

**Design and Implementation of a Portable Omnifont Reading Aid for the Blind**

by

**Nikos Asimopoulos**

Dissertation Submitted to the Faculty of the  
Virginia Polytechnic Institute and State University

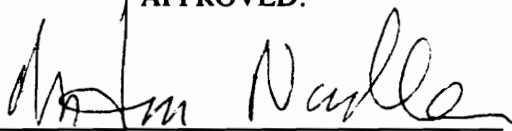
in partial fulfillment of the degree of

**DOCTOR OF PHILOSOPHY**

in

**Electrical Engineering**

**APPROVED:**



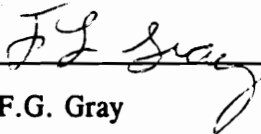
**M. Nadler**



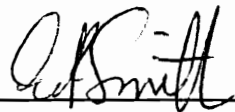
**I. M. Besieris**



**R.O. Claus**



**F.G. Gray**



**E. P. Smith**

July, 1990

Blacksburg, Virginia

Copyright © 1990, Nikos Asimopoulos. All rights reserved.

# **Design and Implementation of a Portable Omnifont Reading Aid for the Blind**

by

Nikos Asimopoulos

Morton Nadler, Chairman

Electrical Engineering

(ABSTRACT)

The design and implementation of a handheld scanner that can help sight-impaired or even blind users to manually scan and read text is discussed in this dissertation. A thorough investigation of all the elements involved in such a system is presented and optimal solutions are proposed. A unique velocity compensation technique based solely on optical information obtained by the scanning device is discussed and a real time segmentation technique based on topological properties (Quasi-Topological Codes) of connected segments is presented. A skew detection algorithm is discussed that can trace typed and printed text manually scanned with skew up to 15 degrees and can guide blind users to properly scan a document. Real time extraction of quasi-topological codes for automatic text recognition and the hardware implementation is also discussed in this work. A hierarchical optical character recognition method is proposed which is based on syntactic and metric analysis of the Quasi-Topological Codes and their position in the scanned image. The proposed method can recognize characters stretched to approximately two times their original width or rotated by a few degrees. Finally, an automated iterative learning process is discussed which includes generalization of the recognition logic and dynamic adaptation of the syntactic and metric recognition rules.

## **Acknowledgements**

I wish to express my gratitude to Professor M. Nadler for his continuous guidance and advice. I am very grateful for his encouragement during the difficult moments of this project.

I wish to thank Professors I.M. Besieris, F.G. Gray, R.O.Claus and E.P Smith for serving on my Ph.D. advisory committee and for their support and patience during my lengthy graduate studies.

I need also to acknowledge my thanks to Brian Cruikshank and Jim Tatem for their contributions in building and debugging the complicated pieces of hardware that were developed for this project.

I am greatly indebted to Image Processing Technologies for their understanding during this thesis.

Finally, I need to express my appreciation to my family. Without their patience, moral and financial assistance, I could not have the opportunity to study for my Ph.D degree.

This work has been supported, in part, by the National Institute of Eye under grant No 3 R44 EY06245-03S1.

# Table of Contents

<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>2. PROPOSED SYSTEM SPECIFICATIONS.....</b>	<b>7</b>
2.1 Velocity Variation and Compensation.....	8
2.2 Feature Extraction and Segmentation.....	9
2.3 Skew Detection and Guidance Subsystem.....	10
2.4 Text Reformatting Subsystem.....	11
2.5 Recognition Subsystem.....	11
2.5.1 Preclassification.....	11
2.5.2 Metric Tests .....	12
2.5.3 Postprocessing.....	12
<b>3. VELOCITY COMPENSATION.....</b>	<b>15</b>
3.1 General Description.....	15
3.2 The Algorithm.....	16
3.3 Implementation.....	17
3.4 Results.....	18
<b>4. SEGMENTATION.....</b>	<b>24</b>
4.1 Quasi Topological Codes (QTCs).....	24
4.2 QTC Description and Detection.....	25
4.3 Segmentation Algorithm Based on Topological Properties of QTCs.....	32
4.4 An Example.....	42
<b>5. TEXT FORMATTING AND ACOUSTIC FEEDBACK.....</b>	<b>46</b>
5.1 Handheld Scanner.....	46
5.1.1 Problem Specification.....	46
5.1.2 The Algorithm.....	49
5.1.3 Comments.....	55
5.2 Page Scanner.....	55
5.2.1 Problem Specification.....	55
5.2.2 Implemented Algorithm.....	57

5.2.3	Comments.....	61
<b>6.</b>	<b>RECOGNITION LOGIC.....</b>	<b>62</b>
6.1	QTCs in Pattern Recognition.....	62
6.2	Grammar Description.....	67
6.3	Preclassification.....	70
6.3.1	Details.....	70
6.3.2	State Automaton Implementation .....	71
6.3.3	Comments.....	74
6.4	Metric Tests.....	75
6.5	Postprocessing.....	82
6.5.1	Application Specific Postprocessing.....	83
6.5.2	Broken Character Reconstruction.....	83
<b>7.</b>	<b>LEARNING PROCESS.....</b>	<b>84</b>
7.1	Labeling Process.....	84
7.2	Iterative Design of Preclassification Tables.....	89
7.3	Knowledge Generalization.....	90
7.3.1	Generalization of Suffixes.....	91
7.3.2	Generalization of Internal Paths.....	91
7.4	Results with Russian and Roman Sets.....	95
7.4.1	Generalized Versus Ungeneralized Learning.....	95
7.4.2	Generalized Learning Statistics.....	98
7.5	Iterative Design Process of Metric Tests.....	114
7.5.1	Automatic Adjustment of Strong Tests.....	114
7.5.2	Manual Adjustment of Weak Tests.....	114
7.6	Results with Russian and Roman Fonts.....	115
7.7	Comments.....	119
<b>8.</b>	<b>SUMMARY AND CONCLUSIONS.....</b>	<b>120</b>
	<b>REFERENCES.....</b>	<b>125</b>
	<b>VITA.....</b>	<b>129</b>

# 1. INTRODUCTION

Optical character recognition has been an active field of research for more than fifty years. Researchers have been interested in automatic text recognition in a variety of applications, the simplest being recognition of single font numerals. The limited vocabulary and the simplicity of the shapes, as well as the possible profits from a successful recognition technique, made the study of recognition of numerals very attractive to the first researchers. Later, when more sophisticated hardware became available, the researchers advanced to more ambitious projects, which included the automatic recognition of typed or printed text, initially in a single font and then in multiple fonts, and handprinted text or even script. One of the major problems that had to be solved was the large variety of typesets and size of machine-generated text and, to a greater degree, the infinite number of variations of handwritten text. Currently available optical character recognition systems are capable of recognizing almost any kind of typed or printed material and even separating graphics from text. All the initial recognition systems and the vast majority of the current recognition techniques assume that the target text is scanned with a mechanical device which will generate a true and accurate image of the scanned text, limiting, thus, the variations of the character images.

Several recognition methods have been proposed in the course of more than half a century, which attempted to generalize the number of characters that could be reliably recognized and at the same time minimize the amount of processing that was required. As in any other application of pattern recognition, two major approaches were considered. Initially, statistical methods were employed to recognize characters, and later, when the need for more flexible techniques arose, syntactic or structural methods were developed. Present optical character recognition systems employ hybrid methods that incorporate both structural and statistical methods to achieve high recognition rates.

The first optical character recognition systems were based on the method of masks or templates. According to that method, the image of an unknown character was tested against a

set of prototype characters (image templates or masks) and was recognized as the one with which it best matched. Initially the masks were actually optical ones, thus the name of the entire field - optical character recognition, while later they were replaced by electrical or electronic masks that represented the character image. Representative applications of recognition systems using the method of masks can be found in [1] through [6]. Today's sophisticated bit correlation methods employed by successful commercial products [48] can also be considered as an extension of the methods of masks. Major advantages of this statistical approach are the relatively simple computational requirements and the capability to recognize broken characters. Disadvantages include the fact that this method can be very slow, if a large vocabulary must be considered, and its lack of flexibility when similar but not identical characters are involved.

Another recognition method which was developed in parallel with the method of masks, is based on a structural analysis of the shape of unknown characters. According to structural pattern recognition techniques, a set of simplified features or primitives is extracted from the unknown bitmap and their topological relation is analyzed to identify the character [6, 7, 8, 9, 10]. Several techniques have been proposed and implemented by researchers, which can be characterized by the features that are detected and the way these features are analyzed.

One very popular technique is the skeletonization of the characters by thinning [11, 12, 13, 14] and analysis of the resulting skeletons. According to this technique the bitmap of a character is reduced by some algorithmic procedure [13, 14] to one pixel wide lines. These lines are then analyzed and a set of features is extracted; the latter is tested against a set of rules that formally describe valid characters. Several researchers proposed a variety of features and formal grammars that can reliably recognize a large variety of characters. The basic disadvantage of skeletonization is the complexity of the thinning process and analysis of features. In addition, small variations of the bitmaps can cause major alterations of the skeleton [14].

Another structural method is stroke analysis [16,17,18]. It is based on the natural way with which humans write characters, that is by a sequence of strokes. As an example, we mention the system proposed by Pavlidis et al. [16], where the orientation and length of each detected stroke is used to describe a character.

Freeman [19,20] introduced the chain codes, yet another way of describing a character. According to his proposal, the contour of a character is described by a sequence of codes that represent the connectivity of the contour pixels. These codes can then be grouped in larger groups, such as straight lines, arcs, etc., and these generalized features can be used to describe a character. A major disadvantage that chain codes present is that they vary with minute changes of a boundary of a character; despite this difficulty, several commercial products are based on chain code analysis [21].

A variation of chain codes is the polygonal approximation of characters [22, 23, 24], whereby the contour of a character is approximated by a sequence of straight lines. The resulting polygon is then used to describe the associated character.

Nadler [25, 26, 27, 28, 29] proposed the use of quasi-topological codes to describe characters. His technique is based on simple topological properties of characters such as start, finish, open, close along two or more directions. We will discuss his proposal later in this work because it presents several major advantages compared to the above recognition methods, the most important being that features can be extracted with only one systematic pass over the bitmap, as opposed to multiple passes required by other character recognition methods.

Several other techniques have been proposed, such as mathematical signatures [30, 31], but they never proved to be reliable recognition methods.

A number of researchers has been involved in the recognition of handprinted text and script. As every person has a unique writing style, it is very difficult to design a general recognition system that will perform with great accuracy. Most of the proposed systems are based on some additional information, particularly dynamic information about the generation of a character. A, so called, on-line recognition system analyzes the sequence of strokes as they are written by an individual and utilizes this information to recognize characters [32, 33, 34, 35, 36, 37].

We need to mention, finally, that even the best recognition methods cannot generally reach exceptional recognition rates without some additional contextual analysis. Contextual analysis for improved recognition capabilities has been proposed by several researchers and has

been applied to a number of commercial and scientific applications. Unidentified characters are extrapolated via spelling correlations or other syntactic analyses [38, 39, 40, 41], improving thus the overall recognition rates.

Optical character recognition is a very active field in commercial data processing applications. The need of automatic data entry is continuously growing, resulting in a very competitive race among companies offering optical character recognition products. The development of accurate OCR systems requires, in general, extensive tests and evaluations of recognition algorithms, as well as access to high speed and, in most cases, expensive hardware, requirements that resulted in a gradual shift of the research from the academic environment to the commercial world. Most of current optical character recognition breakthroughs are coming from industrial research laboratories and not from the academic field, a fact that leads some researchers to think that the optical recognition problem has been solved and only commercialization of the techniques is required. Unfortunately this is not yet true, as many users and developers can witness.

The shift of the research activities to industry has meant less information exchange among researchers, as each company is trying to surpass its competition by offering better, faster and more sophisticated recognition techniques. As a result, we have not been able to gather detailed information about the recognition techniques employed in commercial products and can only speculate about them from the product specifications and capabilities. The vast majority of the products are based on a combination of some syntactic analysis, complemented by statistical information and contextual analysis of the text. The most popular commercial products are offered by Calera Corp. [42, 46], Caere [43], Kurzweil Computer Programs [49], OCR Systems Inc [47], Xerox [44], etc.

Lately, a new field of data capturing has opened up, with the introduction of small, handheld scanners[50, 51, 52, 53]. Handheld scanners introduced new problems for optical character recognition researchers: the scanning velocity and direction are controlled by the user's hand motion; they cannot, therefore, be kept constant as in a mechanical page scanner.

One way to track the scanning velocity is to place some kind of ruler on the document

and use the marks of the ruler to estimate the travelled distance. This approach was implemented in [53], but required precise and careful motion of the scanning wand; as a consequence, it was soon abandoned.

Another way of tracking the scanning velocity is to have a roller attached to the scanning array and measure the velocity as a function of the angular velocity of the roller[53]. Such a solution requires a steady and constant contact between the roller and the scanned surface, conditions that cannot always be met. An open book, for example, does not provide a flat surface that can be easily scanned by such a device.

A third alternative is to use a two dimensional array sensor instead of a linear sensor and take multiple "snapshots" of the text. When a character is completely inside the viewing area of the sensor, it is acquired and the process continues with the next character[43]. This approach presents several limitations; namely, the size of the scanned text is limited by the size of the array sensor. Since several snapshots must be processed for each character, the size of the two dimensional sensor must be small; otherwise, the recognition speed will be extremely slow, only one line of text can be processed at a time and graphics can not be handled.

Nadler [56] proposed a velocity compensation technique which is based on the uniformity of print and requires no physical contact with the document. We will discuss this technique in detail later on since it is the basis of the velocity compensation system that we developed.

In addition to variable scanning speeds, manual scanning introduces inevitable skews caused by the hand motion. Skewed text lines present the additional problem of properly segmenting the characters and reformatting the scanned text. When text is scanned with considerable skew, multiple lines of text must be processed in parallel and characters must be allocated to the appropriate lines. Currently available optical character recognition systems, require almost perfectly aligned scan; otherwise, they fail to properly recognize characters and reconstruct words. In later chapters we will discuss segmentation techniques and text reconstructing algorithms that can tolerate skews up to 15 degrees.

In the particular case of a reading aid for visually impaired users, an additional problem must be considered. The user cannot know which part of the document is being scanned and if

the scanning head moves parallel to the text lines. It follows, therefore, that some realtime feedback is necessary. This means that all the required processing must be accomplished in real time, as the document is being scanned. None of the existing OCR systems can provide such realtime processing because they all require multiple passes over the digitized image.

In conclusion, we need to mention the problem of accurately estimating the error characteristics of a recognition method. Most of the existing products claim recognition rates better than 99% depending upon print quality. Unfortunately, in real applications this recognition rate drops significantly, as many users have discovered. In this thesis we will apply iterative learning techniques [29], which will be based on real characters sampled from a large variety of documents and provide accurate estimates of the recognition rate.

In this dissertation we will discuss the optical character system that we have developed and optimized for usage by visually impaired persons. In Chapter 2 will present an overview of the proposed system and in subsequent chapters we will discuss in detail each subsystem. In Chapter 3 we will discuss the velocity compensation subsystem and in Chapter 4 we will present the realtime segmentation technique that we developed. A text reformatting algorithm and the corresponding user guiding feedback will be presented in Chapter 5. Finally, in Chapters 6 and 7 we will discuss the recognition techniques and the learning methods we employed to achieve high recognition rates.

## **2. PROPOSED SYSTEM SPECIFICATIONS**

In the Introduction we described the existing optical character recognition systems, the methods they employ to perform the recognition and their limitations that make them unsuitable for a reading aid to visually impaired users.

Here we discuss the problems that are unique to a handheld scanning device and the solutions we found for each of these problems. A general diagram of the system is given in Fig. 2.1. Each block corresponds to one of the subsystems described below; the information that flows between subsystems is shown with arrows from one subsystem to the other.

The designed reading device consists of five basic units:

- a) Handheld wand and related digitizing circuitry.
- b) Velocity compensation unit.
- c) Optional image enhancement unit.
- d) Feature detection and segmentation unit.
- e) Skew detection and text reformatting unit.
- f) Recognition unit.
- g) Voice response unit.

Figure 2.2 shows the configuration of a prototype that has been developed based on an AT-type personal computer. Velocity compensation, feature detection and acquisition have been implemented in hardware, as they have to be synchronized with the scanning head, while the remaining operations are performed by the CPU of the controlling PC. The personal computer functions as a general controller and performs those operations that do not require synchronization with the scanning element. Although this configuration is adequate for a prototype, the final system will use dedicated hardware that will achieve higher speeds and will occupy much smaller space.

## 2.1 Velocity Variation and Compensation

Most of the existing scanning devices use a linear photosensitive array which is placed parallel to the paper width (usually called the main scan direction) and moves perpendicularly to its axis (usually called the subscan direction) in such a way that the entire image is scanned when the array travels the entire length of the paper. The speed with which the array moves along the subscan direction is a function of the desired resolution and is kept constant so that the resolution along the two directions is the same.

In the case of a handheld scanner the velocity with which the array moves along the subscan direction cannot be guaranteed to be constant since the scanning motion is manual. Variations of the scanning speed result in distortions of the scanned image. If the velocity is higher than a certain nominal velocity (which is a function of the scanning array), characters are shrunk along the subscan direction; on the other hand, if the velocity is lower than the nominal velocity, characters are stretched along the subscan direction.

The velocity compensation subsystem that has been implemented as part of this research effort is based on an algorithm proposed by Nadler[56]. The algorithm is based on the following principle:

Calling the main scanning direction  $y$  and the subscan direction  $x$ , an estimate of the travelled distance along the  $x$  direction is obtained by comparing the contrast along the  $y$  direction with the contrast along the  $x$  direction. Since the linear array scans the document with a step or pitch defined by the sensor array geometry, the contrast along the  $y$  direction ( $D_y$ ) is a function of the local document quality and illumination. On the other hand, the contrast along the  $x$  direction ( $D_x$ ) is a function of the speed with which the scanning device advances along the subscan direction. If the scanning speed is such that the resolution is the same along the  $x$  and  $y$  directions, then (assuming relatively uniform print) the contrasts along the two directions should be approximately equal. An estimate of the velocity along the  $x$  direction can be obtained as a function of the ratio  $D_x/D_y$ .

The velocity compensation system that has been implemented, on the basis of the above idea, provided a post-compensation (apparent) velocity in the range of 1.5:1 for an initial (real) velocity variation of 10:1. The maximum scanning velocity is about 20 inches per second. A detailed description of the algorithm and the implemented system is presented in Chapter 3.

## **2.2 Feature Extraction and Segmentation**

A reading aid for visually impaired users must be able to extract and process all the information necessary for recognition and guidance in real time and with a single pass over the document. Acoustic feedback must guide the user to scan complete lines of text with minimal skew. The above specifications prohibit the relatively simpler solution of storing the digitized bit map of the scanned text and processing it later. Two basic steps (in general the most time consuming steps) of the recognition process, i.e. segmentation and feature extraction, must be completed in real time.

The segmentation scheme we employed is an improved version of Nadler's algorithm [57] and can identify and isolate individual segments in several lines of text. It is based on sequences of Quasi-Topological Codes (QTCs). QTCs have been used to describe the basic features of segments. In the system under consideration, five basic QTCs can be identified:

- Start, which denotes the start of a new segment.
- Finish, which denotes the end of a segment.
- Open, which signals that a previously acquired segment branches into two subsegments.
- Close, which shows that two subsegments of a previously acquired segment merge back to one common segment.
- External Close, which signifies that two separate segments merge to one segment.

These five QTCs remain unchanged under stretching and under small rotations of the images; as a consequence, they are very well suited to a handheld reading device. The five codes are

detected in two perpendicular directions (main scan direction and subscan direction) and serve a dual purpose. The topological information they provide is used to isolate individual segments and in a subsequent step to recognize each segment. The segmentation algorithm is based on the fact that for every complete segment the number of Starts plus the number of Opens must be equal to the number of Finishes plus the number of Closes plus the number of External Closes. At every new scanned line of the image the detected QTCS are associated with the corresponding segment; a segment is considered complete when the QTCS associated with it meet the above condition. A detailed description of the segmentation scheme is presented in Chapter 4.

## **2.3 Skew Detection and Guidance Subsystem**

A visually impaired user cannot know what kind of text is being scanned, or if the handheld wand scans complete lines of text. The reading system must incorporate logic to guide the user to move the scanner parallel to the lines of text. The feedback must be instantaneous so that the user can correct the scanning direction without losing any information.

The algorithm that we implemented is based on the position of the segments that are detected during the scanning process. When a new segment is acquired, its x and y coordinates are used to find the line of text in which it belongs. The position of each new segment in a line is compared to the last x and y coordinates of that line and an estimate of skew is obtained. When a certain number of segments in a line indicate a consistent skew rate, the user is notified via an acoustic feedback. The exact format of the acoustic feedback has not yet been determined. Tests with visually impaired users will determine an optimal sensitivity and response.

Although the skew detection subsystem can guide the user to move the scanning head very accurately, the user's response may not be fast or accurate. In order to compensate for that, the skew detection subsystem, the text reformatting subsystem and the recognition logic were designed to handle skews of up to 15 degrees. The skew detection algorithm and its implementation are discussed in Chapter 5.

## **2.4 Text Reformatting Subsystem**

As we discussed in a previous section, the segmentation of text is performed in real time and segments are acquired as they are completely scanned by the scanning head. The result of such a segmentation process is that when multiple lines of text are scanned in parallel, segments belonging to several lines of text will be acquired in random order. These segments must be reassembled so that the original text will be regenerated. The text reformatting procedure that we developed is an extension of the skew tracing subsystem that we mentioned in section 2.3. A list of text buffers is used to store the reformatted text. When a new line of text is detected, a text buffer is linked to it and all the text that belongs to that specific line is stored to the linked buffer. The x and y coordinates of each acquired segment are used to find the line of text in which it belongs and the ASCII code that corresponds to the segment is added to the appropriate text buffer. A detailed discussion of the text reformatting process is included in Chapter 6.

## **2.5 Recognition Subsystem**

The recognition logic and process had to be tailored to the specific needs and properties of the entire system. As we mentioned earlier, the digitized bit map is not available; therefore, the recognition had to be based on the features extracted during the segmentation process. These features are the QTCs describing the topology of each segment and a set of coordinates associated with each QTC, that mark the position of the QTCs within the segment. The recognition process we used is a three step hierarchical approach that incorporates structural and statistical methods.

### **2.5.1 Preclassification**

The first step of the recognition process, which we will call preclassification, is purely structural. The QTCs are the primitives of a linear grammar with one rule (concatenation). The

preclassification grammar is implemented as a finite state automaton that accepts valid strings of QTCs. When a string of QTCs is fed to the preclassifier, the latter will respond with one of three responses:

a) If the QTC string is not valid, it will be rejected and the corresponding segment will be flagged as unknown.

b) If the QTC string is a unique valid sequence, the preclassifier will output the ASCII code associated with the input string.

c) If the QTC string is a valid sequence that represents more than one ASCII code, the preclassifier will output a set of similar characters (which we call a confusion class) and will call the second step of recognition, which will try to resolve the ambiguity.

The preclassification stage is robust to stretching of characters as the strings of QTCs remain invariant under stretching. A detailed description of the preclassification stage and the learning process we followed to generate the grammar is presented in Chapter 7.

### **2.5.2 Metric Tests**

The second step of the recognition process uses the coordinates of the QTCs within a character to distinguish characters that belong to a confusion class. The metric tests that we used were designed in such a way that they would be as immune to stretching and small rotation as possible. In order to achieve this goal we selected tests that do not depend on relative dimensions along both the main scan direction and the subscan direction. In Chapter 7 we discuss the tests that were selected and the method with which they were generated.

### **2.5.3 Postprocessing**

The final step of recognition uses contextual and metric information to resolve the remaining ambiguities. For example, the first two steps cannot distinguish between a period and the dot above a lower case *i*, or between a 0(zero) and an O(oh) in some fonts. These ambiguities can be resolved only if the surrounding characters are examined. The purpose of the

postprocessing step is to analyze the sequence of ASCII codes associated with a line of text and correct possible errors or resolve existing ambiguities.

