 4 ;
          i = filePointer ;
          offset = buffer[i++] + 256 * buffer[i++] + \
                         256 * 256 * buffer[i++] + \
                         256 * 256 * 256 * buffer[i++] ;
          buffer[filePointer++] = buffer[filePointer++] \
              = buffer[filePointer++] = buffer[filePointer++]\
              = 0 ;
          masklength = buffer[offset++] + \
                         256 * buffer[offset] ;
          offset-- ;
          do
          {  i = filePointer ;
             offset = buffer[i++] + 256 * buffer[i++] + \
                         256 * 256 * buffer[i++] + \
                         256 * 256 * 256 * buffer[i++] ;
             if ( offset == 0 ) goto end ;
             offset -= ( masklength - 2 ) ;
             n = 0 ;
             findremainder = ldiv ( offset , 256L ) ;
             while ( n++ < 4 )
             { buffer [ filePointer++ ] = findremainder.rem ;
               findremainder =ldiv(findremainder.quot,256);
             }
             end : ;
          } while ( filePointer < 1024 ) ;
          filePointer = 0L ;
          do
          {    fputc ( buffer[filePointer++] , fp ) ;
          }    while ( filePointer < offset ) ;
          filePointer = offset + masklength ;
          do
          {    fputc ( buffer[filePointer++] , fp ) ;
          }    while ( filePointer < filesize ) ;
          fclose ( fp ) ;
       }    /* else loop ends   */
}

/* =========  DELETEMASK  FUNCTION ENDS HERE ================ */
```

```
/* ============================================================ */
/*                     MAKE MASK ROUTINE                         */
/*          THIS ROUTINE MAKES  THE  TEMPLATE  FROM              */
/*          THE PATTERNS NAMED BY THE USER. MAXIMUM              */
/*          NUMBER OF PATTERNS CAN BE USED IS 15.                */
/*     CALLED BY : MAIN  FUNCTION                                */
/*     CALLS  ON : SHOW_OPTION , GET_OPTION , DRAWMASK           */
/*     FORMAT OF THE TEMPLATE LIBRARY :                          */
/*     FIRST 1024 BYTES CONTAIN THE ADDRESSES OF THE             */
/*     TEMPLATES. EACH TEMPLATE ADDRESS IS GIVEN BY 4            */
/*     CONSEQUITIVE BYTES ---- FIRST BYTE IS OF LOWEST           */
/*     VALUE, THE 4TH BYTE IS OF HIGHEST VALUE. EACH             */
/*     TEMPLATE IS PRECEDED BY 4 BYTE HEADER. FIRST 2            */
/*     BYTES GIVES [ LOW ORDER BYTE FIRST ] THE LENGTH           */
/*     OF THE TEMPLATE IN BYTES. THE 3RD BYTE GIVES THE          */
/*     WIDTH OF THE TEMPLATE AND THE 4TH BYTE GIVES THE          */
/*     HEIGHT OF THE TEMPLATE. THEN THE TEMPLATE DATA            */
/*     FOLLOWS.                                                  */
/*      0  REPRESENTS   "STABLE BLACK POINTS",                   */
/*      1  REPRESENTS   "STABLE WHITE POINTS"    AND             */
/*     -1  REPRESENTS   "UNCERTAIN POINTS".                      */
/* ============================================================ */

void    makemask (void)
{ FILE  *fp ;
  struct ffblk filename;
  char *black, *buffer , filenames[25], ch1, ch2, ch3 ;
  char drive[3], dir[20], files[9], ext[5];
  char drive1[3], dir1[20], files1[9], ext1[5];
  int  i, file_no = 0, h, c, n, file_found, code ;
  int width, height, bit, col, nn = 0, counter ;
  int max_width = 0, max_height = 0 , maskfilelength ;
  int pattern_counter;
  unsigned char data;
  char *pattern_buffer[15] ;
  long filePointer, fileSize ;
  ldiv_t findRemainder ;


  clrscr();
  do
  { show_option (msg[7],msg[8],msg[9],msg[3]);
    ch1 = get_option ();
    switch ( ch1 )
    { case '1' :
        do
        { show_option (msg[13],msg[14],msg[15],msg[3]);
          ch2 = get_option ();
```