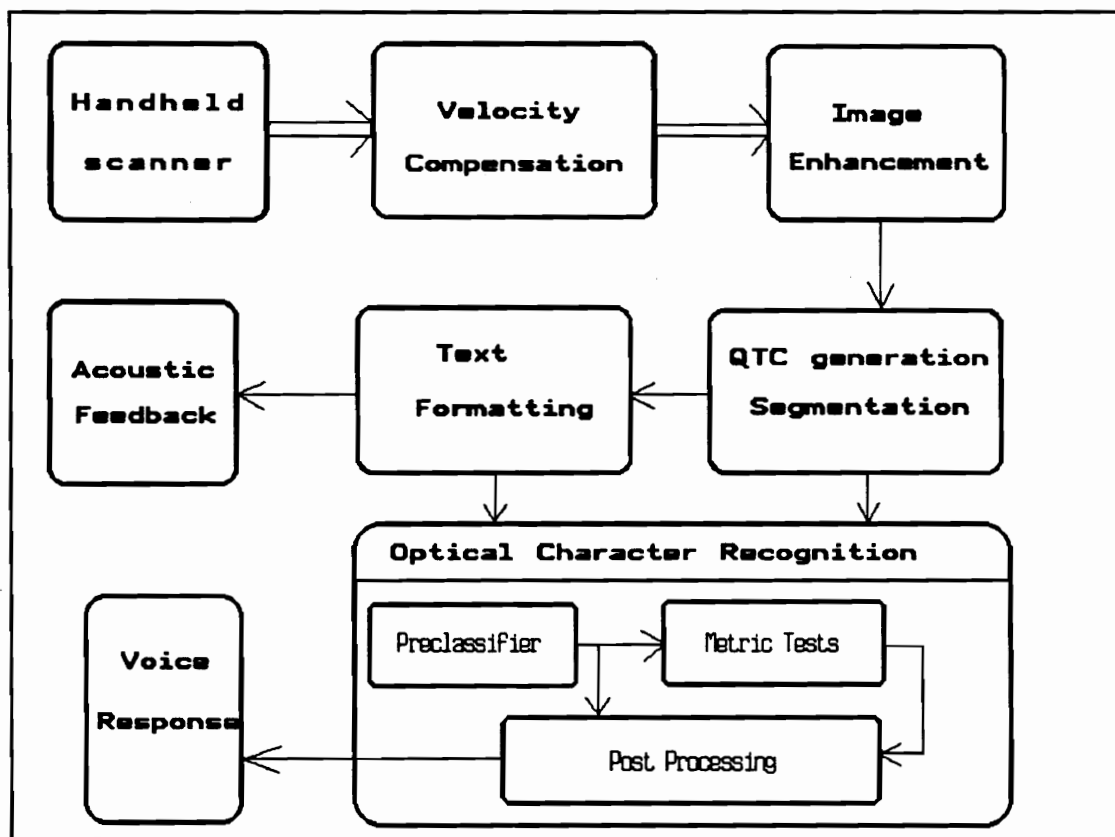
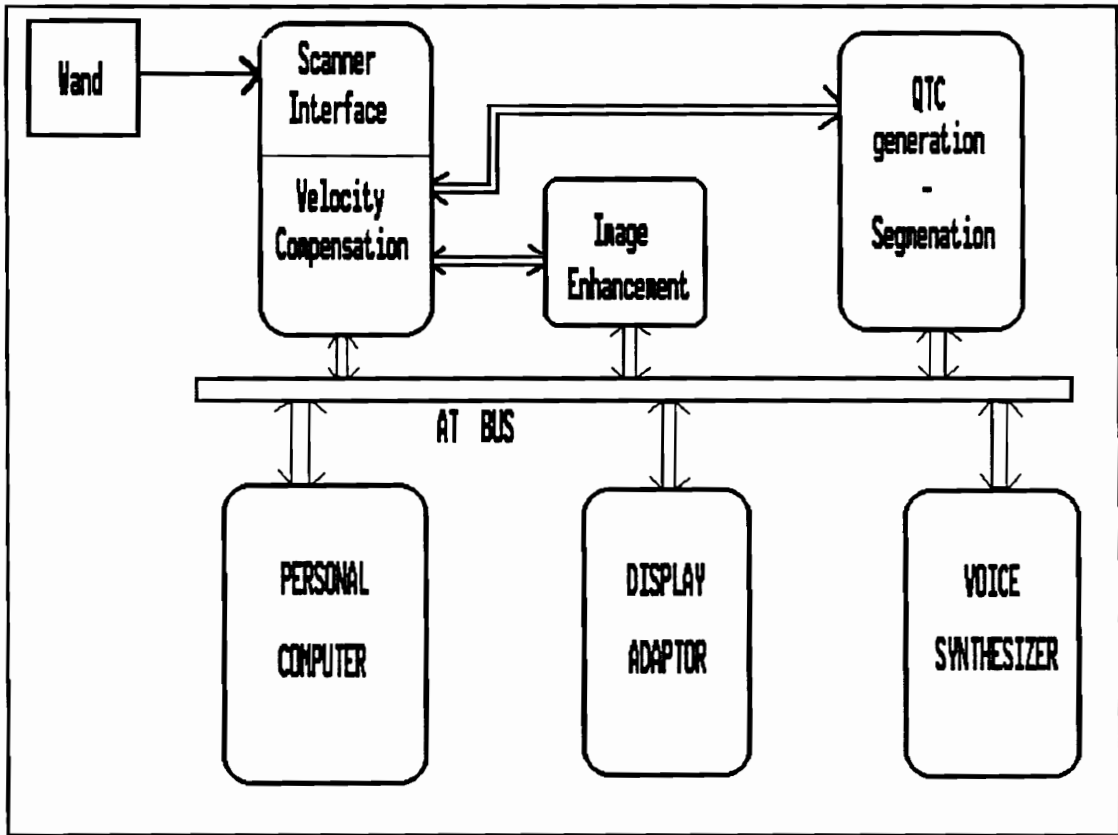


FIGURE 2.1. System Block Diagram.



**FIGURE 2.2.** Prototype configuration.

## 3. VELOCITY COMPENSATION

In this chapter we discuss the velocity compensation subsystem that has been implemented as part of the conducted research.

### 3.1 General Description

The velocity compensation algorithm that we implemented was originally proposed by Nadler [56] and is based on the following principle. Let us call the direction along the scanning linear array  $y$  and the direction along which the scanning head is moved (approximately perpendicular to the  $y$  direction)  $x$ . If we assume relatively uniform print, then an estimate of the travelled distance along the  $x$  direction can be obtained by comparing the local contrast along the  $y$  direction and the local contrast along the  $x$  direction.

As the distance between two consecutive pixels in the  $y$  direction depends only on the geometry of the scanning device, it can be considered constant; then, the local contrast along the  $y$  direction ( $D_y$ ) is a function of the local document quality and illumination only. On the other hand, the contrast along the  $x$  direction ( $D_x$ ) varies with the speed with which the scanner advances along the document. In the case of a handheld scanner this speed is controlled by the user and generally is not constant.

If the velocity along the  $y$  direction is such that the image resolution is the same along the  $x$  and  $y$  directions, then the contrasts  $D_x$  and  $D_y$  along the two directions should be approximately equal. We call the scanning velocity along the  $x$  direction, for which the obtained resolutions in the  $y$  and  $x$  directions are equal, **nominal velocity**. When a document is scanned with nominal velocity then  $D_x$  and  $D_y$  are approximately equal, while if the scanning velocity is less than nominal, then  $D_x$  is a fraction of  $D_y$ .

### 3.2 The Algorithm

Let  $R[i,t]$  denote the  $i$ -th pixel in the  $t$ -th column of video. Then an estimate of the contrast along the  $y$  direction  $DY$  and an estimate of the contrast along the  $x$  direction  $DX$  is computed from the differences between adjacent pixels along the two directions as follows:

**PART A.** For every pixel in a line of video the following steps are executed:

- Step 1. At the beginning of each line contrast estimations  $DX$  and  $DY$  are initialized to 0. If  $i=0$  or  $t=0$  then  $DY=0$  and  $DX=0$ .
- Step 2. The difference between two consecutive pixels of the same column is computed.  
 $Dy[i] = \text{abs}( R[i,t] - R[i-1,t] )$ .
- Step 3. The difference between corresponding pixels of two successive lines is computed.  
 $Dx[i] = \text{abs}( R[i,t] - R[i,t-1] )$ .
- Step 4. The maximum of  $Dy[i]$  is computed and used as an estimate of the contrast  $DY$ .
- Step 5. The maximum of  $Dx[i]$  is computed and used as an estimate of the contrast  $DX$ .

**PART B.** For every line of video the following steps are executed:

- Step 1. The contrast estimate  $DY$  is compared to the last estimate  $DY'$ . If  $DY$  is greater than  $DY'$ , then  $DY'$  is replaced by the new estimate  $DY$ .
- Step 2. The contrast estimate  $DX$  is compared to the last estimate  $DX'$ . If  $DX$  is greater than  $DX'$ , then  $DX'$  is replaced by the new estimate  $DX$ .
- Step 3. An estimate of the distance  $d$  travelled by the scanner along the  $x$  direction is computed as a function of the ratio of the two contrast estimates  $DX'$  over  $DY'$ .
- Step 4. The estimate  $d$  of the travelled distance is added to the last estimate computed during the previous line of video. If the total distance is greater than or equal to the distance between two adjacent elements of the photosensitive array then the current column of video is saved; otherwise, it is discarded.

Step 5. The contrast estimates  $DX'$  and  $DY'$  are decremented by some value in order to prevent local large contrast estimates to propagate through the entire scan.

### 3.3 Implementation

The above velocity compensation algorithm requires two types of computations: Those that must be completed within one pixel time (all steps in Part A) and computations that are required only between two successive lines of video (all steps in Part B). Fig. 3.1 shows the detailed algorithmic sequence of part A as it was actually implemented, while Fig. 3.2 shows in detail the steps of part B of the algorithm.

It can be seen that the computations in part A require fast execution and relatively simple operations. The nature of these operations dictated the need of dedicated hardware that could compute the absolute values of differences and find the maximum of two numbers within one pixel time.

As can be seen in Fig. 3.3, a First In First Out Memory (FIFO) is used to store a complete column of video as required for the computation of  $Dx[i]$ . A pair of Arithmetic Logic Units (ALUs) computes the differences  $Dy[i]$  and  $Dx[i]$  and the carry-out bit is used to invert the function of each ALU if the computed difference is negative, resulting thus in a simple computation of the absolute value of the difference. Following the ALUs is a combination of latches and comparators that find and store the maximum of two differences at every pixel.

At the end of each column of video the maximum contrast estimates  $DX$  and  $DY$  are read by a microcontroller, which performs all the steps of part B. Some additional hardware performs the initializations required at the beginning of each column of video.

Several variants of part B were implemented and tested. Those variants included different functions  $f(d)$  that evaluated the travelled distance as a function of the ratio  $DX /DY$  (Step 3 of Part B) and different decaying rates for  $DX'$  and  $DY'$  (Step 5 of Part B). The detailed sequence of operations that is shown in Figure 3.2 is the one that gave the best velocity

computation results.  $f(d)$  is the normalized log of the ratio  $(DX'/DY')$  and the contrast estimates  $DY'$  and  $DX'$  are decremented by  $2/256$  of their value only when the estimated travelled distance is equal to the distance between two successive scans at nominal velocity. All the steps are executed by an 8751 microcontroller with internal ROM and RAM. It is shown in Fig. 3.3 that if the total estimated travelled distance is equal to the distance between two successive scans at nominal velocity, the microcontroller generates a signal that triggers the column saving circuitry.

### 3.4 Results

The velocity compensation subsystem that was described above has been tested with several documents and illuminations. It has also been tested with a prototype scanning device that could scan documents with variable speeds. The tests were very successful. For scanning velocities that vary between 10 % and 100 % of the nominal velocity, the post-compensation apparent velocities range between 70% and 100% of the nominal velocity. Figures 3.4 to 3.7 show typical results of documents scanned at nominal and minimal velocities with and without compensation.

The illumination of the document is a very important factor for the described algorithm. If the illumination is poor, then the dynamic range of the brightness values read by the photosensitive array is limited and the local differences are very small. Narrow dynamic range results in an inadequate estimation of the local contrasts, which in turn results in a bad estimation of the travelled distance.

A second important factor that is actually related to the illumination is the digitization noise. If the noise in the digital signal is of the same order as the local contrast, then erroneous estimation of the scanning velocity is possible.

The hardware implementation of the algorithm was designed taking into consideration the above factors. An eight-bit Analog to Digital converter is used to digitize the video, providing thus a dynamic range of 256 gray values and the noise was kept below  $4/256$ . The illumination

level was selected in such a way that for typical documents the contrast exceeds by far the noise level, providing thus good estimates of local contrasts  $DX$  and  $DY$ .

When a handheld scanner is used as an input device, the velocity variations are random and can vary by a factor of 10. If the velocity is locally smooth, then the response of the compensator is optimal. Sudden changes in the scanning velocity may cause a slow response which is due to the decaying rate of the stored contrast estimates. This slow response produces a local stretching of characters, which, however, is within the range that the recognition logic can accept.

We must add that, after the logic was tested and the design debugged, part A of the algorithm was implemented in two XILINX(TM) programmable gate array chips. The change was necessary in order to minimize the size of the velocity board and make it suitable for a handheld reading device.

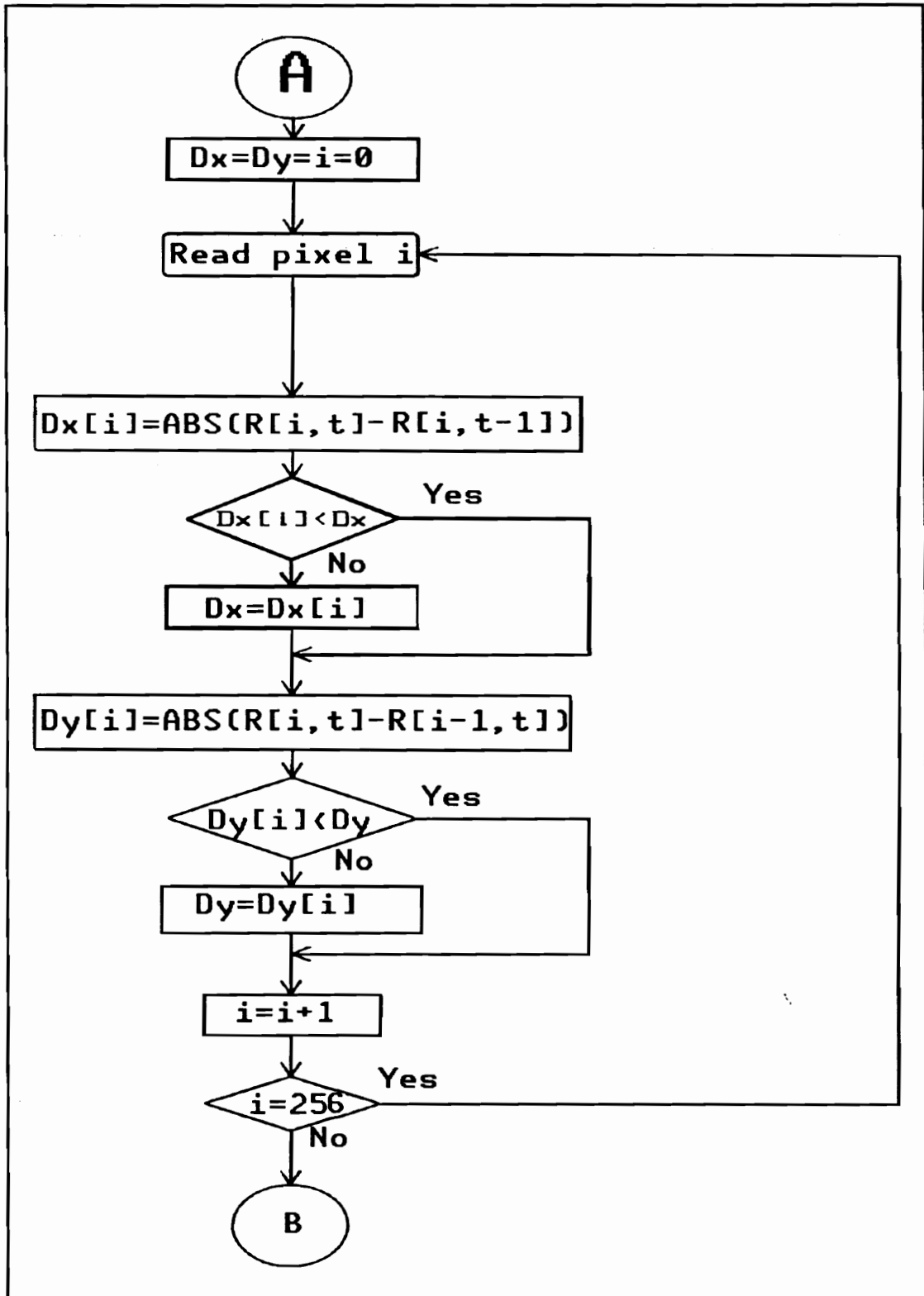


FIGURE 3.1 Velocity Compensation Algorithm (Part A).

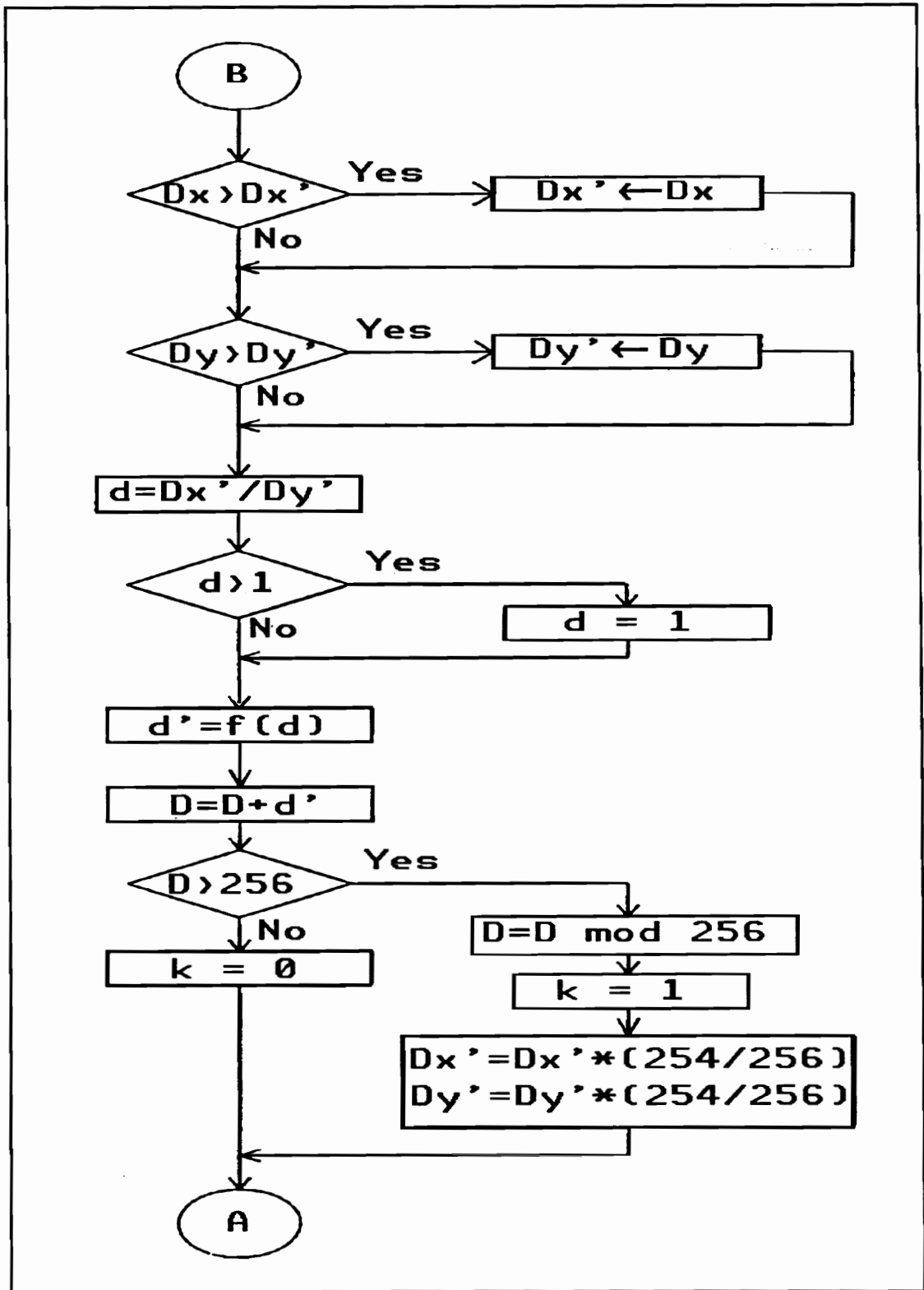


FIGURE 3.2 Velocity Compensation Algorithm (part B).



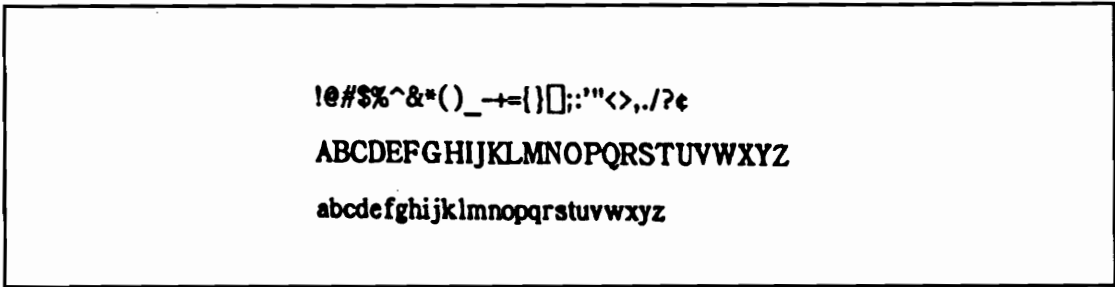


FIGURE 3.4. Document scanned at nominal velocity, no compensation.

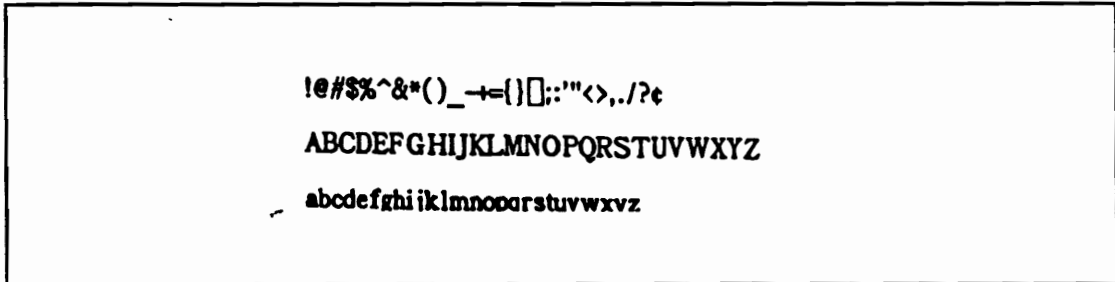


FIGURE 3.5. Document scanned at nominal velocity , with compensation.

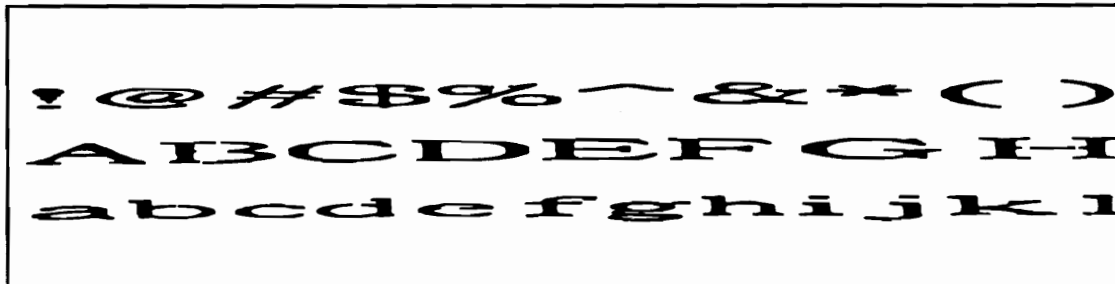


FIGURE 3.6. Document scanned at 10% of nominal velocity, no compensation.

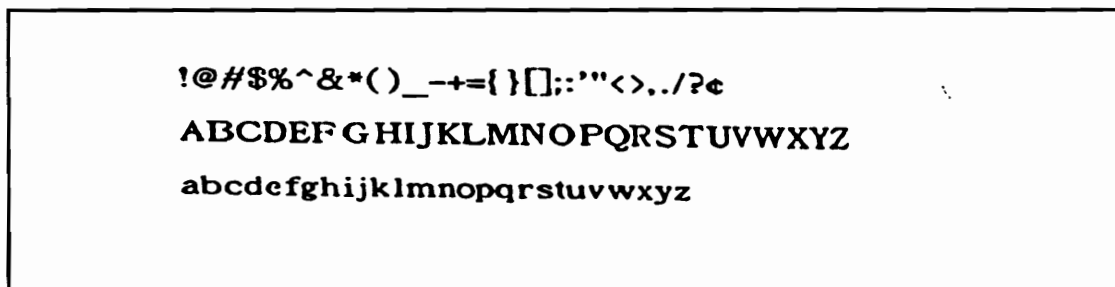


FIGURE 3.7. Document scanned at 10% of nominal velocity, with compensation.

## 4. SEGMENTATION

As we briefly mentioned in Chapter 2, the first task of an optical recognition system is to identify individual characters and separate them from the neighboring segments. This process of isolating continuous graphics objects is generally called segmentation. Several segmentation algorithms have been proposed, as we discussed in the introduction, varying from simple ideas, such as finding a surrounding blank rectangle, to very elaborate solutions used to segment complicated images. Here we are going to discuss the segmentation algorithm, that we employed for this project, which is based on the topological properties of Quasi Topological Codes and which can separate individual segments as the image is being scanned, even if they are occluded by larger objects.

### 4.1 Quasi Topological Codes (QTCs)

QTCs have been used in character recognition to describe the basic features of a scanned character. Nadler [ ] proposed the use of generalized QTCs, that is QTCs detected in more than one direction, to recognize characters. Five basic QTCs can be detected in each direction:

- Start[D]**, which denotes the start of a new segment.
- Finish[F]**, which denotes the end of a segment.
- Open[O]**, which denotes that a previously acquired segment branches to two subsegments.
- Close[C]**, which denotes that two subsegments of a previously acquired segment merge back to one common segment.
- External Close[X]**, which denotes that two separate segments merge to one segment.

In practice, two more codes are detected in addition to the above basic codes. They denote entering [E] and exiting [S] of a segment in the current column of video and are used as book-keeping information to simplify the segmentation process.

In the system developed during this research effort, codes are detected in two directions. QTCs detected along the main scan direction are called vertical and denoted with a "v", while QTCs detected along the subscan direction are called horizontal and denoted with an "h". For example, the sequence of QTCs that corresponds to an "O" is from left to right and top to bottom:

Dv, Ov, Dh, Oh, Ch, Cv, Fv, Fh.

It can be easily seen that for every continuous segment the following equations are true:

$$\text{Number of Dv codes} + \text{Number of Ov codes} = \text{Number of Fv codes} + \text{Number of Cv codes} + \text{Number of Xv codes}.$$
$$\text{Number of Dh codes} + \text{Number of Oh codes} = \text{Number of Fh codes} + \text{Number of Ch codes} + \text{Number of Xh codes}.$$

This idea is the basis of the segmentation algorithm described next (This algorithm is a modification of the one originally developed by Nadler[ ]).

## 4.2 QTC Description and Detection

Pixels that are part of a segment are black, and pixels that are part of the background are white. We assume that all the pixels of the first image line are white and that the first and last pixels of every line are also white. If we assume a systematic raster scan of an image (say top to bottom and left to right), we can define the basic QTCs by analyzing black pixels detected at corresponding columns of consecutive lines. In order to neglect small changes that can be caused by digitization noise, we consider as important only changes that are at least two pixels deep. In order to accomplish this, three image lines are examined in parallel. (If resolutions higher than

250 dpi are required, it may be necessary to increase the width of the viewing window to more than three lines).

The image is scanned systematically from top to bottom and left to right so that at each instance three pixels corresponding to the same column of three consecutive lines are analyzed. If  $i$  is the current image line and  $j$  is the current image column, then pixels  $(i-2,j)$ ,  $(i-1,j)$  and  $(i,j)$  will be analyzed and the next triad of pixels will be  $(i-2,j+1)$ ,  $(i-1,j+1)$  and  $(i,j+1)$ . Based on such a systematic analysis of the image, we can define the basic QTCs as shown in Fig. 4.1.

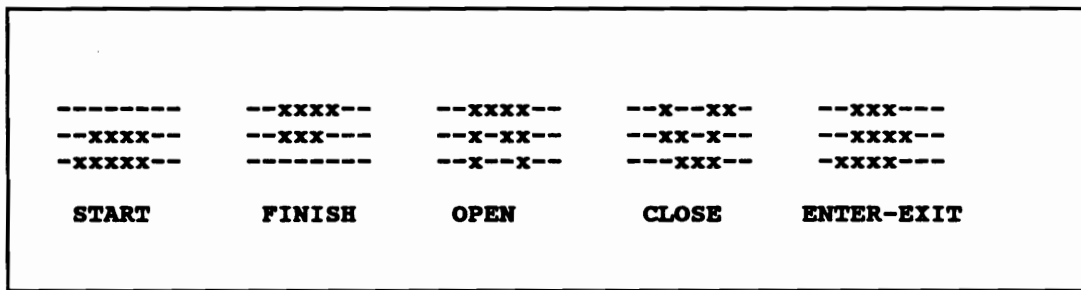


FIGURE 4.1 Quasi-Topological Codes

A Start [D] is detected when a new set of 4-connected black pixels, at least two lines deep, is seen through the three line viewing window. Similarly, a finish [F] is detected when a previously seen set of 4-connected black pixels, at least two lines deep, is interrupted by a line of white pixels. An opening [O] is detected when a set of 4-connected black pixels form a cavity which is two lines deep, and a close [C] is detected when a set of 4-connected black pixels forms a two pixel deep cavity of the opposite direction. When the set of 4-connected black pixels is at least three lines long, then the E and S codes are detected to show that part of a known segment has been encountered.

Quasi-Topological Codes can provide a description of the general topology of a segment, but contain no metric information which is very important in optical character recognition. In order to make this metric information available to the recognition logic, without slowing down

the QTC detection and recognition rate, we tagged each QTC with a set of three coordinates, namely  $x$ ,  $y$  and  $p$ . These coordinates are defined as follows:

$x$  is the image column where the code is detected,

$y$  is the image line where the code is detected,

$p$  is the image column where the code starts.

The meaning of the  $x$ - and  $y$ -coordinates is quite clear; however, the  $p$ -coordinate requires some explanation. The definition of the basic QTC codes is based on properties related to the number of lines a 4-connected set of black pixels occupies. The number of columns that a particular set occupies is not considered. In order to retain this information, an auxiliary code is detected (the  $P$  code) which denotes the possible existence of one of the basic codes; in other words, it marks the possible beginning of a new segment, the possible finish of a segment, or a possible opening or closing. If a basic code is actually detected, then the  $P$  code is validated and its  $x$ -coordinate denotes the column in which the basic code starts. As the only additional information that the  $P$  code provides is the width of a basic code, it is not retained any further; its  $x$ -coordinate is saved as the  $p$ -coordinate of the corresponding basic code.

The detection of QTCs can be implemented as a finite state automaton which is initialized at the beginning of each new line, it accepts as inputs triads of pixels and outputs the corresponding codes. The finite state automaton that generates the QTCs, as they have been described, is given in Fig. 4.2.

As we already mentioned, it is necessary to detect codes in two directions, the main scan direction and the subscan direction. If the entire bit map were available and there were no time restrictions, it would be possible to generate all the codes along one direction and then scan the image from left to right and top to bottom to generate the codes along the second direction. In the particular application however, only one pass is possible and the codes must be detected in real time.

A set of finite state automata can be used to achieve real time detection of codes along the subscan direction. This will be described next. As we discussed previously, the QTC

detection is based only on three consecutive lines of image; as a consequence, the operation of a state automaton in line  $i$  does not depend on a similar automaton operating on any other line.

Suppose that the first line of video is scanned. Then pixels  $(0,0)$ ,  $(0,1)$  and  $(0,2)$  would be the first input to a QTC detector detecting codes along the subscan direction. Let us call  $CURSTATE(0)$  the state of the detector after this triad was analyzed. Then the next state of that particular automaton would depend only on  $CURSTATE(0)$  and the next triad of pixels  $(1,0)$ ,  $(1,1)$  and  $(1,2)$ . If we repeat the procedure with pixels  $(0,1)$ ,  $(0,2)$  and  $(0,3)$  and call the state  $CURSTATE(1)$ , then the next state for that particular automaton would depend only on  $CURSTATE(1)$  and the next triad of pixels  $(1,1)$ ,  $(1,2)$  and  $(1,3)$ . If, therefore, we save  $CURSTATE$  for every pixel of an image line, it is possible to detect all the codes along the subscan direction simultaneously with the codes along the main scan direction. If the horizontal codes are detected simultaneously with the vertical codes then the auxiliary codes E and S (which are used only as book-keeping codes) are not necessary along the subscan direction and the finite state automaton that detects codes along that direction is simplified as shown in Fig. 4.3.

The QTC detector implemented for this project is shown in Fig. 4.4. It accepts image lines 256-pixels long and generates vertical codes and horizontal codes at pixel rate. A set of RAMS is used to store two lines of video as needed for the vertical QTC detector, while another set of RAMS is used to store the states of the horizontal QTC detectors. Two ROMS are used to implement the two finite state automata, while a combination of latches and counters are used to provide the coordinates associated with each detected code. The information related to each code is stored to a set of FIFOs, from where it is read by the segmenter as we shall discuss next.

	000	001	010	011	100	101	110	111
0	0/ -	0/ -	19/ -	2/ P	0/ -	0/ -	1/ P	3/Ev
1	0/Fv	0/Fv	6/ -	9/Ev	8/ -	8/ -	1/ -	3/Ev
2	0/Dv	4/ -	5/ -	2/ -	0/Dv	4/ -	7/Ev	3/Ev
3	0/Sv	10/ P	11/ -	9/ -	12/ P	13/ P	7/ -	3/ -
4	0/Dv	4/ -	19/Dv	2/ -	0/Dv	4/ -	1/DvP	3/Ev
5	0/Dv	0/Dv	5/ -	2/ -	0/Dv	0/Dv	7/Ev	3/Ev
6	0/Fv	0/Fv	6/ -	9/Ev	0/Fv	0/Fv	1/ -	3/Ev
7	0/Sv	0/Sv	11/ -	9/ -	12/ P	12/ P	7/ -	3/ -
8	0/Fv	0/Fv	19/Fv	2/FvP	8/ -	8/ -	1/ -	3/Ev
9	0/Sv	10/ P	11/ -	9/ -	0/Sv	10/ P	7/ -	3/ -
10	0/Sv	10/ -	19/Sv	14/ -	0/Sv	10/ -	1/SvP	3/Cv
11	0/Sv	0/Sv	11/ -	9/ -	0/Sv	0/Sv	7/ -	3/ -
12	0/Sv	0/Sv	19/Sv	2/SvP	12/ -	12/ -	15/ -	3/Ov
13	0/Sv	10/ P	19/Sv	14/ -	12/ P	16/ P	15/ -	3/ -
14	0/Sv	10/ -	17/ -	14/ -	0/Sv	10/ -	7/Cv	3/Cv
15	0/Sv	0/Sv	18/ -	9/Ov	12/ -	12/ -	15/ -	3/Ov
16	0/Sv	10/ P	19/Sv	14/ -	12/ P	16/ -	15/ -	3/Iv
17	0/Sv	0/Sv	17/ -	14/ -	0/Sv	0/Sv	7/Cv	3/Cv
18	0/Sv	0/Sv	18/ -	9/Ov	0/Sv	0/Sv	15/ -	3/Ov
19	0/ -	0/ -	20/ P	2/ P	0/ -	0/ -	1/ P	3/Ev
20	0/DFv	0/DFv	20/ -	2/ -	0/DFv	0/DFv	1/ P	3/Ev

FIGURE 4.2 Vertical QTC detector.

	000	001	010	011	100	101	110	111
0	0/ -	0/ -	19/ -	2/ P	0/ -	0/ -	1/ P	3/ -
1	0/Fh	0/Fh	6/ -	9/ -	8/ -	8/ -	1/ -	3/ -
2	0/Dh	4/ -	5/ -	2/ -	0/Dh	4/ -	7/ -	3/ -
3	0/ -	10/ P	11/ -	9/ -	12/ P	13/ P	7/ -	3/ -
4	0/Dh	4/ -	19/Dh	2/ -	0/Dh	4/ -	1/DhP	3/ -
5	0/Dh	0/Dh	5/ -	2/ -	0/Dh	0/Dh	7/ -	3/ -
6	0/Fh	0/Fh	6/ -	9/ -	0/Fh	0/Fh	1/ -	3/ -
7	0/ -	0/ -	11/ -	9/ -	12/ P	12/ P	7/ -	3/ -
8	0/Fh	0/Fh	19/Fh	2/FhP	8/ -	8/ -	1/ -	3/ -
9	0/ -	10/ P	11/ -	9/ -	0/ -	10/ P	7/ -	3/ -
10	0/ -	10/ -	19/ -	14/ -	0/ -	10/ -	1/ P	3/Ch
11	0/ -	0/ -	11/ -	9/ -	0/ -	0/ -	7/ -	3/ -
12	0/ -	0/ -	19/ -	2/ P	12/ -	12/ -	15/ -	3/Oh
13	0/ -	10/ P	19/ -	14/ -	12/ P	16/ P	15/ -	3/ -
14	0/ -	10/ -	17/ -	14/ -	0/ -	10/ -	7/Ch	3/Ch
15	0/ -	0/ -	18/ -	9/Oh	12/ -	12/ -	15/ -	3/Oh
16	0/ -	10/ P	19/ -	14/ -	12/ P	16/ -	15/ -	3/OCh
17	0/ -	0/ -	17/ -	14/ -	0/ -	0/ -	7/Ch	3/Ch
18	0/ -	0/ -	18/ -	9/Oh	0/ -	0/ -	15/ -	3/Oh
19	0/ -	0/ -	20/ P	2/ P	0/ -	0/ -	1/ P	3/ -
20	0/DFh	0/DFh	20/ -	2/ -	0/DFh	0/DFh	1/ P	3/ -

FIGURE 4.3 Horizontal QTC detector.



### 4.3 Segmentation Algorithm Based on Topological Properties of QTCs

Before we discuss the segmentation algorithm, it is necessary to present the functional elements of the segmenter and define their purpose and structure.

A QTC detected by the QTC detector along with the associated coordinates is stored to a FIFO (called QTC\_FIFO) in the following format:

```
QTC_REC{  
    V_QTC: vertical QTC code.  
    H_QTC: horizontal QTC code.  
    X_CRD: x coordinate.  
    Y_CRD: y coordinate.  
    V_P_CRD: p coordinate of vertical code.  
    H_P_CRD: p coordinate of horizontal code.  
}
```

The QTC\_FIFO serves as a buffer that enables the segmenter and the QTC detector to operate asynchronously. As long as the average rate with which the segmenter reads QTCs from the QTC\_FIFO is equal to or higher than the average rate with which QTCs are detected, the image can be correctly segmented. Asynchronous operation is very important because while the QTC detector detects QTCs at pixel rate, the action that the segmenter needs to take depends upon the sequence of QTCs and varies among QTCs.

The QTCs stored in the QTC\_FIFO are systematically read by the segmenter and, if they are one of the five basic horizontal or vertical QTCs, they are copied to a buffer (BUFFER) allocated to the particular segment to which the QTCs belong. The exact algorithmic process that allocates the buffers and copies the proper QTCs will be presented later.

The buffers that are allocated to detected segments are organized in two FIFOs. The first one (called FREE\_FIFO) contains all the available buffers that have not yet been allocated to a segment, or buffers whose information has been completely processed by the next recognition

steps and can be used again. The second FIFO (called BUSY\_FIFO) contains all the buffers allocated to segments that either are still incomplete or have not been completely processed by the recognition logic. This buffer organization makes it possible to segment large and complicated images using relatively small amount of memory. A maximum number of QTCs (MAXQTC) can be stored in each buffer. If more than MAXQTC codes are associated with a segment, then that particular segment is considered to be graphics and it is not processed by the recognition logic. The format under which data are stored in a buffer is shown below.

```
BUFFER {  
    qtc_index : number of QTCs in buffer.  
    QTC[MAXQTC]: array of stored QTCs.  
    x_crd[MAXQTC]: array of x coordinates.  
    y_crd[MAXQTC]: array of y coordinates.  
    p_crd[MAXQTC]: array of p coordinates.  
    fragment: number of branches in segment.  
    text: flag that marks graphics or text.  
}
```

In the following six pages we present in detail the segmentation algorithm as it has been implemented for the particular project under consideration. We assume that at the end of each image line the QTC detector generates a special code ENDOFLINE.

## SEGMENTATION PROCESS

**BEGIN**

**Initialize structures**

**Clear buffers.**

**Initialize FREE\_FIFO to point to all available buffers.**

**Initialize BUSY\_FIFO to be empty.**

**FOR every image line DO**

**WHILE (QTC\_REC.V\_QTC and QTC\_REC.H\_QTC are not ENDOFLINE)**

**pop new QTC\_FIFO\_REC from QTC\_FIFO.**

**IF (QTC\_REC.V\_QTC is a valid code)**

**IF (QTC\_REC.V\_QTC is Ev) process Ev code.**

**ELSE IF (QTC\_REC.V\_QTC is Sv) process Sv code.**

**ELSE IF (QTC\_REC.V\_QTC is Dv) process Dv code.**

**ELSE IF (QTC\_REC.V\_QTC is Fv) process Fv code.**

**ELSE IF (QTC\_REC.V\_QTC is Ov) process Ov code.**

**ELSE IF (QTC\_REC.V\_QTC is OCv) process OCv code.**

**ELSE IF (QTC\_REC.V\_QTC is DFv) process DFv code.**

**ELSE**

**process horizontal code.**

**END.**

**END.**

**END.**

**END.**

---

**process Dv code**

**BEGIN**

**IF (Dh\_present flag is not set)**

**get new BUFFER N from FREE\_FIFO.**

**clear Dh\_present flag.**

**END.**

**add Dv to BUFFER N.**

**set BUFFER[N] fragment to 1**

**push BUFFER N to BUSY\_FIFO.**

**process horizontal code.**

**END.**

---

**process Fv code**

**BEGIN**

**IF (Dh\_present flag is not set)**

**pop next BUFFER N from BUSY\_FIFO.**

**clear Dh\_present flag.**

**END.**

**add Fv to BUFFER N.**

**process horizontal code.**

**decrement BUFFER[N].fragment by one.**

**IF (BUFFER[N].fragment is 0)**

**IF (next vertical code is null) and (next horizontal code is Dh)**

**set Fh\_pending flag.**

**ELSE**

**clear Fh\_pending flag.**

**send BUFFER N to recognition logic.**

put BUFFER N to FREE\_FIFO.

END.

END.

END.

---

process Ev code

BEGIN

IF (REC\_QTC.H\_CODE is Fh)

process horizontal code.

pop next BUFFER N from BUSY\_FIFO.

ELSE

IF (Dh\_present flag not set )

pop next BUFFER N form BUSY\_FIFO.

ELSE

clear Dh\_present flag.

END.

process horizontal code.

END.

END.

---

process Sv code

BEGIN

push BUFFER N to BUSY\_FIFO.

process horizontal code.

END.

---

process Ov code

BEGIN

push BUFFER N to BUSY\_FIFO.  
increment BUFFER[N].fragment by one.  
add Ov to BUFFER N.  
process horizontal code.

END.

---

process Cv code

BEGIN

pop next BUFFER M from BUSY\_FIFO.  
IF (current BUFFER N is equal to BUFFER M)  
    decrement BUFFER[N].fragment by one.  
    add Cv to BUFFER N.  
ELSE  
    increment BUFFER[N].fragment by BUFFER[M].fragment minus one.  
    add Xv to BUFFER N.  
    merge BUFFER M to BUFFER N.  
    update BUSY\_FIFO after merge.  
    put BUFFER M to FREE\_FIFO  
END.  
process horizontal code.

END.

---

process OCv code

BEGIN

add Ov to BUFFER N.  
add Cv to BUFFER N.  
process horizontal code.

END.

---

**process DFv code**

**BEGIN**

**IF (Dh\_present flag is set)**

**get new BUFFER N from FREE\_FIFO.**

**clear Dh\_present flag**

**END.**

**add Dv to BUFFER N.**

**add Fv to BUFFER N.**

**process horizontal code.**

**send BUFFER N to recognition logic.**

**put BUFFER N to FREE\_FIFO.**

**END.**

---

**process horizontal code**

**BEGIN**

**IF (QTC\_REC.V\_CODE is Fv) and (QTC\_REC.H\_CODE is Dh)**

**add Dh to BUFFER N.**

**ELSE IF (QTC\_REC.H\_CODE is Dh)**

**IF (next vertical code is DFv)**

**get new BUFFER N from FREE\_FIFO.**

**ELSE IF (next vertical code is Dv)**

**get new BUFFER N from FREE\_FIFO.**

**ELSE**

**pop next BUFFER N from BUSY\_FIFO.**

**END.**

**add QTC\_REC.H\_QTC to BUFFER N.**

**set Dh\_present flag.**

```
ELSE IF (Fh_pending flag is set)
    clear Fh_pending flag.
    add QTC_REC.H_CODE to BUFFER N.
    send BUFFER N to recognition logic.
    put BUFFER N to FREE_FIFO.
ELSE IF (QTC_REC.H_QTC is OCh)
    add Oh to BUFFER N.
    add Ch to BUFFER N.
ELSE IF (QTC_REC.H_QTC is DFh)
    add Dh to BUFFER N.
    add Fh to BUFFER N.
ELSE
    add QTC_REC.H_CODE to BUFFER N.
END.
```

END.

---

add V\_QTC to BUFFER N

BEGIN

increment the qtc\_index of BUFFER N by one.

IF (BUFFER[N].qtc\_index less than MAXQTC)

copy V\_QTC and coordinates to the location pointed by qtc\_index.

ELSE

reset qtc\_index.

set text flag equal to FALSE.

END.

---

add H\_QTC to BUFFER N

BEGIN

increment the qtc\_index of BUFFER N by one.

**IF (BUFFER [N].qtc\_index less than MAXQTC)**

    copy H\_QTC and coordinates to the location pointed by qtc\_index.

**ELSE**

    reset qtc\_index.

    set text flag equal to FALSE.

**END.**

---

**merge BUFFER M TO BUFFER N**

**BEGIN**

**IF (BUFFER[N].qtc\_index + BUFFER[M].qtc\_index less than MAXQTC) then**

        append QTCs and coordinates from BUFFER M to BUFFER N.

        set BUFFER[N] equal to BUFFER[N].qtc\_index+BUFFER[M].qtc\_index.

**ELSE**

        reset qtc\_index of BUFFER[N].

        set text flag of BUFFER[N] equal to FALSE.

**END.**

**END.**

---

**update BUSY\_FIFO after merge.**

**BEGIN**

    change all references to BUFFER M to references to BUFFER N.

**END.**

---

**put BUFFER N to FREE\_FIFO**

**BEGIN**

    increment input\_pointer of FREE\_FIFO.

    store N to FREE\_FIFO[input\_pointer].

```
    set BUFFER[N].text equal to TRUE;
    set BUFFER[N].qtc_index equal to 0.
END.
```

---

```
get new BUFFER N from FREE_FIFO
BEGIN
    IF (no more available buffers)
        flag an error
    ELSE
        return the number N of the next available BUFFER.
END.
```

---

```
push BUFFER N to BUSY_FIFO
BEGIN
    increment input_pointer of BUSY_FIFO.
    store N to BUSY_FIFO[input_pointer].
END.
```

---

```
pop next BUFFER N from BUSY_FIFO
BEGIN
    increment output_pointer of BUSY_FIFO.
    return number N of buffer stored in BUSY_FIFO[output_pointer].
END.
```

## 4.4 An Example

We will end this section by following the segmentation steps in a sample image shown in Fig. 4.5, which shows the detailed bit map of four characters (w, i, t and part of h which is connected to the bottom of the t). As the image is scanned from top to bottom and from left to right, QTCs associated with each of the segments are detected systematically from left to right for each image line. These QTCs are plotted on top of the image at the position where they were detected but as it is not easy to distinguish the various codes, they are displayed again in Fig. 4.6. V\_CODES are the codes detected along the main scan direction and H\_CODES are the codes along the subscan direction (when only a V\_CODE or only an H\_CODE is detected then the null code is automatically generated for the other direction).