```
switch ( ch2 )
{ case '1':
     do
     {  show_option (msg[10],msg[12],msg[11],msg[3]);
        ch3 = get_option ();
        switch ( ch3 )
        { case '1' :
               if (( spawnl ( 0 , "c:\\scan\\scan.exe" ,\
                              " c:\\scan\\scan.exe") ) < 0 )
               { cprintf("Sorry! Cannot load scanning program");
                 cprintf("\r\nProgram is aborting ...");
                 fcloseall() ;
                 exit(1) ;
               }
          case '2' :
             clrscr() ;
             file_found = findfirst("template.lib",&filename,0);
             if ( file_found )
             { if ( ( fp = fopen ( "template.lib" , "wt" ) ) \
                                                == NULL)
               {     perror ( "template.lib" ) ;
                     cprintf ( "\nUnable to open TEMPLATE.LIB");
                     cprintf ( "\nProgram is aborting ..." ) ;
                     fcloseall() ;
                     exit ( 1 ) ;
               }
               if ( ( buffer = ( char * ) calloc ( 1024 , \
                                    sizeof(char) ) ) == NULL )
               {     cprintf ( "\nUnable to allocate memory");
                     cprintf ( "\nProgram is aborting ..." ) ;
                     fcloseall() ;
                     exit ( 1 ) ;
               }

               if (  1024 != ( fwrite ( buffer , sizeof(char) ,\
                                         1024 , fp ) ) )
               {     cprintf("\nWrite error to disk");
                     cprintf("\nProgram is aborting .. ") ;
                     fcloseall() ;
                     exit (1) ;
               }
               fclose ( fp ) ;
               free ( buffer ) ;
             }
             clrscr() ;
             cprintf ( "\n\rfile name ( pathname  with ");
             cprintf ( "\n\rwild  cards  allowed ) ... \n\r");
             scanf ("%s" , patternname);
```

```
/* ====================================================== */
/*        FINDS ALL THE FILES IN THE DISKETTE            */
/*        MATCHED WITH THE WILD CARDS.                   */
/* ====================================================== */
    fnsplit(patternname,drive,dir,files,ext);
    file_found = findfirst ( patternname, &filename,0);
    while ( !file_found)
    {    fnsplit(filename.ff_name,drive1,dir1,files1,ext1);
         fnmerge (filenames,drive,dir,files1,ext1);
         if ((fp=fopen(filenames,"rb"))==NULL)
         {    perror(filenames);
              cprintf("Program is aborting ... ") ;
              exit(1);
         }
         size = (int) filelength(fileno(fp));
         if((pattern_buffer[file_no]=(char *) calloc((int)size,\
                                   sizeof(char))) == NULL )
         {    cprintf("\nbuffer memory allocation failure");
              cprintf("Program is aborting ... ") ;
              exit(1);
         }
         if( !fread ( pattern_buffer[file_no], ( int ) size ,\
                                   sizeof ( char ) , fp ) )
              cprintf("\nREAD ERROR");
         fclose(fp);
         if ( ( file[file_no] = (NORM_INFO *) calloc ( \
                   sizeof(NORM_INFO),sizeof(char)))==NULL)
         {    cprintf("\nstructure memory allocation failure");
              cprintf("Program is aborting ... ") ;
              exit(1);
         }
         normalise ( pattern_buffer[file_no], file[file_no]);
         if ( max_height < ( h = ( file[file_no]->row2 \
                        - file[file_no]->row1 + 1 ) ) )
              max_height = h;
         if ( max_width < ( h = ( 8 * (file[file_no]->col2\
                        - file[file_no]->col1-1) + \
                        (8-file[file_no]->pix1)\
                        + (file[file_no]->pix2 + 1))))
              max_width = h;
         file_no++;
         if ( file_no == 15 )  break ;
         file_found = findnext(&filename);
    }
```

```
    if ( ( black = ( char * ) calloc ( (int)size = max_width \
              * max_height , sizeof(char) ) ) == NULL )
    {    cprintf ( "\nUnable to allocate memory ..." ) ;
         cprintf ( "\nProgram is aborting ..." ) ;
         fcloseall() ;
         exit ( 1 ) ;
    }
    for (i = 0 ; i < file_no ; i++)
    {   nn = 0 ;
        counter = 1 ;
        height = file[i]->row1 - 1;
        while (height++ < file[i]->row2 )
        { pattern_counter = 32+(height-1)*sepe_file->width\
                                      +file[i]->col1-1;
          n = 0 ; width = file[i]->col1;
          while ( width++ <= file[i]->col2)
          {  data = pattern_buffer[i][pattern_counter++];
             n++;
             if ( (n == 1) && ( file[i]->pix1 != 0))
             {     col = file[i]->pix1; h= 7;
                   data <<= col;
             }
             else if ( (n==(file[i]->col2 - file[i]->col1 +1)\
                                && (file[i]->pix2 != 7)))
                  {  col = 0; h = file[i]->pix2;  }
             else {  col = 0 ; h = 7 ;}
             for ( c = col ; c <= h ; c++ )
             {     bit = data & MASK1;
                   if (bit == 0) black[nn] += 1 ;
                   nn++; counter++;
                   data <<= 1;
              }
              if ( (wherey() ) > 10 )
              {    getch();    clrscr();    }
          }
          while (counter++ <= max_width)
                   nn++;
          counter = 1;
    }
    free ( file[i] ) ;
}           /* for loop with i ends */
c = 0 ; i = 0 ; bit = 0 ;
for ( n = 0 ; n < size ; n++ )
{   if (black[n] <= M)
    {    black [n] = 1; /* white point */
         c++;
    }
    else if ( black [n] >= file_no - M)
```