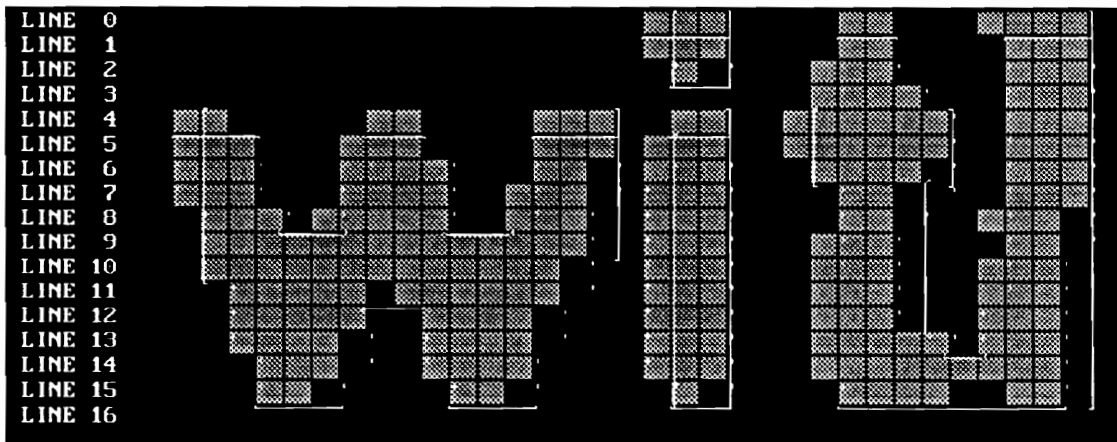


FIGURE 4.5. A sample image and detected QTCs.

The first code appears in line 1 and is a Dv. It denotes the presence of a new segment (in this example it corresponds to the dot above the 'i') and according to the algorithm code Dv is stored to a new buffer say BUFFER 1 (actually in addition to the code itself its x-, y- and p-coordinates are also stored in the same buffer but for simplicity we will ignore them in this example). The second code is also a Dv and indicates the presence of a second segment (the beginning of 't'), therefore a second buffer say 2 is allocated to this segment and code Dv is

stored to that new buffer. A third Dv (which corresponds to 'h') is detected in line 1 and, according to the algorithm, is stored to a new buffer 3.

LINE 1	V_Codes:	D	D	D																
	H_Codes:	φ	φ	φ																
LINE 2	V_Codes:	E	S	E	S	E	S													
	H_Codes:	φ	φ	φ	φ	φ	φ													
LINE 3	V_Codes:	φ	F	E	S	E	S													
	H_Codes:	D	F	φ	φ	φ	φ													
LINE 4	V_Codes:	E	S	E	S															
	H_Codes:	φ	φ	φ	φ															
LINE 5	V_Codes:	D	D	D	D	E	S	E	S											
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ											
LINE 6	V_Codes:	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 7	V_Codes:	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	D	φ	F	φ	φ	φ	φ
LINE 8	V_Codes:	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 9	V_Codes:	E	C	C	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 10	V_Codes:	E	S	φ	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S
	H_Codes:	φ	φ	F	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 11	V_Codes:	φ	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S
	H_Codes:	D	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 12	V_Codes:	E	O	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 13	V_Codes:	E	S	E	S	E	S	E	φ	S	E	S	E	S	E	S	E	S	E	S
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ	O	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 14	V_Codes:	E	S	E	S	E	S	E	S	E	C	S	E	S	E	S	E	S	E	S
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 15	V_Codes:	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E	S	E
	H_Codes:	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
LINE 16	V_Codes:	F	F	φ	F	F	φ													
	H_Codes:	φ	φ	D	F	φ	F													

FIGURE 4.6. QTCs detected in image shown in Fig 4.5.

As we start the second image line, we can either detect the beginning of one or more new segments or encounter the first segment that we detected in line 1. It is not possible to encounter part of segments 2 or 3 before we see the continuation of segment 1. As is shown in Fig. 4.6, the first code detected in line 2 is an E which denotes that we entered an already acquired segment (the dot) and it is followed by an S which indicates that we exited that segment and we

are again ready either for a new segment or for the continuation of segment 2. Two more pairs of E and S codes in line 2 indicate that we have encountered the other two segments.

The first code detected in line 3 is a Dh which is part of segment 1, it is, therefore, stored in buffer 1. Next two codes are detected simultaneously: an Fv and an Fh. They both belong to the same segment, namely the dot, and are stored to buffer 1. At this point buffer 1 contains a Dv an Fv an Dh and an Fh and according to the topological properties we discussed earlier it should contain a complete segment. We can easily see that the dot has actually been completely scanned, therefore, the information in buffer 1 is ready to be processed by the recognition logic. When the recognition of that segment is completed, buffer 1 is returned to the pool of available buffers and can be used again.

In line four we detect two pairs of E and S codes which indicate that the two remaining segments continue and in line five we detect the presence of four new segments say 4, 5, 6 and 7, which correspond to the three peaks of the character 'w' and the top of character 'i'. According to the segmentation algorithm these 4 segments are stored to 4 separate buffers say 4, 5, 6 and 7 respectively. The two pairs of E and S codes follow indicate that segments 2 and three continue but now in a systematic scan from left to right we will encounter first segment 4 then segment 5 and then 6, 7, 2 and 3. Note that at this point there is no indication that segments 4, 5 and 6 belong to the same character. The segmenter knows only that six segments are present.

The codes detected in line six indicate that all six segments continue further and in line seven we detect a Dh and Fh code which belong to segment 2. In line nine we detect a E which says that we entered segment 6 and then we detect a Cv. Since no openings were detected yet, the segmenter joins the two adjacent segments 4 and 5, copies the information from buffer 5 to buffer 4 and frees buffer 5. The second Cv which is detected next results in merging buffers 4 and 6. At the end of line nine the segmenter knows that there are four segments present and their order is from left to right 4, 7, 2 and 3.

In line 12, after the first E code, an Ov is detected, which indicates that segment 4 branches to two separate segments and the E and S pairs that follow indicate that segments 7, 2

and 3 are still incomplete. In line 14, a Cv after the third E code indicates that segments 2 and 3 merge to one common segment, therefore, the contents of buffer 3 are copied to buffer 2 and buffer 3 is freed.

	LINE NUMBER															
	1	3	5	7	9	10	11	12	13	14	16					
1	Dv	DhFvFv -	-													
2	Dv			DhFh						Oh	XvDv	FvFh -				
3	Dv															
4	-	-	Dv		XvDvXvDv	Fh	Dh	Ov				FvFv -				
5	-	-	Dv													
6	-	-	Dv													
7	-	-	Dv													DhFvFh -

FIGURE 4.7. Segmentation buffers for image in Fig 4.5.

Finally, the Fv and Fh codes detected in line sixteen indicate the completion of both branches of the segment stored in buffer 4, the completion of segment 7 and the completion of the merged segments in buffer 3. Figure 4.7 displays the contents of each segmentation buffer at the end of every image line. As we discussed above, BUFFER 1 holds the QTCs associated with the dot above the 'i', BUFFER 4 contains the QTCs detected at the 'w', BUFFER 7 holds the QTCs associated with the 'i' and, finally, the QTCs that correspond to the connected t and h are stored in BUFFER 3.

## 5. TEXT FORMATTING AND ACOUSTIC FEEDBACK

In this section we will address the problem of reassembling the segments detected by the segmentation system, into their proper order and position within the text. In the previous chapter we saw that characters, or segments of characters are not acquired in the order with which they appear in a text line, but in the order in which they are detected by a systematic raster scan. Here we will discuss the steps that are necessary to properly reorganize these segments if documents are scanned from top to bottom (as in a page scanner), or from left to right (as is the case with the handheld scanning device of this project).

### 5.1 Handheld Scanner

#### 5.1.1 Problem Specification

When a document is scanned from left to right with a handheld scanner, multiple lines of text are scanned in parallel and segments belonging to several lines of text are acquired in random order. Since the motion of the scanning head is controlled by the user, considerable skew must also be expected, especially when the user cannot see the text that is being scanned.

Fig. 5.1 is an example of a document scanned with a handheld scanner from left to right. The skew is deliberately exaggerated in order to demonstrate the problem and the capability of the reformatting procedure. Fig. 5.2 shows the order in which segments are detected by the segmentation. Each box corresponds to the circumscribed rectangle that contains a detected segment, and the number in the box shows the order in which the segment is acquired. The sample image in Fig. 5.2 presents several interesting features that will help us discuss the reformatting process and the acoustic guidance feedback. The text consists of five lines that start at various places, include upper, lower case characters and numerals and overlap each other. In addition, there is a considerable upwards and downwards skew.

The text reformatter must be able to correctly associate each segment with the appropriate line of text, and it must have the capability to detect skew in any direction and notify the user accordingly. The feedback must be instantaneous so that the user can correct the scanning direction without losing any information. If the segment could be recognized instantaneously, then that additional information could be used to reformat the text. Unfortunately though, in many cases it is not possible to recognize a segment unless its relation with the neighboring segments is known. For example, a period cannot be distinguished from the dot above an i without analysis of the neighboring segments. For this reason, we decided to design the reformatting and feedback system utilizing only the positional information associated with each segment.

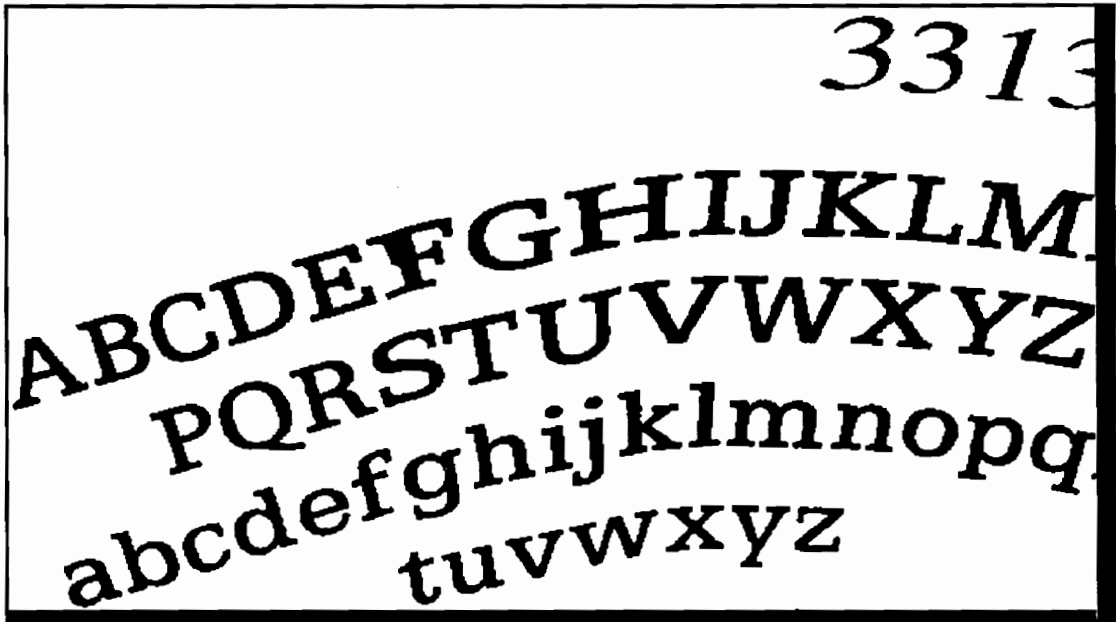


FIGURE 5.1 Sample hand-scanned image.

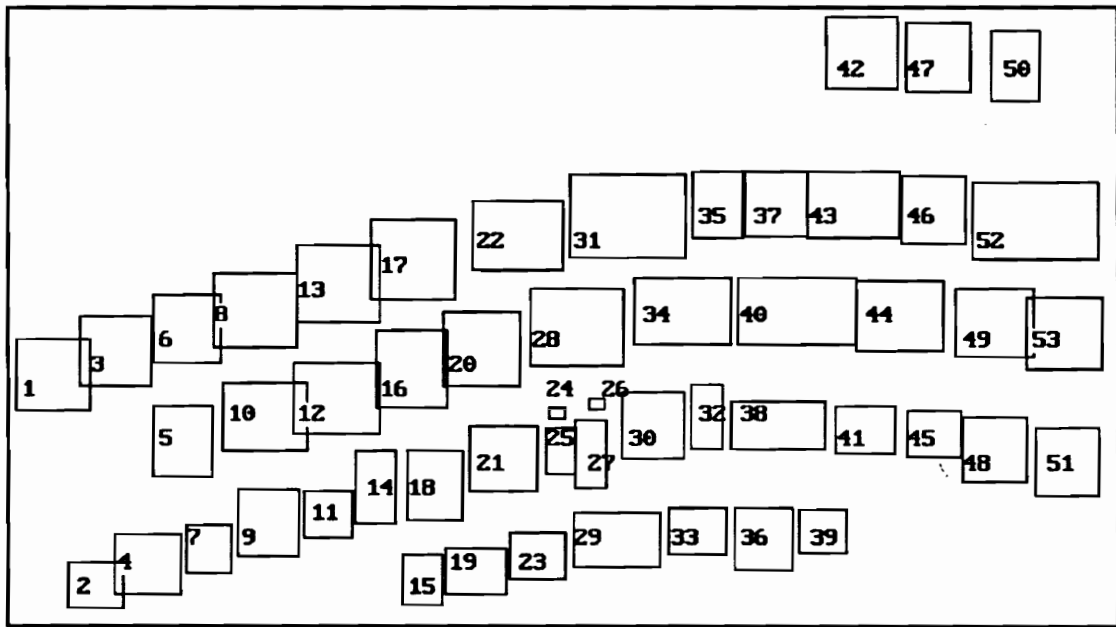


FIGURE 5.2 Order of detected segments of image shown in Fig 5.1

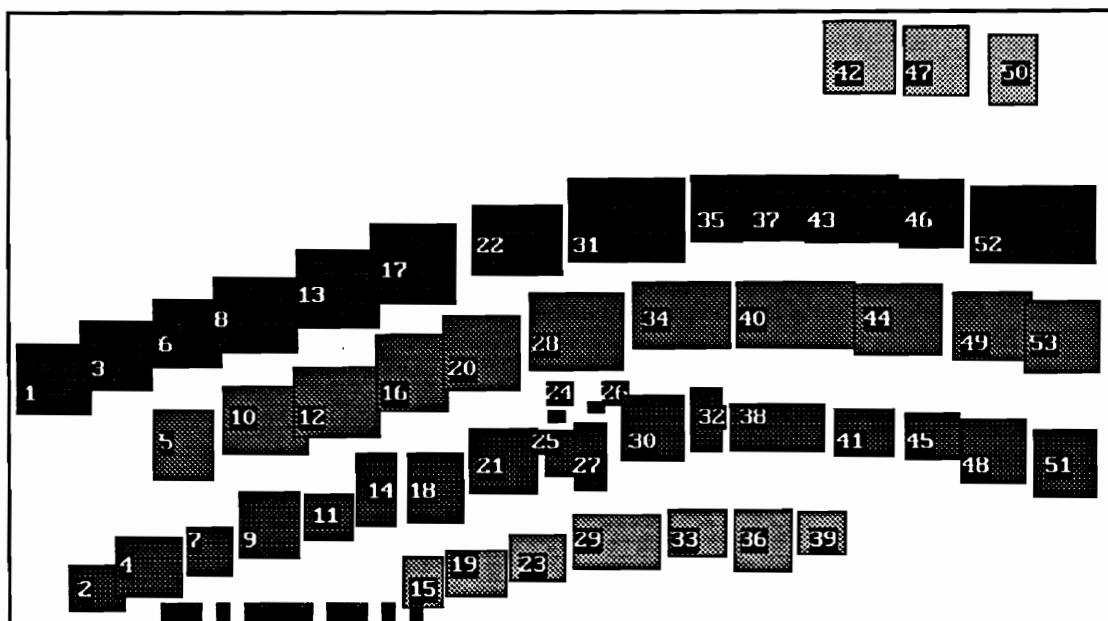


FIGURE 5.3 Line Tracing and Feedback Indication of image in Fig 5.1.

### 5.1.2 The Algorithm

We will use the sample image of Fig. 5.1 in order to discuss the reformatting process and the skew detection mechanism. When a new segment is detected, its position is compared to the position of already acquired segments. If it is "close enough" to the last segment of some text line and "far enough" from the other lines, it is allocated to the closest text line. If, on the other hand, a segment is "far" from all the existing text lines, then it is assumed to belong to a new line of text.

In the sample image of Fig. 5.1 the first complete segment will be (see Fig. 5.2) the upper case 'A' and as there are no other text lines it will be the first segment of a new line say line 1. The second complete segment is the lower case 'a' which is "far" from the last known coordinates of line 1; therefore, it is allocated to a new line say line 2. The third segment 'B' is "close enough" to the last segment of line 1 and "far enough" from line 2; it is, thus, allocated to line 1. Similarly 'b' will be allocated to line 2. The next segment is the upper case 'P' which is "far" from line 2 but could be "close" to line 1. The direction of line 1 though shows that 'P' is probably not part of that line; therefore, it is allocated to a new line say line 3. Fig. 5.3

shows how each segment in Fig. 5.1 was allocated to the proper line of text. (Separate lines are denoted with different shading patterns).

Although this text reformatting idea is simple and easy to implement, it required long experimentation and tests to obtain a good and reliable estimation of what is "close" and what is "far". The final version that we present next performed very well with a large variety of documents and skews up to 15 degrees.

The information associated with each line of text is stored in a structure called ZONE, which has the following format:

```
ZONE {  
    Xmin      : Current minimum X coordinate.  
    Xmax      : Current maximum X coordinate.  
    Xmed      : Current average X coordinate.  
    Ylast     : Current Y coordinate.  
    Width     : Current character width.  
    Text_ptr  : Current number of segments in line.  
    direction : Upward or Downward skew indicator.  
    text      : Array holding the ASCII text in line.  
    Active_flag : Flag indicating available or busy zone.  
}
```

When a segment is acquired, its position is compared to the information stored in each ZONE. If it is found to belong to a certain ZONE then the fields of that ZONE are updated so that at each instant ZONE N contains the latest information associated with the text line N. In the detailed algorithm that follows several constants are used, namely THRES, MAXD and MINSIZE. The actual values of these constants depend upon the scanning resolution and the range of text that is to be scanned. They were experimentally set to handle the most typical text sizes scanned at 200 dots per inch and to allow skews up to 15 degrees.

## **TEXT REFORMATTING AND SKEW DETECTION**

**BEGIN**

read segment I.

**IF** (Number of Active Zones is 0)

get new ZONE N.

update new ZONE N with info from segment I.

**ELSE IF** segment I belongs to ZONE N

update active ZONE N with info from segment I.

check skew of ZONE N.

**ELSE**

get new ZONE N.

update new ZONE N with info from segment I.

**END.**

---

get new ZONE N

**BEGIN**

**IF** no more ZONES

report error.

**ELSE**

initialize ZONE N.

increment Number of Active ZONES by one.

return ZONE N.

**END.**

---

update new ZONE N with info from segment I.

**BEGIN**

set ZONE[N].Active\_flag equal to TRUE.

set ZONE[N].Xmin equal to the minimum x of segment I.

set ZONE[N].Xmax equal to the maximum x of segment I.  
set ZONE[N].Xmed equal to the average of Xmax and Xmin.  
set ZONE[N].width equal to maximum y minus minimum y of segment I.  
set ZONE[N].Ylast equal to maximum y of segment I.  
set ZONE[N].text\_ptr equal to one.

END.

---

update active ZONE N with info from segment I.

BEGIN

IF (size of segment I is less than MINSIZE)

ignore segment I.

IF (ZONE[N].Xmed less than average of min and max x of Segment I)

IF (ZONE[N].direction less than MAXD)

increment ZONE[N].direction by one.

ELSE IF (ZONE[N].Xmed greater than average of min and max x of Segment I)

IF (ZONE[N].direction greater than -MAXD)

decrement ZONE[N].direction by one.

set ZONE[N].Xmin equal to the minimum x of segment I.

set ZONE[N].Xmax equal to the maximum x of segment I.

set ZONE[N].Xmed equal to the average of Xmax and Xmin.

set ZONE[N].Width equal to average of last Width and width of Segment I.

increment ZONE[N].Text\_ptr by the distance between the last two segments divided  
by the average width of the line.

set ZONE[N].Ylast equal to maximum y of segment I.

END.

---

segment I belongs to ZONE N

**BEGIN**

set MIDX equal average of min x and max x of segment I.

**FOR** every active ZONE N

**IF** (MIDX plus a threshold THRES is greater than ZONE[N].Xmin) **AND**  
(MIDX less than ZONE[N].Xmax plus threshold THRES) **AND**  
(distance between min y of segment I and ZONE[N].Ylast is less than two  
times ZONE[N].Width )

segment I belongs to ZONE N.

return ZONE N.

**END FOR.**

**FOR** every Active ZONE N

find ZONE M who's Xmed is closest to min x segment I and Ymax close to  
min y of segment I.

call distance between MIDX of segment and ZONE[M].Xmed DIST\_M.

**END FOR.**

**FOR** every Active ZONE N except M

find ZONE L who's Xmed is closest to min x segment I and Ymax close to  
min y of segment I.

**END FOR.**

**IF** (DIST\_M less than THRES)

segment I belongs to ZONE M.

**ELSE IF** (ZONE[M].direction is positive) **AND**

(MIDX greater than ZONE[M].Xmed) **AND**

(MIDX minus ZONE[M].Xmed less than THRES)

segment I belongs to ZONE M.

**ELSE IF** (ZONE[M].direction is negative) **AND**

(MIDX less than ZONE[M].Xmed) **AND**

(ZONE[M].Xmed minus MIDX less than THRES)

```

        segment I belongs to ZONE M.
ELSE IF (ZONE[L].direction is positive) AND
    (MIDX greater than ZONE[L].Xmed) AND
    (MIDX minus ZONE[L].Xmed less than THRES)
        segment I belongs to ZONE L.
ELSE IF (ZONE[L].direction is negative) AND
    (MIDX less than ZONE[L].Xmed) AND
    (ZONE[L].Xmed minus MIDX less than THRES)
        segment I belongs to ZONE L.
ELSE
        segment I belongs to a new ZONE.
END.

```

---

check skew of ZONE N

```

BEGIN
    IF (ZONE[N].direction greater than MAXSKEW)
        report UPWARDS skew.
    ELSE IF (ZONE[N].direction less than -MAXSKEW)
        report DOWNWARDS skew.
END.

```

The acoustic feedback is based on the skew detection routine presented above. When the direction flag in a line of text becomes higher than a certain value MAXD, then a consistent skew towards one direction has been detected and the user is notified. The actual value of MAXD was experimentally set to 4, a value that seemed to generate a smooth response that was not affected by random variations of the segment positions, such as presence of punctuation marks, characters with long descenders, etc. The exact form of the acoustic feedback has not yet

been finalized. Tests with visually impaired users will determine an optimal sensitivity and response. In Fig. 5.3 we can see an indicative response to a consistent upwards skew between segments 6 and 18 until the scanning direction is corrected. The arrows at the bottom of the figure mark the places where the acoustic feedback will notify the user.

### **5.1.3 Comments**

Although the present text reformatting algorithm and skew detection algorithm can handle a large variety of typical printed material, there are situations for which they may fail to correctly reformat the scanned text. One such situation may occur when characters of very different sizes are present, and the large characters overlap lines of smaller print. In that case, the line formatting process will try to associate all the small characters with the line of larger print until it realizes that more than one line is present. It is possible to resolve these extreme cases but it will require more complicated analysis, which could not be completed within the tight timing requirements of a real-time operating prototype. It is something that will be addressed when dedicated hardware and firmware will be built.

## **5.2 Page Scanner**

### **5.2.1 Problem Specification**

If the scanning device is a page scanner that scans the documents from top to bottom, then the text reformatting process is different from the one applied to handheld scanners. There are two major differences: 1) the document is scanned mechanically so that the scanning motion can be constant and accurate; 2) the lines of text appear one after the other, with distinct blank spaces between them. (It can be safely assumed that there is no significant skew). The constant scanning velocity enables the use of sensitive metric tests to accurately organize the segments and separate words or columns of text, and makes it possible to use metric information obtained from previous lines to format a new line (assuming that several lines of text are of the same font and

size). The task of the reformatting process is then to detect the end of a text line, to sort the associated segments and to reconstruct words and/or columns. The information that is available to the text formatter is again only metric for the same reasons as in the case of a handheld scanner.