```
    {     black [n] = 0; /* black point */
          i++;
    }
    else {      black [n] = -1; /* uncertain point */
                bit++ ;
          }
}
cprintf ("\r\nCode of the template... ");
scanf( "%d",&code ) ;
if ( ( fp = fopen ( "template.lib","r+b")) == NULL )
{     perror ("template.lib");
      printf("Program is aborting ... ") ;
      exit(1);
}
fileSize = filelength ( fileno ( fp ) ) ;
filePointer = ( code - 1 ) * 4 ;
if ( ! (fseek ( fp , filePointer , 0 ) ))
{     cprintf("\r\nWRITE ERROR !!!") ;
      cprintf("\r\nProgram is aborting ...") ;
      fcloseall() ;
      exit(1) ;
}
n = 0 ;
findRemainder = ldiv ( fileSize , 256L ) ;
while ( n++ < 4 )
{   fputc ( (int) findRemainder.rem , fp ) ;
    findRemainder = ldiv ( findRemainder.quot , 256L ) ;
}
if ( ! (fseek ( fp , 0 , 2 ) ))
{   cprintf("\r\nWRITE ERROR !!!") ;
    cprintf("\r\nProgram is aborting ...") ;
    fcloseall() ;
    exit(1) ;
}
maskfilelength = (int) size + 2 ;
divide = div ( maskfilelength , 256 ) ;
fputc ( divide.rem , fp ) ;
fputc ( divide.quot , fp ) ;
fputc (max_width,fp);
fputc ( max_height, fp);
n = 0 ;
while ( n < size )
    fputc ( black[n++] , fp ) ;
fclose ( fp );
fcloseall();
break;
```

```
                                    case '\x1b':
                                    case '3'    : break ;
                                }
                              } while ( ch3 == '1' ¦¦ ch3 == '2' );
                              break;
                        case '2'     :  deletemask() ;  break ;
                        case '\x1b'  :
                        case '3'     :  break ;
                      }
                 }      while ( ch2 == '1' ¦¦ ch2 == '2' ) ;
                 break ;
               case '2'     : drawmask() ; break ;
               case '\x1b'  :
               case '3'     : break;
            }
         } while ( ch1 == '1' ¦¦ ch1 == '2' );
}
/* ===============   END OF MAKE MASK ROUTINE   ============== */



/* ============================================================ */
/*                        MAIN  ROUTINE                          */
/* ============================================================ */

void main ( void )
{       char option;

        welcome ();
        mainmenu:
        do
        {       show_option (msg[0],msg[1],msg[2],msg[3]);
                option = get_option ();
                switch ( option )
                {       case '1'  : recognise(); break;
                        case '2'  : makemask (); break;
                        case '3'  : thanks ();
                }
        } while ( option == '1' ¦¦ option == '2' );
}
```

APPENDIX B

TABLE OF THE FREQUENCY OF OCCURRENCE
OF THE BANGLA CHARACTERS

Character name	% of occurrence	Character name	% of occurrence
অ	1.6234	ত	3.6599
আ		থ	0.6319
ই	1.1408	দ	1.7087
ঈ	0.5267	ধ	0.6326
উ	0.3916	ন	3.6762
ঊ	0.0085		
ঋ	0.0036	প	1.9504
এ	0.9781	ফ	0.1621
ঐ	0.0306	ব	3.1282
ও	0.5992	ভ	0.5686
ঔ	0.0021	ম	2.1345
ক	3.7871	য	0.6511
খ	0.5729	র	5.3843
গ	0.9482	ল	2.1836
ঘ	0.1564	শ	1.0029
ঙ	0.0341	ষ	0.2872
চ	0.7136	স	2.0577
ছ	0.7846	হ	1.3107
জ	0.9510	ড়	0.4044
ঝ	0.0725	ঢ়	0.0028
ঞ	0.0043		
		ক্ষ	0.2196
ট	0.9368	য়	1.7073
ঠ	0.2161	ৎ	0.1592
ড	0.1542	ং	0.2239
ঢ	0.0370	ঃ	0.0611
ণ	0.4428	ঁ	0.2090

APPENDIX C

SOURCE CODING OF THE FUNCTION FOR MAKING SKELETON OF THE CHARACTERS