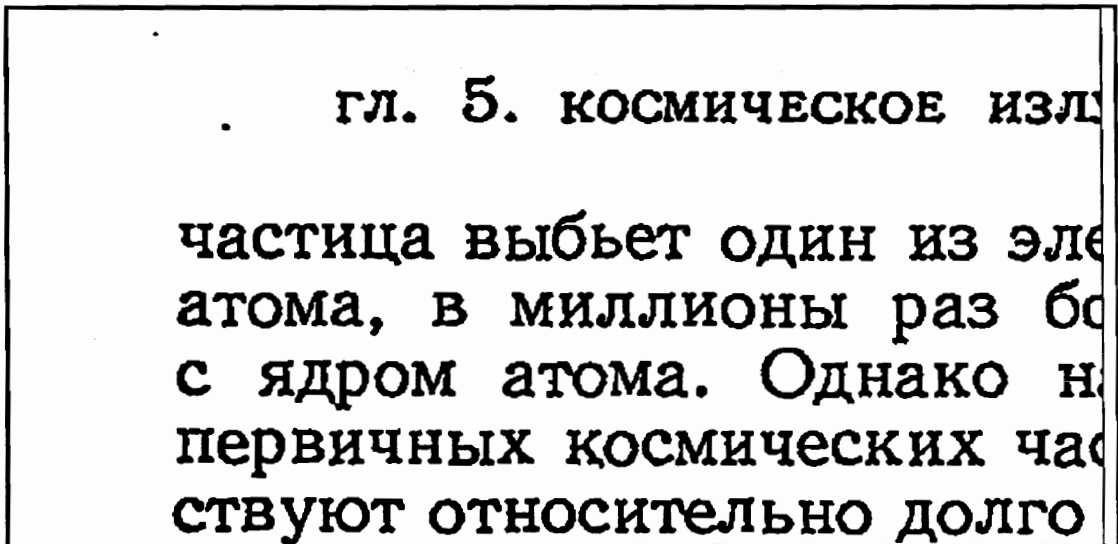


FIGURE 5.4 Sample image scanned with a page scanner from top to bottom.

Fig. 5.4 shows a sample image of Cyrillic text scanned with a page scanner, at 300 dots per inch, from top to bottom and from left to right. Cyrillic text was used for this example to illustrate the generality of the problem and of the proposed reformatting procedure. As is shown in Fig 5.5, segments are detected in random order within a line of text but all the segments in a particular line are completed before any segments of the next text line. ( This may not be true in the presence of skew). As can be seen in Fig. 5.6, the formatting of words and lines is accurate and improves with each new line of text. The detailed text reformatting process is presented in the next section.

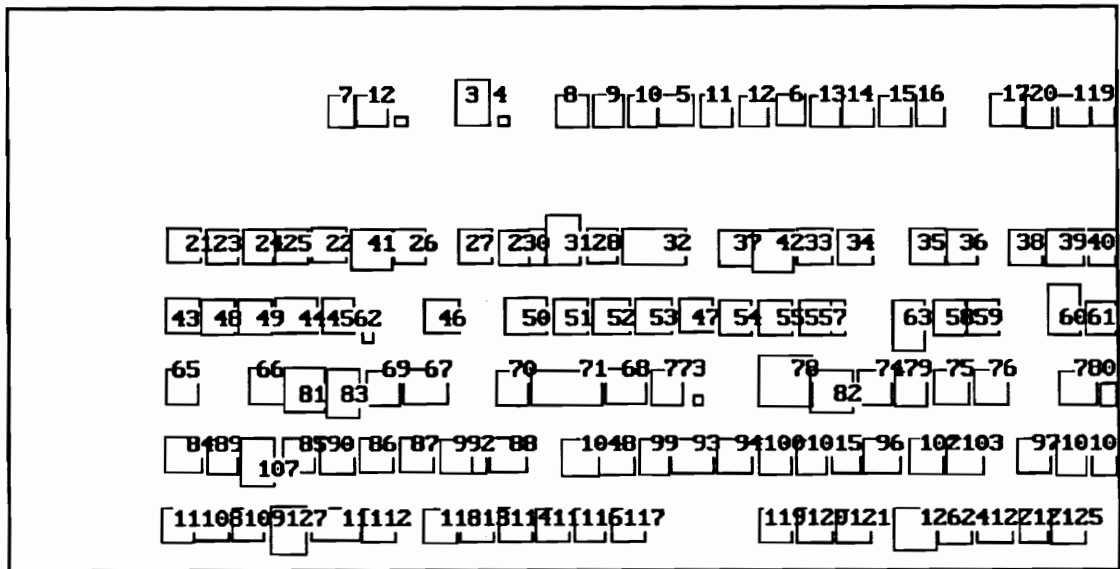


FIGURE 5.5 Order of segment detection in sample image of Fig 5.4.

ГЛ- 5- КОСМИЧЕСКОЕ ☺ЗЛУ  
 частица ВЪБЪЕТ ОДИН ИЗ ☺ЛЕ  
 атома, в Миллионь | раз 60  
 с ядром атома- однако Н<-  
 ПЕРВИЧНЫХ КОСМОЧЕСКИХ ЧАС  
 СТВУЮТ ОТНОСИ ЬНО ДОЛГО

FIGURE 5.6 Reformatted text of sample image in Fig 5.4.

### 5.2.2 Implemented Algorithm

As each new segment is completed, it is allocated to the current active line of text and the metric dimensions of the line are updated until a segment is detected which is "considerably" lower than the previous segments. Then it is assumed that the current line has been completed and a new text line starts; the segments in the line are sorted from left to right and the appropriate spaces are inserted.

## PAGE FORMATTING PROCESS

---

process buffer N

**BEGIN**

**IF (buffer N belongs to in\_line L)**

**add buffer N to line L.**

**ELSE**

**format text line L.**

**increment Line Number L by one.**

**reset data in line L.**

**add buffer N to line L.**

**END.**

---

buffer N belongs to Line L

**BEGIN**

**IF (first segment in line)**

**return TRUE.**

**ELSE IF (BUFFER[N].Ymax less than LineYmax + AveHeight/2)**

**return TRUE.**

**ELSE IF (Buffer[N].Ymin less than LineYmax)**

**return TRUE;**

**ELSE**

**return FALSE;**

**END.**

---

**add buffer N to line L**

**BEGIN**

**set CharHeight equal to BUFFER[N].Ymax - BUFFER[N].Ymin.**

**set CharWidth equal to BUFFER[N].Xmax - BUFFER[N].Xmin;**

**IF (the size of the segment is too big)**

**ignore BUFFER[N].**

**IF (first segment in line)**

**set LineYmin equal to BUFFER[N].Ymin.**

**set LineYmax equal to BUFFER[N].Ymin + AveHeight.**

**IF (LineYmax less than BUFFER[N].Ymax)**

**set LineYmax equal to BUFFER[N].Ymax.**

**set LineHeight equal to LineYmax-LineYmin.**

**set LineWidth equal to average of AveWidth and CharWidth.**

**ELSE**

**IF (LineIndex less than a constant K )**

**set Weight equal to LineIndex.**

**ELSE**

**set Weight equal to K.**

**set LineYmin to weighted average of LineYmin and BUFFER[N].Ymin.**

**set LineYmax to weighted average of LineYmax and BUFFER[N].Ymax.**

**set LineHeight to weighted average of old LineHeight and CharHeight.**

**set LineWidth to the weighted average of old LineWidth and CharWidth.**

**update AveWidth.**

**update AveHeight.**

**END.**

---

**format text line L**

**BEGIN**

**sort segments form left to right.**

**find word breaks.**

**END**

---

**find word breaks**

**BEGIN**

**IF (first segment in line)**

**return FALSE.**

**ELSE IF (distance between adjacent segments grater than LineWidth/2)**

**return TRUE.**

**ELSE**

**return FALSE.**

**END.**

### 5.2.3 Comments

One useful result of the segmentation and page reformatting process is that text can be separated from graphics in a very simple manner. Since graphic objects do not in general conform with printed text lines, they will be isolated by the reformatting procedure. In Fig. 5.7 we present an example of how text embedded within graphics is extracted and properly reformatted.

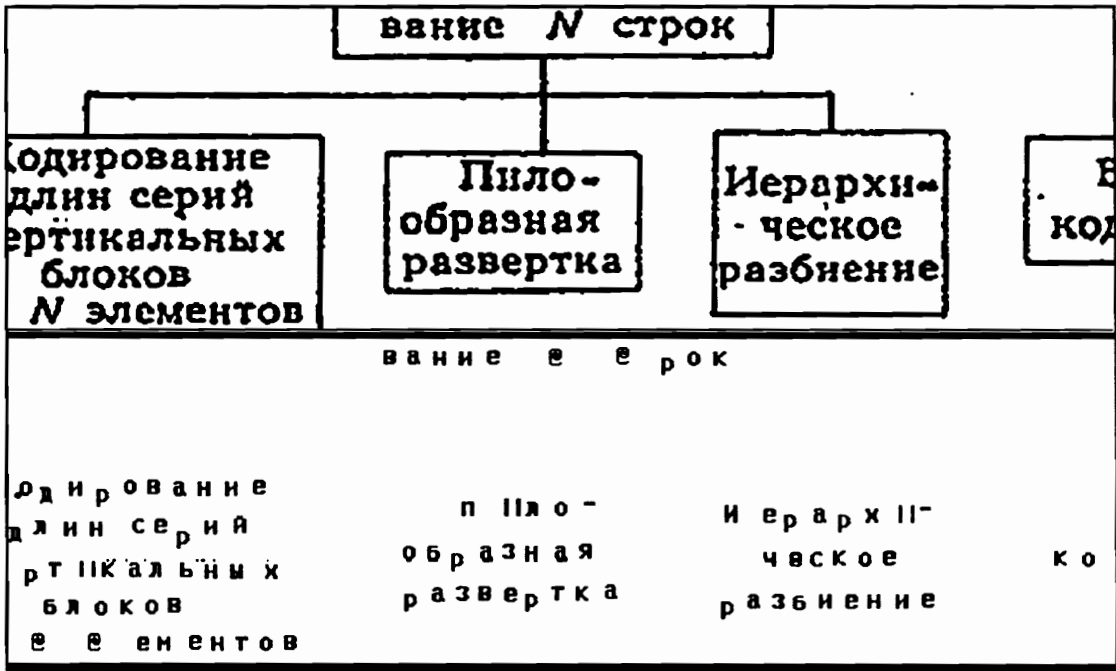


FIGURE 5.7 Text and graphics separation example.

## 6. RECOGNITION LOGIC

In Chapter 4, where we discussed the segmentation algorithm, we mentioned that the QTC sequences and their coordinates serve also as the segment describing features used by the recognition logic. In Chapter 2, we briefly presented the three steps of a hierarchical recognition process, the first step being a structural analysis of the QTC sequences, the second a statistical (or metric) analysis of their relative positions in the characters and the third step being a contextual analysis of segments. Here we discuss in detail these three recognition steps and their implementation in the developed system.

### 6.1 QTCs in Pattern Recognition

The analysis of Quasi-Topological Codes in pattern recognition was introduced several years ago, but was abandoned and never established itself as a good recognition process. The basic reason of that abandonment was the fact that the mere presence of a QTC provides only topological information about a segment but no associated metric information which is very important in optical character recognition. As we discussed in Chapter 2, an uppercase sans-serif D and a sans-serif P or an O can be described by exactly the same sequence of Quasi-Topological Codes and the only differences between these characters are the actual positions of the codes. Nadler[55,57] proposed an improved recognition method that utilized metric information, obtained during the segmentation process, to distinguish among characters with identical topology. The auxiliary codes E and S that we presented earlier were used as a measure of distance between basic QTCs. According to that scheme a 'D' could be represented by the following string of codes:

**D<sub>v</sub>,ES,ES,...,ES,O<sub>v</sub>,ES-ES,Oh,ES-ES,Ch,...,ES-ES,C<sub>v</sub>,ES,...,ES,F<sub>v</sub>.**

The number of ES pairs provided a measure of the distance between the basic codes.

That recognition method was tested with a limited alphabet and proved to be very accurate. One negative point was that representations of the same character could have different number of ES pairs between basic QTCs (due to noise or because of the actual size of the character or scanning resolution). It was, therefore, necessary to generalize the recognition logic to accept such "similar" sequences. The presence of ES pairs in the recognition logic required the construction of huge tables with thousands of states that had to be continuously generalized in order to accept all possible variations of similar characters without explicitly learning them.

In the specific case of manual scan, when the scanning speed is not constant, the problem is compounded. Even with a very good velocity compensation method, characters will be stretched or shrunk in a random manner. As a consequence, the possible variations of a specific character can be practically infinite.

The same property that led to the abandonment of the analysis of QTCs as a reliable recognition scheme (segments with identical topology but different metric properties have identical sequences of QTCs) can be of great help in the case of manual scanning and, as we will see, in omnifont recognition. If we use only the basic QTCs in the two directions (as we discussed in Chapter 4), the number of similar sequences corresponding to the same character is significantly smaller and, assuming small skews, the differences are mainly permutations of horizontal codes. (Because of the systematic scan, codes along the main scan direction tend to appear in the same order, while codes along the sub-scan direction can appear in different places in the QTC string, an observation that led to the generalization techniques to be discussed in Chapter 7).

In Fig 6.1 we can see examples of stretched and slightly rotated instances of the character 'a'. Next to each bitmap we show the QTC sequence that corresponds to it, while in Fig 6.2 we see similar examples of the character 'd'. The QTC sequences for the two sets of characters are very similar, with no reliable structural difference; in particular, the second 'a' and the second 'd' have identical QTC sequences. In Fig 6.3 we can see some examples of a lower case 'e'. Comparing the QTCs sequences corresponding to the characters 'a' and 'd' to those

corresponding to the character 'e', we can see that there is a significant difference both in the QTCs that are present and in their order of appearance. As we discussed in Chapter 4, we store a set of coordinates x-, y- and p- with every QTC. This provides metric information about the specific code. In Fig 6.4 we show the same characters 'a' , 'd' and 'e', but instead of printing the sequence of codes, we use a more graphic representation: every code is displayed as a line from the p-coordinate to the x or y coordinate. Now we can look at the representation of the characters and distinguish between an 'a' and a 'd'. In the particular case of the 'a' and 'd' with identical sequences of QTCs, the position of the upper Dv code can be used as a distinguishing feature.

The above examples can be generalized to a formal hierarchical recognition process, which includes a structural step, a statistical step and a final contextual step. In a first step, the QTC sequence is analyzed and the specific segment is classified to a class of similar characters (which we will call a **confusion class**), while, in a second step metric information is utilized to distinguish among characters of the same confusion class. Any remaining ambiguities are resolved in the final third step, which uses a combination of metric and contextual information to identify a character. In the next sections we will present an attributed grammar which is the basis of such a recognition process. We will also discuss the implementation of that grammar as a finite state automaton and a set of automated metric tests.

a	Dv Ov Oh Dh Ch Cv Ch Xv Dv Dh Ch Fh Dh Fv
a	Dv Ov Dh Oh Ch Cv Xv Dv Fh Ch Dh Fv
a	Dv Ov Dh Oh Ch Ch Cv Xv Dv Dh Ch Fh Dh Fv
a	Dv Ov Dh Oh Ch Cv Xv Dv Dh Ch Fh Ch Dh Fv

FIGURE 6.1. QTCS sequences for sample images of character 'a'.

d	Dv Ov Oh Ch Dh Fh Ch Oh Cv Xv Dv Fh Dh Fv
d	Dv Ov Dh Oh Ch Cv Xv Dv Fh Ch Dh Fv
d	Dv Ov Xv Dv Dh Oh Ch Cv Fh Ch Dh Fv
d	Dv Ov Dh Fh Ch Oh Xv Dv Oh Cv Ch Fh Dh Fv

FIGURE 6.2. QTCS sequences for sample images of character 'd'.

e	Dv Ov Ov Oh Ch Cv Fh Oh Dh Fv Fh Fv
e	Dv Ov Ov Oh Ch Dh Cv Fh Oh Fv Fh Fv
e	Dv Ov Ov Oh Ch Fh Dh Oh Cv Fv Fh Fv
⓪	Dv Ov Ov Ch Oh Fh Dh Oh Cv Fv Fh Fv

FIGURE 6.3. QTC sequences for sample images of character 'e'.

a		d		e	
a		d		e	
a		d		⓪	
a		<b>d</b>		e	

FIGURE 6.4. Graphic representation of QTCs showing their position in the segment.

## 6.2 Grammar Description

Formal languages [6,7,8] are artificial languages that are restricted in such a way that they can be described in terms of a finite set of elements (an alphabet) and a formal way of associating these elements. For example, any computer language is a formal language. These formal languages are usually described by the corresponding **formal grammars**, which are defined as follows:

A formal grammar is, in general, a four-tuple

$$G = \langle V_T, V_N, P, S \rangle,$$

where:

$V_N$  is a set of variables or **non-terminals**.

$V_T$  is a set of **terminals** or **primitives** (alphabet).

$P$  is a set of **productions**, which consists of rules of the form  $a \rightarrow b$  that describe how to generate valid members of the language.

$S$  is a particular non-terminal that denotes the start of a string.

A formal language  $L(G)$  generated by such a grammar  $G$  is, then, defined as a set of strings that consist only of terminals in  $V_T$ , and every string in  $L(G)$  can be derived from the non-terminal  $S$  by applying productions from  $P$ .

A special class of formal grammars is the **regular** or **finite state grammars**, for which the set of productions is very limited: If  $A$  and  $B$  are non-terminals and  $b$  is a terminal, then the only valid production rules are

$$A \rightarrow bB \quad \text{or} \quad A \rightarrow b.$$

Any regular grammar generates a regular language which can be accepted by a **finite state automaton**. The finite state automaton that accepts such a regular grammar  $G$  can be generated as follows:

1. For every rule  $A \rightarrow bB$  in the grammar we create a state  $A$ , a state  $B$  and a link from state  $A$  to  $B$  with input  $b$ .

2. For every rule  $A \rightarrow b$  in the grammar we create a final state  $F$  and a link from  $A$  to  $F$  with input  $b$ .
3. All the other possible links go to an error state.

Another class of formal grammars, which was developed primarily for pattern recognition applications, is **attributed grammars**. In an attributed grammar every element in the set of terminals and possibly non-terminals is associated with a finite set of **attributes**  $A$ . The set of production rules  $P$  consists of two subsets of rules: a set  $P_{\text{syn}}$  of syntactic rules, as in ordinary grammars, and a set  $P_{\text{sem}}$  of semantic rules, which are functions of the attributes. The set of attributes can, for example, be a set of numeric values related to each primitive, and the semantic productions can be arithmetic measurements of these numeric values. If the set of syntactic rules is equivalent to that of a regular grammar, then the attributed grammar is a **regular attributed grammar**.

A grammar that accepts the sequences of QTCs associated with characters and analyzes the position of each QTC within the character, is the regular attributed grammar that we define next:

- A. The set of primitives is the set of basic Quasi-topological codes

$D_v, D_h, F_v, F_h, O_v, O_h, X_v, C_v, C_h$

where each primitive is tagged with three metric attributes, namely the three coordinates  $x$ -,  $y$ - and  $p$ -, as they have already been defined. An additional primitive is used which explicitly denotes the end of a string. The use of a terminating symbol is not necessary, but it simplifies the implementation of the grammar as a finite state automaton.

- B. Only one syntactic rule is allowed in the grammar: right concatenation. A string can be generated by concatenating new primitives at the end of a substring. This concatenation corresponds to the sequential detection of codes as a segment is systematically scanned from left to right (handheld scanner) or top to bottom (page scanner). Actually, two separate grammars are required to recognize characters scanned from left to right and from top to bottom, but as the

idea is the same we will consider only one grammar at this point. Later we will discuss the two grammars that were developed to handle the two cases.

A basic restriction on the valid strings of QTCs is that they have to correspond to connected segments and, as we discussed when we presented the segmentation algorithm, the QTCs of any connected segment must satisfy the following equations:

$$\#Dv + \#Ov = \#Xv + \#Cv + \#Fv,$$

$$\#Dh + \#Oh = \#Ch + \#Fh.$$

Another restriction is that any valid segment must start with Dv or Dh and end with Fv or Fh. Other restrictions exist that require certain codes to appear before others (for example, Cv cannot appear before Ov is present and Xv cannot be detected before at least two Dv have been associated with the segment), but fortunately the grammar does not have to explicitly test for any of the above restrictions. The segmentation process we discussed earlier tests the validity of each QTC sequence. Therefore, it can be assumed that any sequence of QTCs sent to the recognition logic will be a valid topological sequence; the only task of the recognizer is to test whether a certain sequence of QTCs corresponds to a known character or not.

C. The set of semantic or statistical rules is a set of metric tests on the normalized coordinates associated with every QTC.

All the rules that describe valid members of the grammar are inferred from a finite set of prototypes. Through an iterative process, new prototypes are acquired and new constraints or generalizations are added to the initial set of rules until a desired recognition level is achieved. Both structural and statistical rules are added to the grammar through the iterative process. The structural rules are the first stage of recognition (preclassification) and the statistical are metric tests on the attributes associated with each primitive that resolve possible ambiguities left from the first stage.

In the next chapter we will present the results of such iterative learning processes, using a set of Cyrillic characters and a set of Roman characters. In the rest of this chapter we shall assume that we have generated an acceptable grammar and we will discuss the steps that are performed when a segment is presented to the recognizer.

## **6.3 Preclassification**

### **6.3.1 Details**

We already mentioned that the codes Dh and Fh do not add any topological information about a segment. Their presence can be inferred from the number of openings and closings associated with that segment. For example, if a character has only one opening along the subscan direction, we can determine that it has one Dh and two Fh codes.

As we saw in the previous section, the position of the Dh and Fh codes within the QTC sequence can change with small variations of the bit map but their presence and their number can be completely specified from the other codes. If their position in the sequence could be reliably used to distinguish among similar characters, then they should be included in the preclassification analysis, otherwise they could be ignored in that first step. Our initial observations showed that the position of the Dh and Fh codes was not adding any reliable structural information and, in an effort to simplify the preclassification stage, we decided to analyze their effect on the number of confusion classes and on the size of the preclassification tables.

We used a large sample of characters and generated two preclassification tables, one that included the Dh and Fh codes and one that ignored them (the final error rate was the same in both cases). The percentage of characters resulting to confusion classes was approximately the same in both cases but the actual table size was significantly smaller when the Dh and Fh codes were not included (see Fig. 6.5). As the experiment verified our initial observations, we decided to exclude Dh and Fh from the preclassification analysis and use the information that they carried only in the subsequent metric tests.

<b>Dh, Fh</b>	<b>ERROR RATE</b>	<b>CONFUSION CLASS PERCENTAGE</b>	<b>TABLE SIZE</b>
<b>YES</b>	<b>12%</b>	<b>45%</b>	<b>31,000</b>
<b>NO</b>	<b>12%</b>	<b>48%</b>	<b>5,000</b>

**FIGURE 6.5.** Analysis of structural information contributed by Dh and Fh.

**6.3.2 State automaton implementation**

As we said in section 6.2, any regular grammar can be implemented as a finite state automaton (or state machine) [7]. The language primitives are the inputs to the automaton and every state in the state machine corresponds to a valid substring accepted by the grammar. When a sequence of primitives is sent to the finite state automaton, the state of the latter changes with every new primitive; as long as the automaton can go to a new next state, the sequence is considered valid. If, on the other hand, a next state is not defined for some primitive, then the sequence is rejected as invalid.

The linear grammar that we described in 6.2 was implemented in a finite state automaton, which was automatically built during the learning process, as we will discuss later. For practical purposes the length of the accepted strings is limited to 32. Longer strings would require more memory, and in general do not correspond to valid characters. When the length of the QTC string is greater than 32 codes, the corresponding segment is considered to be graphics and is labeled accordingly.

The implemented finite state automaton (which we call preclassification table) is a Moore machine. It accepts sequences of QTCs as they become available from the segmenter and classifies them accordingly (as we said, it ignores codes Dh and Fh). If the sequence is unique, then the automaton outputs the code corresponding to that specific sequence. If the sequence

corresponds to more than one character, the output of the automaton is a pointer to a confusion class that contains all the similar characters; a set of metric tests is then employed to identify the segment. Finally, if the sequence is not valid, the automaton responds with a special rejection code.

Figure 6.6 shows the preclassification table that accepts the sequences of QTCs corresponding to the twelve characters shown in Figs. 6.1, 6.2 and 6.3. All sequences are assumed to end with a special NULL code, which causes the preclassification table to generate the appropriate output. In this particular example, the first column of the automaton contains all the output codes, while all the other columns contain intermediate next states, with -1 denoting an undefined next state. If a sequence is unique, the output corresponds to the ASCII code of the sample character (97 for 'a', 100 for 'd' and 101 for 'e'), while confusion classes are labeled with numbers larger than 256. In our example state 27 is the final state corresponding to the similar 'a' and 'd' and the output code 256 is a pointer to a confusion class table that contains the members of each confusion class and the representative sequences of QTCs, as shown in Table 6.2.

**TABLE 6.1. Preclassification Table for Characters in fig 6.4.**

$\phi$	Dv	Ov	Cv	Fv	Dh	Fh	Oh	Ch	Xv	
-1	1	-1	-1	-1	-1	-1	-1	-1	-1	0
-1	-1	-1	2	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	17	-1	-1	-1	3	44	33	2
-1	-1	-1	-1	-1	-1	-1	-1	4	-1	3
-1	-1	-1	-1	11	-1	-1	-1	5	-1	4
-1	-1	-1	-1	28	-1	-1	6	-1	-1	5
-1	-1	-1	-1	7	-1	-1	-1	-1	-1	6
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	7
-1	9	-1	-1	-1	-1	-1	-1	-1	-1	8
-1	-1	10	-1	-1	-1	-1	-1	-1	-1	9
100	-1	-1	-1	-1	-1	-1	-1	-1	-1	10
-1	-1	-1	-1	-1	-1	-1	-1	12	24	11
-1	-1	-1	-1	-1	-1	-1	-1	-1	13	12
-1	14	-1	-1	-1	-1	-1	-1	-1	-1	13
-1	-1	-1	-1	-1	-1	-1	-1	15	-1	14
-1	-1	16	-1	-1	-1	-1	-1	-1	-1	15
97	-1	-1	-1	-1	-1	-1	-1	-1	-1	16
-1	-1	-1	-1	-1	-1	-1	18	-1	-1	17
-1	-1	-1	-1	-1	-1	-1	-1	19	-1	18
-1	-1	-1	-1	20	-1	-1	40	-1	-1	19
-1	-1	-1	-1	-1	-1	-1	21	-1	-1	20
-1	-1	22	-1	-1	-1	-1	-1	-1	-1	21
-1	-1	23	-1	-1	-1	-1	-1	-1	-1	22
101	-1	-1	-1	-1	-1	-1	-1	-1	-1	23
-1	25	-1	-1	-1	-1	-1	-1	-1	-1	24
-1	-1	-1	-1	-1	-1	-1	-1	26	-1	25
-1	-1	27	-1	-1	-1	-1	-1	52	-1	26
256	-1	-1	-1	-1	-1	-1	-1	-1	-1	27
-1	-1	-1	-1	-1	-1	-1	-1	-1	29	28
-1	30	-1	-1	-1	-1	-1	-1	-1	-1	29
-1	-1	-1	-1	-1	-1	-1	-1	31	-1	30
-1	-1	32	-1	-1	-1	-1	-1	-1	-1	31
97	-1	-1	-1	-1	-1	-1	-1	-1	-1	32
-1	34	-1	-1	-1	-1	-1	-1	-1	-1	33
-1	-1	-1	-1	-1	-1	-1	-1	35	-1	34
-1	-1	-1	-1	-1	-1	-1	-1	36	-1	35
-1	-1	-1	-1	37	-1	-1	-1	-1	-1	36
-1	-1	-1	-1	-1	-1	-1	-1	38	-1	37
-1	-1	39	-1	-1	-1	-1	-1	-1	-1	38
100	-1	-1	-1	-1	-1	-1	-1	-1	-1	39
-1	-1	-1	-1	41	-1	-1	-1	-1	-1	40
-1	-1	42	-1	-1	-1	-1	-1	-1	-1	41
-1	-1	43	-1	-1	-1	-1	-1	-1	-1	42
101	-1	-1	-1	-1	-1	-1	-1	-1	-1	43
-1	-1	-1	-1	-1	-1	-1	45	-1	-1	44
-1	-1	-1	-1	-1	-1	-1	-1	-1	46	45
-1	47	-1	-1	-1	-1	-1	-1	-1	-1	46
-1	-1	-1	-1	-1	-1	-1	48	-1	-1	47
-1	-1	-1	-1	49	-1	-1	-1	-1	-1	48
-1	-1	-1	-1	-1	-1	-1	-1	50	-1	49
-1	-1	51	-1	-1	-1	-1	-1	-1	-1	50
100	-1	-1	-1	-1	-1	-1	-1	-1	-1	51
-1	-1	53	-1	-1	-1	-1	-1	-1	-1	52
97	-1	-1	-1	-1	-1	-1	-1	-1	-1	53

**TABLE 6.2.** Confusion class Table for sample images in fig 6.4.

CLASS		MEMBERS		QTC SEQUENCE
256,	97	100	0	Dv Ov Dh Oh Ch Cv Xv Dv Fh Ch Dh Fv
257,	0	0	0	$\phi$
258,	0	0	0	$\phi$
259,	0	0	0	$\phi$
260,	0	0	0	$\phi$

### 6.3.3 Comments

Most of the data structures were implemented as fixed arrays rather than linked lists of data. Although dynamic linked lists could have saved us considerable amount of memory, we chose the fixed arrays because they enable easy and fast random access of the data. Since approximately 50 percent of the time some metric test is required (as is shown in Fig. 6.5), we chose speed versus memory in our implementation. In the case of the confusion class table we stored some redundant information, such as a representative sequence, because it helped to systematize the learning process. As we will see later, a sequence of QTCs may be accepted as a valid sequence even if it has not explicitly been added to the preclassification table. In order to avoid the generation of faulty confusion classes during learning, we required that any character entering a confusion class must have at least the same codes as all the other members of the class. The representative sequence provided a prototype to compare with; in addition, it provided easily accessible information about the general structure of the characters in the confusion class, something that was very useful in the process of designing appropriate metric tests.

## 6.4 Metric Tests

The second step of the recognition process analyzes the coordinates of the QTCs within a character in order to separate characters that belong to the same confusion class. The metric tests we used were designed in such a way that they would be as immune to stretching and small rotation as possible. In order to achieve this goal, we selected tests that did not depend on relative dimensions along both the main scan direction and the subscan direction.

One important consideration in designing the metric tests that would distinguish among characters belonging to a confusion class was that they had to be automatically generated and updated. As our goal was to achieve error rates of the order of  $10^3$  to  $10^4$ , it was not reasonable to manually design the metric tests. Instead, we had to find an efficient way of selecting the metric differences that would reliably separate the members of a confusion class.

One approach would be to compute a difference matrix, where we store the normalized distances between each QTC in a segment, and select as reliable tests the distances that differ the most from segment to segment. However, as we mentioned earlier, the overall size of a character can change in a manual scan and therefore diagonal distances would also change. (If we assume that the character is uniformly stretched, then the normalized distances remain unchanged. Experiments showed, however, that bitmaps of manually scanned characters did not appear to be uniformly stretched). One modification we considered was to create three such matrices, one for each of the  $x$ ,  $y$  and  $p$  coordinates associated with a QTC. Unfortunately, the complexity of such a solution seems prohibitive. Another reason that made us look for other solutions was that there was no intuitive connection between this type of tests and the structure of the character.

Believing in structural methods more than in statistical ones, we decided to tie the metric tests to actual structural features of the characters. From the representative sequence of QTCs stored with each confusion class, we identified a list of features present in all members of the class. The features were simply structural properties of the characters, such as number of

openings, number of closings, etc. The complete list of features we decided to use is shown below in Table 6.3.

**TABLE 6.3** List of features used in metric tests.

1. Single Line	9. One Cv	17. Two Oh	25. Three Ov
2. Single Loop	10. One Ch	18. Two Cv	26. Three Oh
3. One Dv	11. One Xv	19. Two Ch	27. Three Cv
4. One Dh	12. Two Dv	20. Two Xv	28. Three Ch
5. One Fv	13. Two Dh	21. Three Dv	29. Three Xv
6. One Fh	14. Two Fv	22. Three Dh	30. Xv,Ov
7. One Ov	15. Two Fh	23. Three Fv	31. Oh, Ch
8. One Oh	16. Two Ov	24. Three Fh	32. -

The above list of features is organized in such a way that more robust tests are considered first, while more sensitive tests are used only if necessary. The tests are performed sequentially, until a unique decision is made, or until all the tests are exhausted and no match is found, or until, at the end of all tests, more than one prototype matches the unknown character. It can be seen that most of the features are mutually exclusive. For example, if a certain segment has one vertical start, then features 12 and 21 cannot be present. The maximum number of simultaneously present features is ten.

With each present feature, we associate a number of metric tests. They measure the relative positions of the present QTCs and/or their normalized position within the circumscribed rectangle. We shall list the exact tests later, but now we have to mention that some ideas of fuzzy logic [59,60] had to be employed in order to handle the wide distribution of bitmaps associated with an ASCII code. Referring to Fig. 6.4, we can see that, in order to distinguish between the 'a' and 'd' with identical QTC sequences, we have to check if the horizontal coordinate of the upper Dv is towards the left of the character or towards the right. How far to the left or how far to the right is not well defined. In an initial attempt to implement some stochastic process, we tried the following scheme. We predefined a range of values for a certain measurement and tested it with a number of characters. For example, if feature 3 (one Dv) was

present, we tested if the x-coordinate of Dv was in the upper third of the character, or the middle third, or the lower third; if, for all the tested characters with the same ASCII, the x-coordinate was consistently at the same third, while for all other characters the same x-coordinate was at a different third, we could assume that this measurement was a reliable test and could be used to separate that specific ASCII character from the rest of the characters in the confusion class. If, on the other hand, the x-coordinate of the QTC fell in more than one of these predefined areas, we assumed that it was not a reliable measurement. Each test could take one of three values 0,1,X. 0 denoted that a measurement was always outside a predefined range, 1 that the measurement was consistently inside the range, and X that some measurements were inside the range and some outside. The test in the last case was not reliable and could be ignored.

A typical set of tests corresponding to the 'a' and 'd' of Fig. 6.4, is shown in Table 6.4. The features are listed at the left of the tests, while each test corresponds to a predefined measurement on those features. We will not discuss the measurements we selected for each present feature because they are very similar to those we selected in the final version, which we discuss next. It suffices to say that the X's shown in this example were obtained by testing a large number of similar 'a' and 'd'. It can be seen that the only consistently different measurements are the ones related to the two Dv codes.

**TABLE 6.4** Metric tests distinguishing an 'a' from a 'd'.

	a				d			
One Fv	1	0	0	0	1	0	0	0
One Fh	X	X	0	X	X	X	0	X
One Ov	X	X	0	X	X	X	0	X
One Oh	0	X	X	X	0	X	X	X
One Cv	X	0	0	0	X	0	0	X
One Xv	0	X	X	0	0	X	X	X
Two Dv	1	1	0	0	1	0	1	X
Two Dh	X	X	0	0	X	X	0	0
Two Ch	1	1	0	0	1	1	0	X

The major disadvantage of the above approach was that we had to predefine the range of each measurement. If the range was too small, then, during an iterative learning process, a large number of characters was rejected, while, if the range was too wide, a large number of confusions remained unresolved. If the dominant errors were rejections, we could learn the characters that failed, expecting that the tests that caused the false rejections would change to don't care (X) without causing any additional ambiguities. If, on the other hand, the dominant errors were substitutions or unresolved ambiguities, then the addition of new characters could not correct the test results. In that case we had to manually intervene and change the range of the tests so that the ambiguities could be resolved. As we mentioned, however, such a learning process was not practical because of the large set of characters we had to process. We needed to find a better way to determine the range of each test.

In the final version of metric test design, we used the same structural features that we show in Table 6.3, and with each feature we associated the set of tests we present in Table 6.5. Instead of predefining the ranges though, we let the set of characters specify the exact range of each measurement. We define a minimum and maximum value for each measurement, and we update them with every new prototype that we add to the design test. If a measurement falls within the current range, no change is made; otherwise, the minimum or maximum value is changed to include the new measurement. In order to save memory space and simplify the computations, we decided to quantize the measurement in increments of 1/16 of the length or the width of a character. In Table 6.6 we show the tests that correspond to the 'a' and 'd' of Fig. 6.4. The left number in each test corresponds to the minimum acceptable value and the right to the maximum acceptable value.

This approach generates more accurate metric tests and tends to generate mainly rejection errors rather than substitutions. As we will discuss in chapter 7, minimal manual intervention was required, even when very low error rates were achieved.

**TABLE 6.5a.** List of metric tests associated with features present in a segment.

**SINGLE LOOP**

- |                                      |                                       |
|--------------------------------------|---------------------------------------|
| 1. Lower Loop ( $y_{Ov}/YSize$ )     | 2. Upper Loop ( $y_{Cv}/YSize$ )      |
| 3. Sharp left side( $p_{Dv}/XSize$ ) | 4. Sharp right side( $p_{Fv}/XSize$ ) |
| 5. Left Stroke ( $x_{Dv}/XSize$ )    | 6. Right stroke ( $x_{Fv}/XSize$ )    |

**ONE Dv**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Dv}/XSize$ ) | 2. Y coordinate ( $y_{Dv}/YSize$ )   |
| 3. P coordinate ( $p_{Dv}/XSize$ ) | 4. Width ( $(x_{Dv}-p_{Dv})/XSize$ ) |

**ONE Dh**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Dh}/XSize$ ) | 2. Y coordinate ( $y_{Dh}/YSize$ )   |
| 3. P coordinate ( $p_{Dh}/YSize$ ) | 4. Width ( $(x_{Dh}-p_{Dh})/YSize$ ) |

**ONE Fv**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Fv}/XSize$ ) | 2. Y coordinate ( $F_{Dv}/YSize$ )   |
| 3. P coordinate ( $p_{Fv}/XSize$ ) | 4. Width ( $(x_{Fv}-p_{Fv})/XSize$ ) |

**ONE Fh**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Fh}/XSize$ ) | 2. Y coordinate ( $y_{Fh}/YSize$ )   |
| 3. P coordinate ( $p_{Fh}/YSize$ ) | 4. Width ( $(x_{Fh}-p_{Fh})/YSize$ ) |

**ONE Ov**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Ov}/XSize$ ) | 2. Y coordinate ( $y_{Ov}/YSize$ )   |
| 3. P coordinate ( $p_{Ov}/XSize$ ) | 4. Width ( $(x_{Ov}-p_{Ov})/XSize$ ) |

**ONE Oh**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Oh}/XSize$ ) | 2. Y coordinate ( $y_{Oh}/YSize$ )   |
| 3. P coordinate ( $p_{Oh}/YSize$ ) | 4. Width ( $(x_{Oh}-p_{Oh})/YSize$ ) |

**ONE Cv**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Cv}/XSize$ ) | 2. Y coordinate ( $y_{Cv}/YSize$ )   |
| 3. P coordinate ( $p_{Cv}/XSize$ ) | 4. Width ( $(x_{Cv}-p_{Cv})/XSize$ ) |

**ONE Ch**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Ch}/XSize$ ) | 2. Y coordinate ( $y_{Ch}/YSize$ )   |
| 3. P coordinate ( $p_{Ch}/YSize$ ) | 4. Width ( $(x_{Ch}-p_{Ch})/YSize$ ) |

**ONE Xv**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| 1. X coordinate ( $x_{Xv}/XSize$ ) | 2. Y coordinate ( $y_{Xv}/YSize$ )   |
| 3. P coordinate ( $p_{Xv}/XSize$ ) | 4. Width ( $(x_{Xv}-p_{Xv})/XSize$ ) |

**TABLE 6.5b.** List of metric tests associated with features present in a segment (continued).

---

<b>TWO Dv (Dv1 upper code, Dv2 lower code).</b>	
1. X coordinate ( $x_{Dv1}/XSize$ )	2. Y coordinate ( $y_{Dv1}/YSize$ )
3. P coordinate ( $p_{Dv1}/XSize$ )	4. Width ( $(x_{Dv1}-p_{Dv1})/XSize$ )
5. X coordinate ( $x_{Dv2}/XSize$ )	6. Y coordinate ( $y_{Dv2}/YSize$ )
7. P coordinate ( $p_{Dv2}/XSize$ )	8. Width ( $(x_{Dv2}-p_{Dv2})/XSize$ )
<b>TWO Dh (Dh1 left code, Dh2 right code)</b>	
1. X coordinate ( $x_{Dh1}/XSize$ )	2. Y coordinate ( $y_{Dh1}/YSize$ )
3. P coordinate ( $p_{Dh1}/YSize$ )	4. Width ( $(y_{Dh1}-p_{Dh1})/YSize$ )
5. X coordinate ( $x_{Dh2}/XSize$ )	6. Y coordinate ( $y_{Dh2}/YSize$ )
7. P coordinate ( $p_{Dh2}/YSize$ )	8. Width ( $(y_{Dh2}-p_{Dh2})/YSize$ )
<b>TWO Fv (Fv1 upper code, Fv2 lower code).</b>	
1. X coordinate ( $x_{Fv1}/XSize$ )	2. Y coordinate ( $y_{Fv1}/YSize$ )
3. P coordinate ( $p_{Fv1}/XSize$ )	4. Width ( $(x_{Fv1}-p_{Fv1})/XSize$ )
5. X coordinate ( $x_{Fv2}/XSize$ )	6. Y coordinate ( $y_{Fv2}/YSize$ )
7. P coordinate ( $p_{Fv2}/XSize$ )	8. Width ( $(x_{Fv2}-p_{Fv2})/XSize$ )
<b>TWO Fh (Fh1 left code, Fh2 right code)</b>	
1. X coordinate ( $x_{Fh1}/XSize$ )	2. Y coordinate ( $y_{Fh1}/YSize$ )
3. P coordinate ( $p_{Fh1}/YSize$ )	4. Width ( $(y_{Fh1}-p_{Fh1})/YSize$ )
5. X coordinate ( $x_{Fh2}/XSize$ )	6. Y coordinate ( $y_{Fh2}/YSize$ )
7. P coordinate ( $p_{Fh2}/YSize$ )	8. Width ( $(y_{Fh2}-p_{Fh2})/YSize$ )
<b>TWO Ov (Ov1 upper code, Ov2 lower code).</b>	
1. X coordinate ( $x_{Ov1}/XSize$ )	2. Y coordinate ( $y_{Ov1}/YSize$ )
3. P coordinate ( $p_{Ov1}/XSize$ )	4. Width ( $(x_{Ov1}-p_{Ov1})/XSize$ )
5. X coordinate ( $x_{Ov2}/XSize$ )	6. Y coordinate ( $y_{Ov2}/YSize$ )
7. P coordinate ( $p_{Ov2}/XSize$ )	8. Width ( $(x_{Ov2}-p_{Ov2})/XSize$ )
<b>TWO Oh (Oh1 left code, Oh2 right code)</b>	
1. X coordinate ( $x_{Oh1}/XSize$ )	2. Y coordinate ( $y_{Oh1}/YSize$ )
3. P coordinate ( $p_{Oh1}/YSize$ )	4. Width ( $(y_{Oh1}-p_{Oh1})/YSize$ )
5. X coordinate ( $x_{Oh2}/XSize$ )	6. Y coordinate ( $y_{Oh2}/YSize$ )
7. P coordinate ( $p_{Oh2}/YSize$ )	8. Width ( $(y_{Oh2}-p_{Oh2})/YSize$ )
<b>TWO Cv (Cv1 upper code, Cv2 lower code).</b>	
1. X coordinate ( $x_{Cv1}/XSize$ )	2. Y coordinate ( $y_{Cv1}/YSize$ )
3. P coordinate ( $p_{Cv1}/XSize$ )	4. Width ( $(x_{Cv1}-p_{Cv1})/XSize$ )
5. X coordinate ( $x_{Cv2}/XSize$ )	6. Y coordinate ( $y_{Cv2}/YSize$ )
7. P coordinate ( $p_{Cv2}/XSize$ )	8. Width ( $(x_{Cv2}-p_{Cv2})/XSize$ )

---

**TABLE 6.5c. List of metric tests (associated with features present in a segment (continued)).**

**TWO Xv (Xv1 upper code, Xv2 lower code).**

- |                                     |  |
|-------------------------------------|--|
| 1. X coordinate ( $x_{Xv1}/XSize$ ) | 2. Y coordinate ( $y_{Xv1}/YSize$ )    |
| 3. P coordinate ( $p_{Xv1}/XSize$ ) | 4. Width ( $(x_{Xv1}-p_{Xv1})/XSize$ ) |
| 5. X coordinate ( $x_{Xv2}/XSize$ ) | 6. Y coordinate ( $y_{Xv2}/YSize$ )    |
| 7. P coordinate ( $p_{Xv2}/XSize$ ) | 8. Width ( $(x_{Xv2}-p_{Xv2})/XSize$ ) |

**THREE Dv**

No tests implemented

**THREE Dh**

No tests implemented

**THREE Fv**

No tests implemented

**THREE Fh**

No tests implemented

**THREE Ov**

No tests implemented

**THREE Oh**

No tests implemented

**THREE Cv**

No tests implemented

**THREE Ch**

No tests implemented

**THREE Xv**

No tests implemented

**ONE Ch ONE Oh**

- |   |   |
|---|---|
| 1. Oh Lower ( $(x_{Ch}-x_{Oh})/XSize$ ) | 2. Ch lower ( $(x_{Oh}-x_{Ch})/XSize$ ) |
| 3. Ch Left of Oh ( $(y_{Oh}-y_{Ch})$ )  | 4. Ch Right of Oh ( $(y_{Ch}-y_{Oh})$ ) |

**ONE XV ONE OV**

- |   |   |
|---|---|
| 1. Ov Lower ( $(x_{Xv}-x_{Ov})/XSize$ ) | 2. Xv lower ( $(x_{Ov}-x_{Xv})/XSize$ ) |
| 3. Xv Left of Ov ( $(y_{Ov}-y_{Xv})$ )  | 4. Xv Right of Ov ( $(y_{Xv}-y_{Ov})$ ) |

**TABLE 6.6 Updated Metric Tests distinguishing 'a' from 'd' (see Fig. 6.4).**

<b>a</b>									
One Fv	F,F	F,F	D,F	0,2	—	—	—	—	—
One Fh	0,0	5,B	1,A	2,7	—	—	—	—	—
One Ov	7,D	1,2	6,B	1,6	—	—	—	—	—
One Oh	D,E	2,B	2,A	1,5	—	—	—	—	—
One Cv	7,D	B,C	7,B	1,4	—	—	—	—	—
One Xv	1,5	B,C	1,4	0,4	—	—	—	—	—
Two Dv	0,3	0,2	0,3	0,2	7,E	0,0	7,D	0,5	—
Two Dh	E,F	1,8	1,8	1,7	E,F	9,F	9,E	0,5	—
Two Ch	7,A	2,9	2,8	1,4	D,E	9,B	9,B	0,3	—
<b>d</b>									
One Fv	F,F	F,F	D,F	0,2	—	—	—	—	—
One Fh	0,0	4,B	1,B	0,8	—	—	—	—	—
One Ov	9,D	1,2	8,A	1,5	—	—	—	—	—
One Oh	D,E	2,B	2,A	1,5	—	—	—	—	—
One Cv	7,D	B,D	7,C	1,4	—	—	—	—	—
One Xv	1,7	B,D	1,6	0,5	—	—	—	—	—
Two Dv	0,3	5,A	0,3	0,2	8,E	0,0	8,D	0,5	—
Two Dh	E,F	1,8	1,8	1,7	E,F	A,F	9,E	0,4	—
Two Ch	7,9	2,9	2,8	1,4	D,E	9,C	9,B	0,3	—

## 6.5 Postprocessing

The final step of recognition uses contextual and metric information to resolve the remaining ambiguities. For example, the first two steps cannot distinguish between a period and the dot above a lower case i, or between a 0(zero) and an O(oh) in some fonts. These ambiguities can be resolved only if the surrounding characters are examined. The purpose of the postprocessing step is to analyze the sequence of ASCII codes associated with a line of text and correct possible errors or resolve existing ambiguities.

In many cases the text to be recognized contains badly shaped characters and or broken characters. In some fonts certain characters tend to break in places where the lines are very thin.

The segmenter will separate the broken pieces and the recognition steps will probably reject the pieces as unknown characters. One task that the postprocessor can perform is to reassemble the segments and reconstruct the original character.

### **6.5.1 Application Specific Postprocessing**

In an effort to test the feasibility of such a process, we used a Cyrillic font, in which certain characters tend to break in two segments, left and right or top and bottom. We explicitly labeled the broken pieces as the left part of character X or the right part of character X, etc. The preclassifier and the metric tests treated these pieces as independent segments. If the broken piece could be identified as part of some character, then the postprocessor examined the adjacent segments to find the other part and merge the two segments. The results of this experiment were satisfactory but the test was limited to family of similar fonts only.