```
void skeleton  ( void )
{ FILE  *fp;
  char *black,filenames[25],ch;
  struct ffblk filename;
  char drive[3],dir[20],files[9],ext[5];
  char drive1[3],dir1[20],files1[9],ext1[5];
  int  i,file_no = 0,h,c,n,file_found;
  int width,height,bit,col,nn = 0,counter;
  int max_width = 0, max_height = 0,y;
  int pattern_counter;
  char maskname [25];
  unsigned char data;

  clrscr();
  cprintf ( "\n\rfile name ( pathname  with ");
  cprintf ( "\n\rwild  cards  allowed ) ... \n\r");
  scanf ("%s" , patternname);
  /* ==================================================== */
  /*            FINDS ALL THE FILES IN THE DISKETTE        */
  /*            MATCHED WITH THE WILD CARDS.                */
  /* ==================================================== */
  fnsplit(patternname,drive,dir,files,ext);
  file_found = findfirst ( patternname,&filename,0);
  while ( !file_found)
  {  fnsplit(filename.ff_name,drive1,dir1,files1,ext1);
     fnmerge (filenames,drive,dir,files1,ext1);
     if ( ( fp = fopen ( filenames, "rb" ) ) == NULL )
        { perror(filenames);exit(1);     }
     size = filelength ( fileno ( fp ) ) ;
     if ( ( pattern_buffer[file_no] = (char *) calloc ( size,\
                                      sizeof(char)))==NULL)
        { printf("\nbuffer memory allocation failure");
          exit(1);
        }
     if( !fread ( pattern_buffer[file_no], size, \
                                          sizeof(char), fp ))
                printf("\nREAD ERROR");
     fclose(fp);
     if((file[file_no] = (INFO *) calloc(sizeof(INFO),\
                                sizeof(char)))==NULL)
     {  printf("\nstructure memory allocation failure");
        exit(1);
     }
```

```
normalise ( pattern_buffer[file_no], file[file_no]);
if (max_height <(h=(file[file_no]->row2 -\
                     file[file_no]->row1 +1)))
   max_height = h;
if (max_width < (h=(8*(file[file_no]->col2\
                - file[file_no]->col1-1)\
                + (8-file[file_no]->pix1)\
                + (file[file_no]->pix2 + 1))))
   max_width = h;
file_no++;
file_found = findnext(&filename);
}
cprintf("\n\rThere are %d files.",file_no);
black = calloc( (size = max_width * max_height),\
                sizeof(char));
for (i = 0 ; i < file_no ; i++)
{  nn = 0; counter = 1;
   height = file[i]->row1 - 1;
   while (height++ < file[i]->row2 )
  {  pattern_counter = 32+(height-1)*file[i]->width\
                          +file[i]->col1-1;
     n = 0 ;
     width = file[i]->col1 ;
     while ( width++ <= file[i]->col2)
     {  data = pattern_buffer[i][pattern_counter++];
        n++;
        if ( (n == 1) && ( file[i]->pix1 != 0))
        {  col = file[i]->pix1; h= 7;
           data <<= col;
        }
        else if ( (n==(file[i]->col2 - file[i]->col1 +1) && \
                   (file[i]->pix2 != 7)))
              {  col = 0; h = file[i]->pix2;}
        else { col = 0 ; h = 7 ;}
        for ( c = col ; c <= h ; c++ )
        {  bit = data & MASK1;
           if (bit == 0) black[nn] += 1 ;
           nn++; counter++;
           data <<= 1;
        }
      }
      while (counter++ <= max_width)
                nn++;
      counter = 1;
  }
}            /* for loop with i ends */
c = 0 ; i = 0 ; bit = 0 ;
```

```
        for ( n = 0 ; n < size ; n++ )
        {  if (black[n] <= M)
           {  black [n] = 1; /* white point */ c++;}
              else if ( black [n] >= file_no - M)
              {  black [n] = 0; /* black point */ i++;}
              else {     black [n] = -1; bit++ ; }
        }
        fcloseall();
}
```

APPENDIX D

THE CHARACTERISTICS OF THE SCANNER USED

The output of the scanner depends on the followings :

1. The speed with which the scanner is being moved. If the scanner is moved slowly then the letter has a tendency to elongate. On the other side the letter shrinks vertically if the scanner is moved swiftly.
2. The position of the LIGHT...DARK slide control. Rotating the setting ccw will produce a darker picture.
3. The pressure of the scanner on the paper. If the pressure is more then the output will be darker. With the same LIGHT...DARK setting less pressure will result in a less dark output.
4. The darkness of the printed matter.