### **6.5.2 Broken Character Reconstruction**

A more sophisticated (and more practical) approach would be to combine the information about the position of each segment and the topological information provided by the QTCs and join the broken pieces so that complete characters are presented to the recognition logic.

Such a reconstruction can be applied as a preprocessor to existing optical character recognition systems or can be used as an image enhancement technique in digital imaging applications. We are currently investigating algorithms that could reliably perform such a reconstruction.

## **7. LEARNING PROCESS**

In this chapter we discuss the procedure we followed to train the recognition logic down to a desired error rate. The learning process includes several steps starting with the generation of a training sample of characters and ending with the customization of the postprocessing logic to fit specific applications. Each step of the learning process is analyzed next.

### **7.1 Labeling Process**

A tedious but very important step of the learning process is the generation of a training set of characters. In order to reliably achieve a "good" error rate, several thousand characters must be acquired and properly labeled. The distribution of characters or other symbols in the training set should be representative of their distribution in the language that is being learned, otherwise the estimation of the error rate will not be realistic.

The procedure that was followed to generate a good training set of characters consisted of the following steps:

**Step 1.** A large number of pages was scanned and stored in a standard compressed form.

The use of actual text as the training set guaranteed realistic estimation of the error rates.

**Step 2.** Each page was then retrieved, decompressed by a software decompression procedure and passed through a segmentation and feature extraction program that saved the features detected at each segment. In addition to the features, the position of the segment in the page was saved so that the stored information could be linked to the actual bit map.

The format with which the information was stored was the same as the format used by

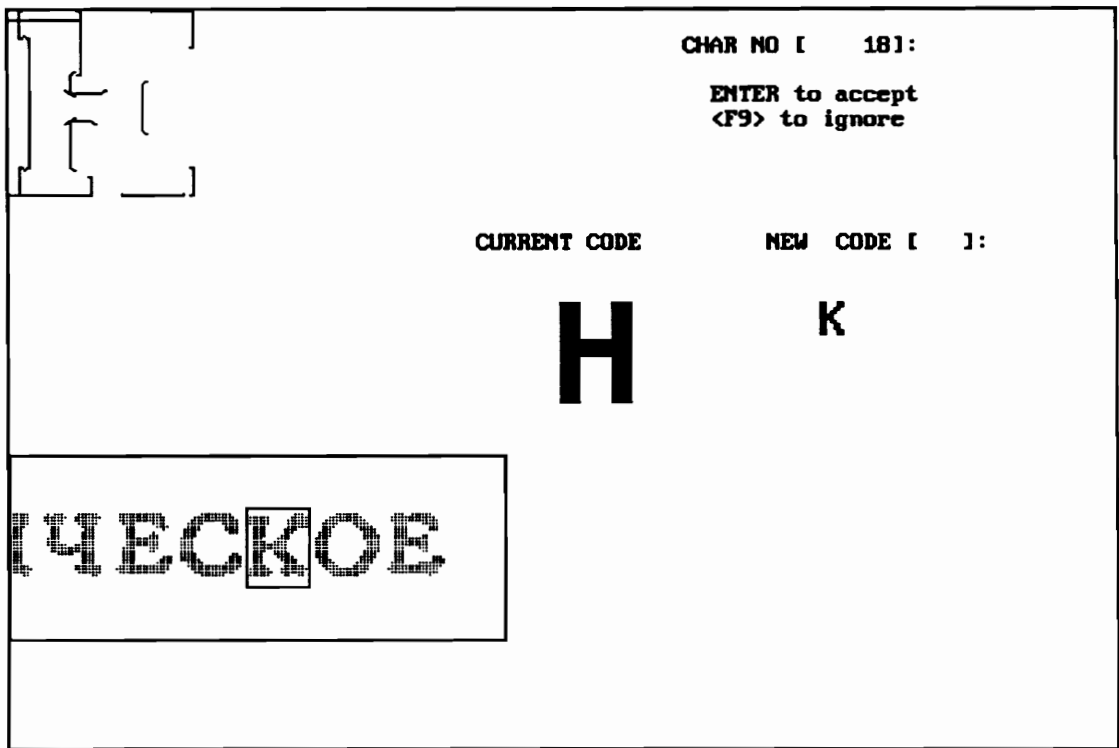
the recognition logic. All the operations of this step were completely automated and required no operator intervention.

Step 3. Each acquired segment was presented to an operator who looked at a graphic representation of the extracted features and the corresponding bit map and inserted the proper code. Figure 7.1 shows a typical screen that was presented to the operator for each segment. At the top left of the screen one can see a graphic representation of the detected features, while the corresponding bit map is displayed at the bottom left. The operator could accept the label that was attached to the segment (shown at the center of the screen as CURRENT CODE), or enter a new code. If the bit map information was not adequate, the operator could view a larger window around the questionable character. Graphical representation of the information improved the operator's efficiency and reduced the number of keying errors. In order to facilitate acquisition of defective segments (broken or connected characters), each segment could be labeled as one character, half of a character or two joined characters. Any segment that did not fit in one of the above categories was rejected.

Step 4. Every page was labeled a second time. Initially, the second labeling was performed by a different operator; however, when the recognition logic was trained to a reasonable error rate (5%), the second labeling of a page was automated and performed by the recognition system. With the automation of the second labeling, the labeling process became more efficient as each page had to be manually labeled only once.

Step 5. The two independently labeled pages were compared segment by segment and the differences were presented to an operator who made a final decision. Figure 7.2 shows a typical screen of the verification process. The operator can select one of the two entries or enter a new label. Statistics kept during the entire labeling process showed that approximately 4 to 5 percent of the characters had different labels and had to be

corrected. It is interesting to mention that in several cases the automatic labeling caught keying errors that could have been otherwise unnoticed (similar looking characters). Another interesting result of the verification process was that it revealed error patterns that helped us detect defects in the recognition logic, such as software bugs or logical errors.



**FIGURE 7.1.** Typical screen for manual character labeling.

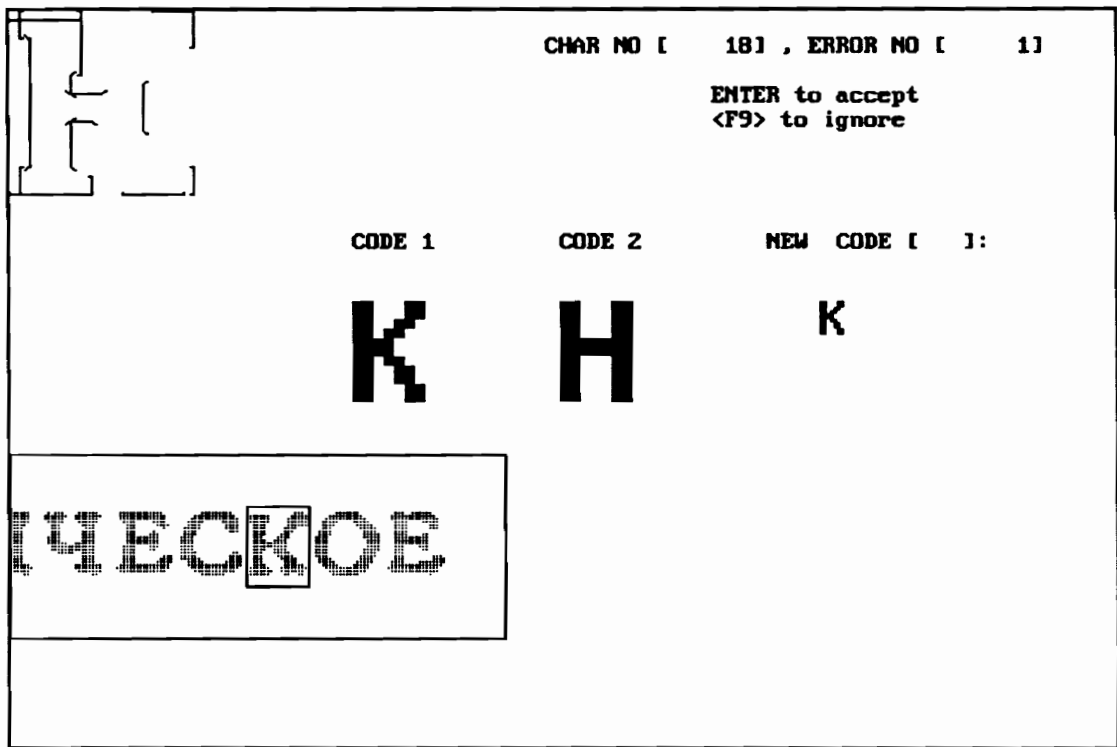


FIGURE 7.2. Typical label verification screen.

## 7.2 Iterative Design of Preclassification Tables

As we discussed earlier, the first step in the recognition process we implemented was to use the sequence of QTCs to identify a character. This preclassification step was implemented as a state automaton which was built with the following iterative process:

Step 1. A set of labeled characters was tested with the current version of the generalized preclassification table until a predefined number of distinct new characters was detected. When a character was not accepted, its QTC sequence was compared with the new QTC sequences that had been acquired up to that point. If another segment with the same label and sequence of QTCs had not yet been encountered, the character was added to the set of new characters. The process was repeated until the predefined number of new characters was acquired.

Step 2. The new characters were added to the current version of the ungeneralized preclassification table. When a new sequence of QTCs was added to the state machine one of three types of changes occurred:

A. If the sequence was unique then for some QTC in the sequence there would be no next state, in which case new states were added at the end of the state machine. When the end of the sequence was reached, a final state was added that output the code corresponding to the new character.

B. If the sequence was not unique and the current output of the final state was another character then a new confusion class had been detected. The output of the final state was changed to the newly created confusion class and the two codes were added to the class.

C. If the sequence was not unique and the current output of the final state was a confusion class then the new character was added to the list of characters that already belonged to the confusion class.

Step 3. The version of the preclassification table that included the newly added characters was generalized to accept additional sequences similar to the ones explicitly learned. The generalization algorithms that were employed are discussed in detail in section 7.3. Step 3 was not absolutely necessary but it increased significantly the learning rate and reduced substantially the size of the preclassification state machine. Erroneous paths that might be generated by generalization were detected in a following iteration and the constraints that were imposed by the new characters prevented those faulty paths from being created again.

Step 4. Steps 1, 2 and 3 were repeated until a desired error rate was achieved.

In section 7.4 we present detailed statistics that were collected during the learning process of a Cyrillic font and several Roman fonts.

### **7.3 Knowledge Generalization**

Several generalization procedures were evaluated. Formal state reduction [61, 62], merging of similar states [54] and other techniques were programmed and tested but they did not provide adequate knowledge generalization.

Formal state reduction introduced a large number of erroneous paths because it did not take under consideration the topological properties of the QTC sequences.

Another generalization method that was tested has been proposed by Nadler in [55]. According to this method, states are merged in a hierarchical order, starting from identical states

and moving to less and less similar states. Similar states are defined to be states that for the same set of inputs call a set of equivalent next states. Tests with this generalization technique showed that the number of erroneous paths was reduced but the generalization was minimal. The reason for the small generalization is that the length of the QTCs sequences is very small; therefore, there is not enough redundancy in the preclassification table to allow significant generalization.

The fact that the preclassifier is the implementation of a tree with limited depth enabled us to use some simple generalization techniques that achieved a significant and relatively accurate generalization.

### **7.3.1 Generalization of Suffixes**

As was mentioned above, the preclassification state automaton is the implementation of a loop-free tree, where the final states in the state machine correspond to the leaves of the tree and the output of each final state is the code corresponding to the sequence that leads to that final state.

The construction process is such that each state in the table calls only states that are after the calling state in the table and each state is called by only one state. If, therefore, the table is travelled from the last state to the first, each state will be visited before its calling state. In addition, as each final state contains only null next state calls, final states that correspond to the same ASCII character are identical. The above properties enabled us to develop a simple and efficient generalization algorithm that adds to the recognition logic sequences with suffixes similar to the learned sequences. Starting from a final state, the generalization process simply merges each state to its calling state until there is a conflicting call. As each state is after its calling state, a systematic merging of states starting from the bottom of the table will merge all the suffixes with only one pass over the table. The detailed steps of the algorithm are shown in Table 7.1.

### **7.3.2 Generalization of Internal Paths**

Small variations of the character bit maps can cause changes in the order in which the QTCS are detected. This type of change tends to affect the relative positions of the codes in the

subscan directions with respect to the codes in the scan direction. In order to include such variations of characters in the preclassification table without explicitly learning all possible sequences, a second stage of generalization is employed. Certain pairs of codes are considered interchangeable. For example, if a State A calls State B with code Oh and has no call for code Ch, while State B calls another State with code Ch and has no call for code Oh, then it is assumed that the sequence Oh,Ch and Ch,Oh are equivalent. The generalization process shown in Table 7.2 is based on this assumption.

**TABLE 7.1.** Generalization of suffixes.

---

**For every State in the table and Starting with the last state**

```
BEGIN
  IF the State is a final State
    BEGIN
      Enable Merging
    END
  ELSE IF the State calls only one State
    BEGIN
      IF Merging is Enabled
        BEGIN
          merge called State to current State
          Eliminate called State
        END
      END
    ELSE IF State calls more than one State
      BEGIN
        IF all called States are equivalent
          BEGIN
            Enable Merging
            Merge all called States to current State
            Eliminate all called States.
          END
        ELSE
          BEGIN
            Disable Merging
          END
        END
      END
    END
  END
```

**For every State in the Table Starting with the First State**

```
BEGIN
  Renumber current State
  Renumber called States
END
```

---

**TABLE 7.2.** Generalization of internal paths.

---

```
For every State in the Table
  IF State calls another State with code Oh
    BEGIN
      IF no call for Code Ch
        Copy calling State for code Oh to call for Ch
      IF no call for Code Cv
        Copy calling State for code Oh to call for Cv
    END
  ELSE IF State calls another State with code Ch
    BEGIN
      IF no call for Code Oh
        Copy calling State for code Ch to call for Oh
      IF no call for Code Cv
        Copy calling State for code Ch to call for Cv
    END
  ELSE IF State calls another State with code Cv
    BEGIN
      IF no call for Code Oh
        Copy calling State for code Cv to call for Oh
      IF no call for Code Ch
        Copy calling State for code Cv to call for Ch
    END
  IF State calls another State with code Xv
    BEGIN
      IF no call for Code Fv
        Copy calling State for code Xv to call for Fv
      IF no call for Code Ov
        Copy calling State for code Xv to call for Ov
    END
  ELSE IF State calls another State with code Fv
    BEGIN
      IF no call for Code Xv
        Copy calling State for code Fv to call for Xv
      IF no call for Code Ov
        Copy calling State for code Fv to call for Ov
    END
  ELSE IF State calls another State with code Ov
    BEGIN
      IF no call for Code Xv
        Copy calling State for code Ov to call for Xv
      IF no call for Code Fv
        Copy calling State for code Ov to call for Fv
    END
END.
```

---

## **7.4 Results with Russian and Roman Sets**

To demonstrate the versatility of the recognition logic, two separate sets of tables were constructed, following the described learning method. One set of tables was trained with a Cyrillic font and the second set of tables was trained with a number of different English fonts. The pages with Cyrillic text were scanned from top to bottom, as they would normally be scanned with a page scanner, while the Roman characters were scanned from left to right with a handheld scanner, so that they would be compatible with the way text is scanned by the developed handheld reading device.

### **7.4.1 Generalized Versus Ungeneralized Learning**

The generalization techniques were evaluated by comparing the statistics of the learning process when a generalization technique was applied between iterations with the statistics of the learning process when the same set of characters was learned without any generalization.

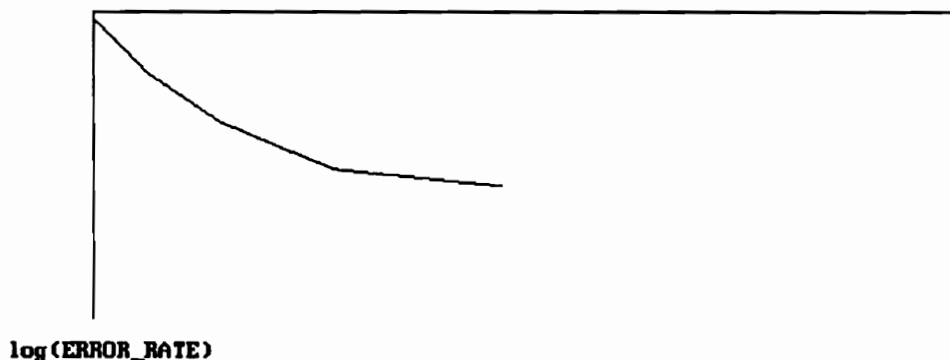
A set of Cyrillic characters was used to generate the learning statistics shown in Fig 7.3 and Fig 7.4. Fig 7.3 shows the error rates for five iterations without any generalization, while Fig 7.4 shows the error rates for five iterations with generalization between each iteration. It can be seen that the learning is significantly and consistently faster when the preclassification table is generalized at each iteration.

**LEARNING PROCESS STATISTICS**

**NO GENERALIZATION**

SAMPLE SIZE :	160	320	640	1280	2520
DISTINCT CHARS:	121	173	263	390	695
REJECTIONS :	151	211	287	403	753
SUBSTITUTIONS :	0	4	24	51	50
TABLE SIZE :	929	2116	3573	5700	9417
ERROR RATE :	0.944	0.672	0.486	0.355	0.319

**NUMBER OF PROTOTYPES**



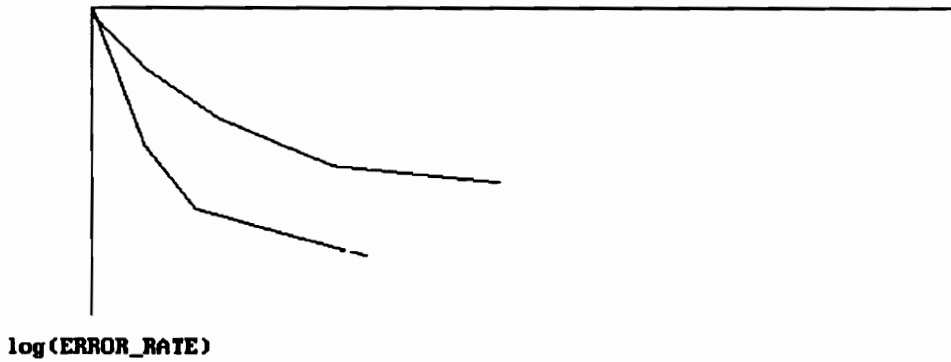
**FIGURE 7.3.** Ungeneralized Learning.

**LEARNING PROCESS STATISTICS**

**GENERALIZED TABLES**

SAMPLE SIZE :	160	320	640	1280	2520
DISTINCT CHARS :	121	117	161	237	424
REJECTIONS :	160	84	101	166	346
SUBSTITUTIONS :	0	46	71	137	148
TABLE SIZE :	929	1779	2732	4031	6426
ERROR RATE :	1.000	0.406	0.269	0.237	0.196

**NUMBER OF PROTOTYPES**



**FIGURE 7.4.** Learning with Generalization.

#### 7.4.2 Generalized Learning Statistics

Here we present and discuss the learning statistics for the two training sets mentioned earlier. Fig. 7.5 shows the preclassification error rates per iteration for the Cyrillic set of characters, while Fig 7.10 shows the same data for the Roman set. In both cases we observe distinct changes of the learning rate, which we believe are related to the distribution of "easy" and "difficult" characters in the alphabet. Although we did not try to prove the above statement, we could see, by analyzing the new prototypes that were added in each iteration (Table 7.3 shows this distribution), that at the steeper parts the learning curve was dominated by the "easier" characters (shorter QTC strings and less variations), while the more "difficult" characters (longer QTC string with more variations) were the majority of new prototypes at the flatter parts of the curve. From Figs 7.9a, 7.9b and 7.9c, which show the distribution of new prototypes at iterations 4, 10 and 18 of the Cyrillic set, it can be seen that the percent of joined characters (which are described by long QTC strings) increases with each iteration. Similarly, Figs 7.14a, 7.14b and 7.14c show that in the Roman set the percent of prototypes with more complicated shapes increases with each iteration. Negative slopes seem to be caused by the fact that new prototypes introduced constraints that inhibited certain generalizations. It can be seen, though, that after one or two iterations the "forgotten", characters were learned again and the error rates continued to drop.

In Fig. 7.6 and Fig. 7.11 are shown the percentage of characters in the two sets that entered into confusion classes. In the Cyrillic font (Fig 7.6), where many upper and lower case characters have identical shapes, the percentage saturates at about 78 %, while in the Roman fonts (Fig 7.11) the percentage of characters creating confusion classes saturates at about 65%. It must be pointed out that the distribution of characters in the Roman Fonts is not representative of their distribution in typical text. Figures 7.8 and 7.13 show the distribution of characters in the Cyrillic and Roman sets respectively.

Finally, in Figs 7.7 and 7.12, the sizes of the generalized and ungeneralized preclassification tables are shown. As can be seen, the size of the generalized tables is significantly smaller than the size of the corresponding ungeneralized tables.

**TABLE 7.3.** Distribution of prototypes per iteration.

---

ITERATION #	RUSSCII	ROMAN	LEFT	RIGHT	CONNECTED
1	209	6	0	0	1
2	190	2	9	10	3
3	155	7	0	2	6
4	124	4	6	0	5
5	122	1	1	0	8
6	108	2	4	3	12
7	166	0	3	6	4
8	90	0	3	6	8
9	89	5	2	2	12
10	98	2	7	9	5
11	93	1	1	3	13
12	108	0	1	2	21
13	82	0	1	1	22
14	82	1	13	4	18
15	89	0	6	4	10
16	103	0	1	1	18
17	95	1	0	1	25
18	96	0	1	1	14
19	82	0	1	1	24

---

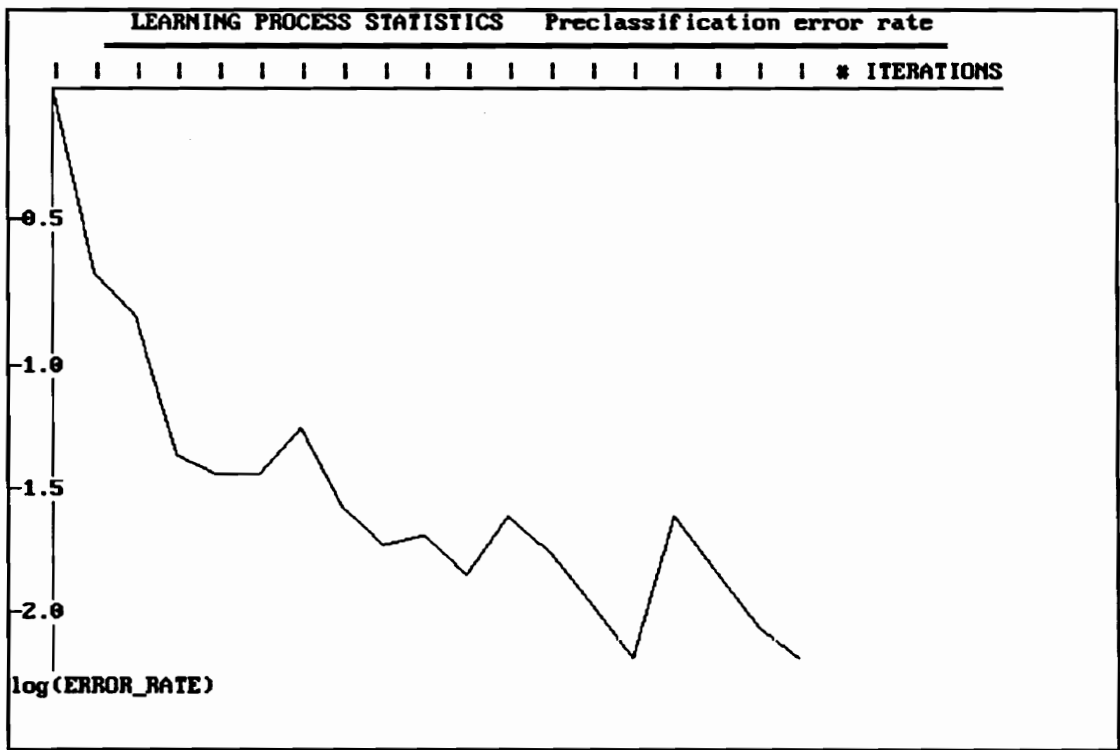


FIGURE 7.5. Preclassification error rates (Cyrillic Set).

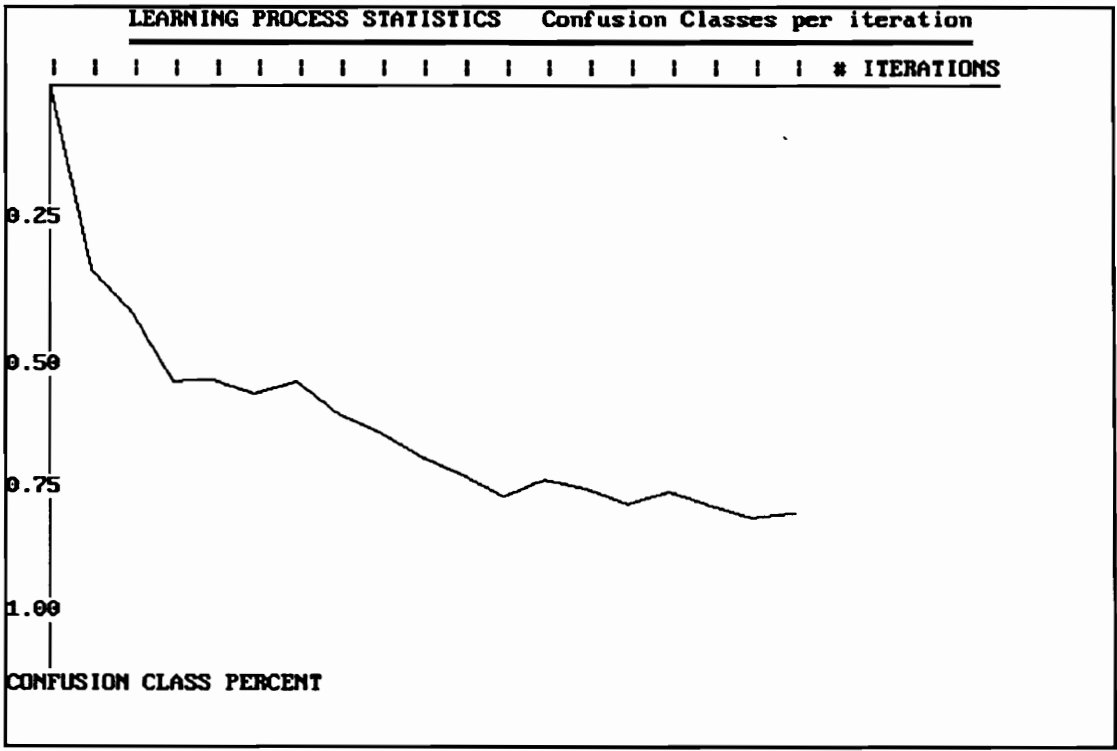
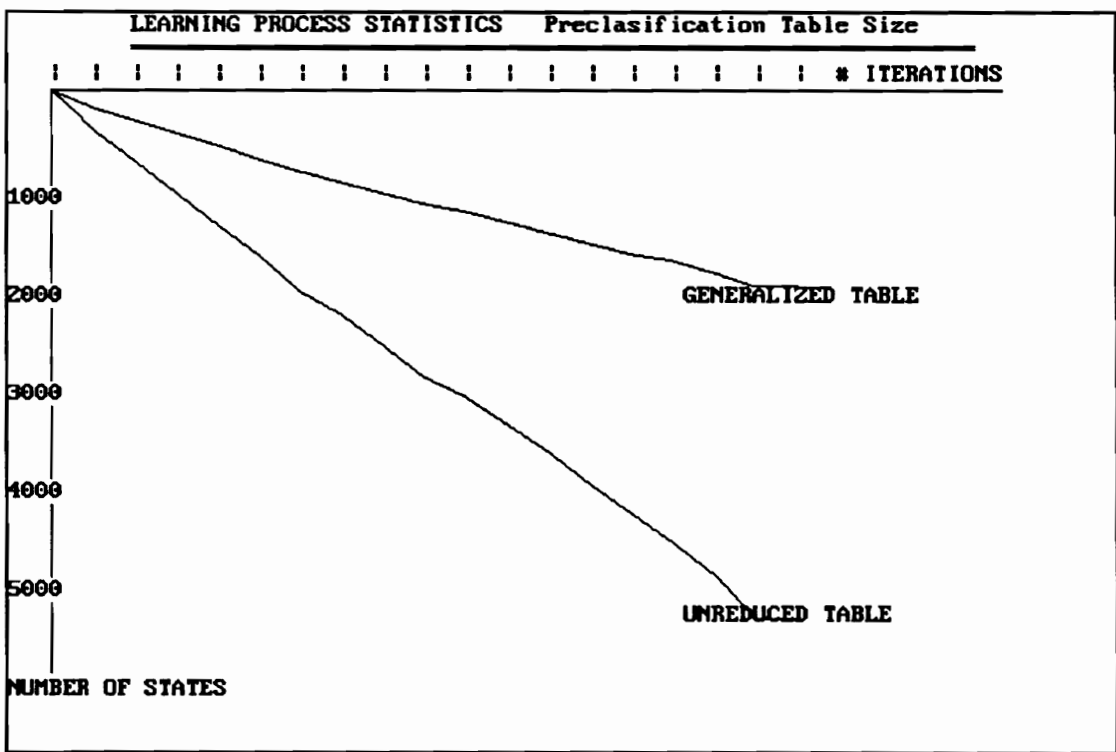
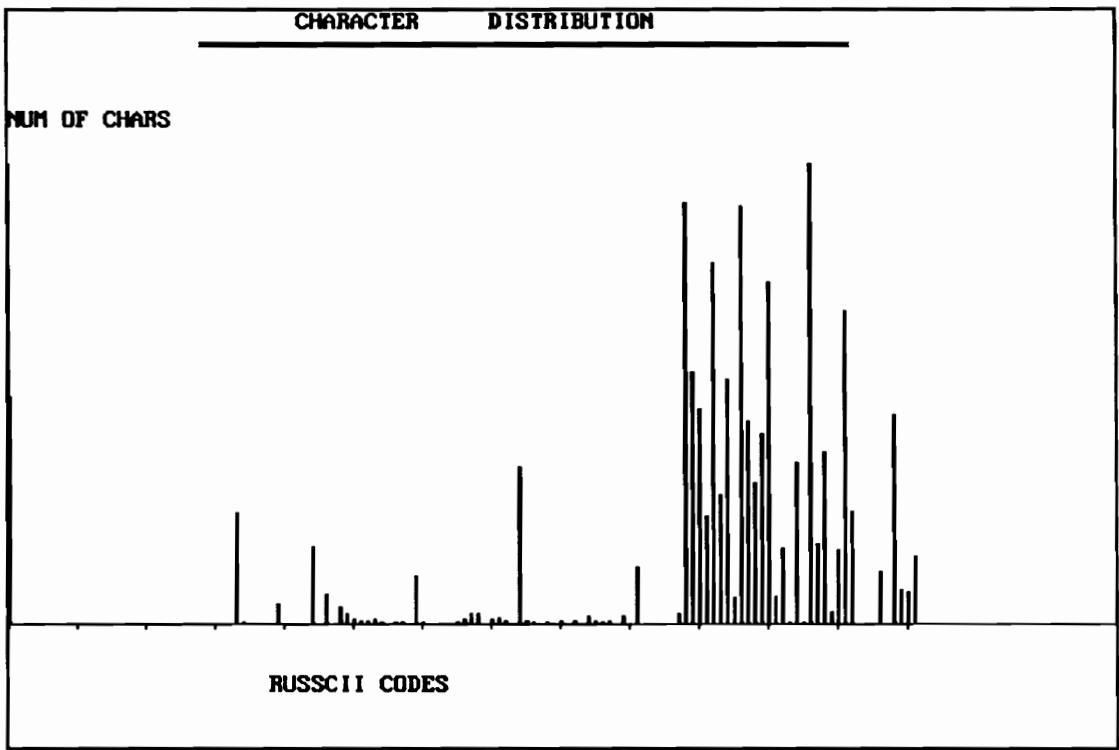


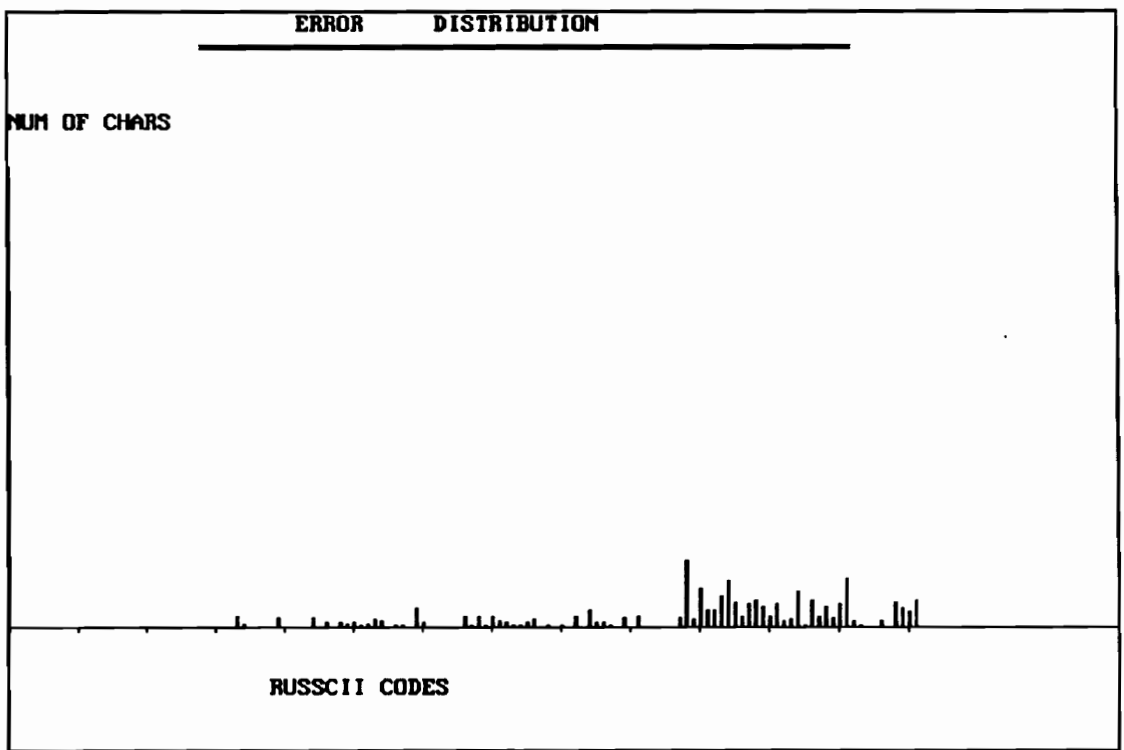
FIGURE 7.6. Percent of characters in Confusion Classes.



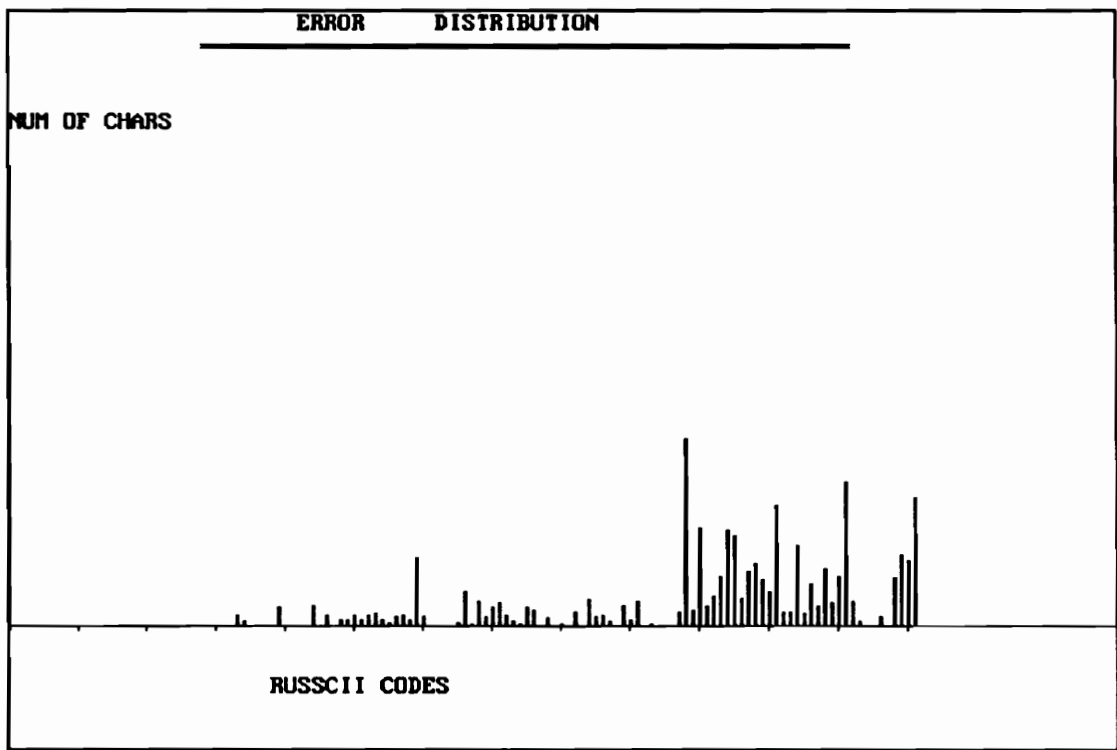
**FIGURE 7.7:** Preclassification table size (Cyrillic set).



**FIGURE 7.8.** Character distribution in Cyrillic text.



**FIGURE 7.9a.** New Cyrillic prototypes at 4th iteration.



**FIGURE 7.9b.** New Cyrillic prototypes at 10th iteration.

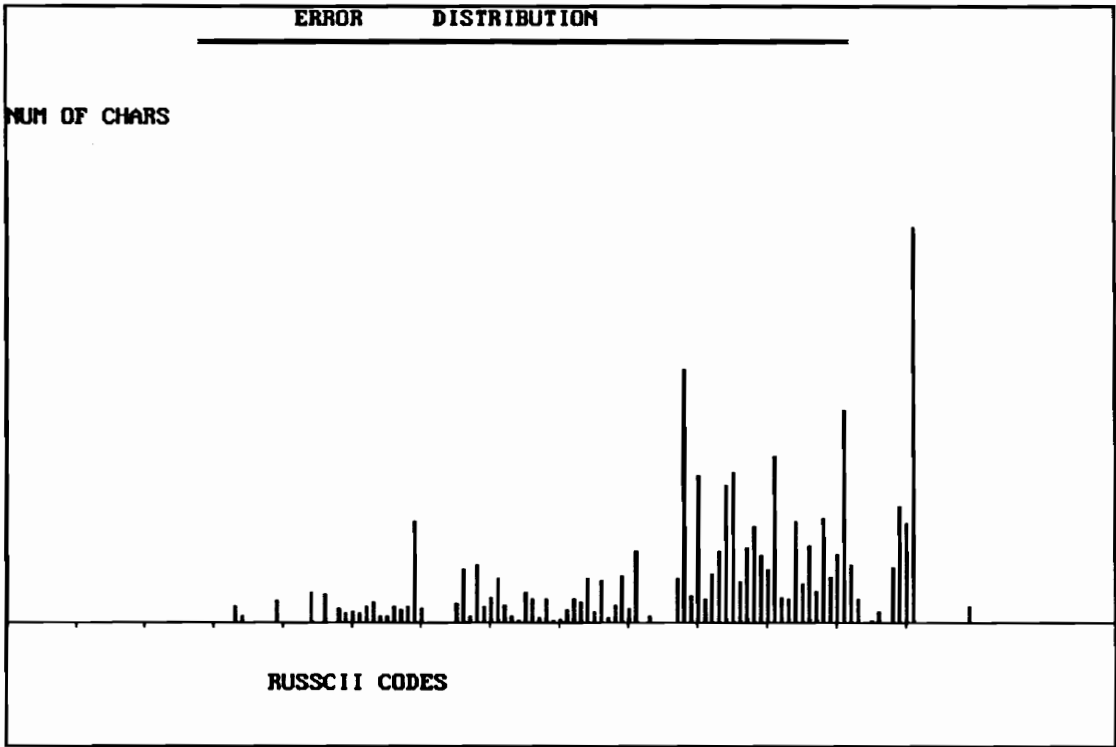


FIGURE 7.9c. New Cyrillic prototypes at 18th iteration.

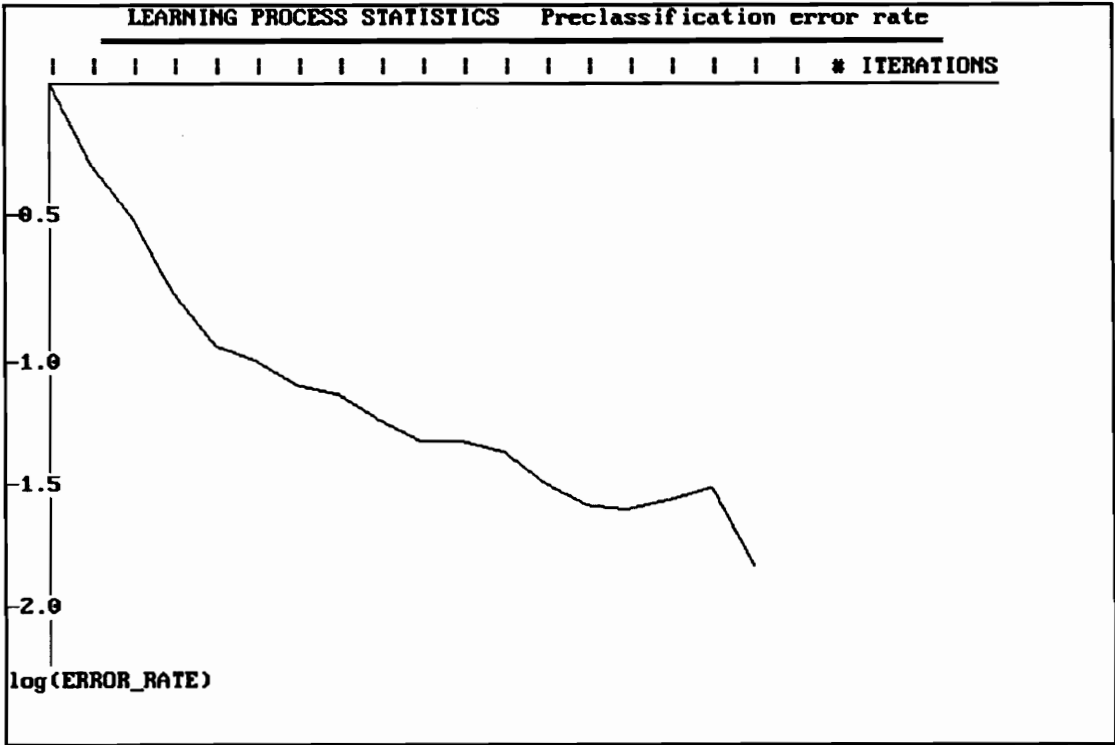


FIGURE 7.10. Preclassification error rates (Roman set).

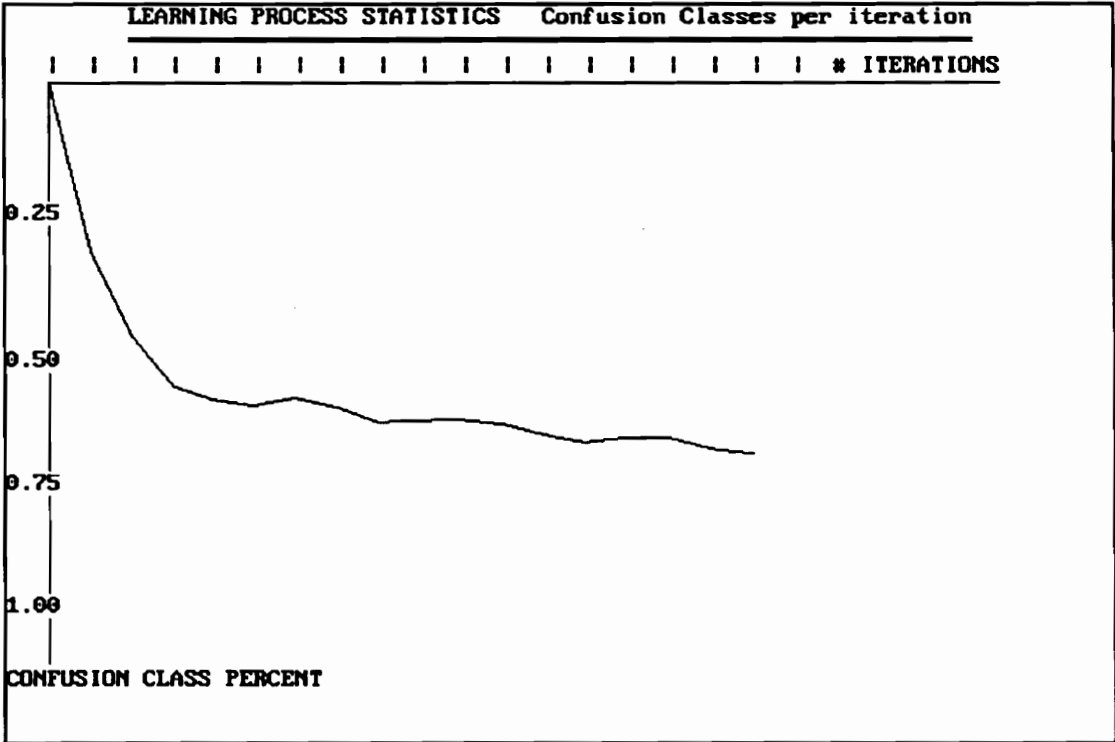


FIGURE 7.11. Percent of characters in confusion classes (Roman set).

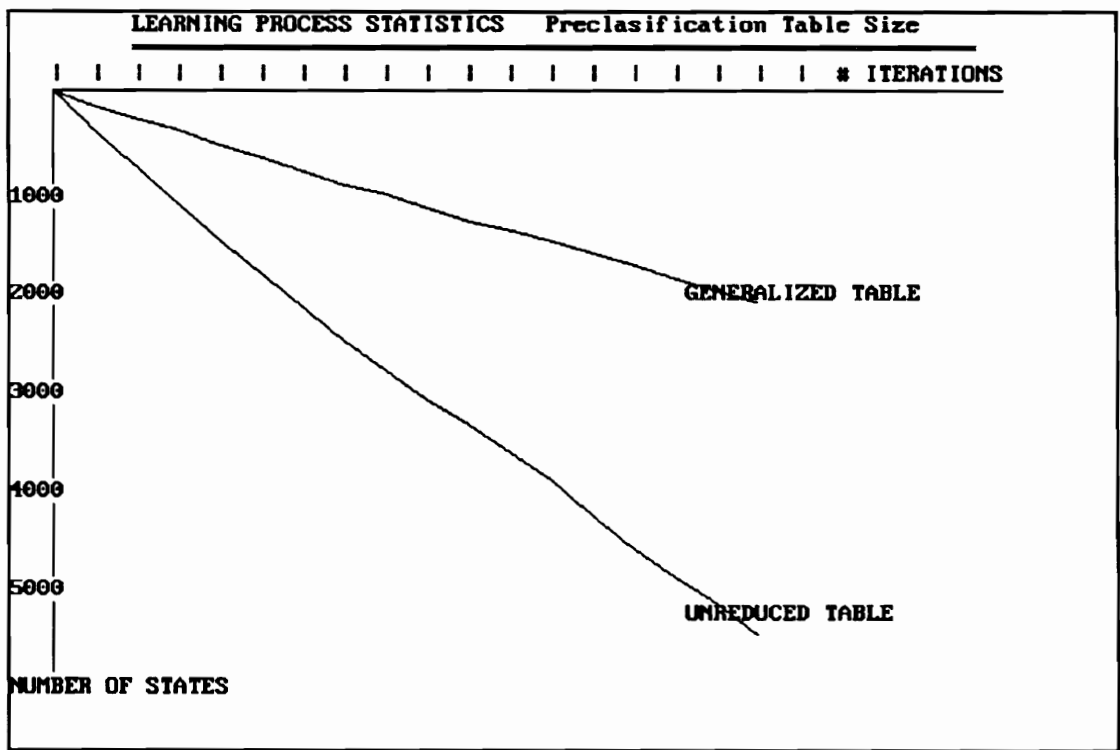
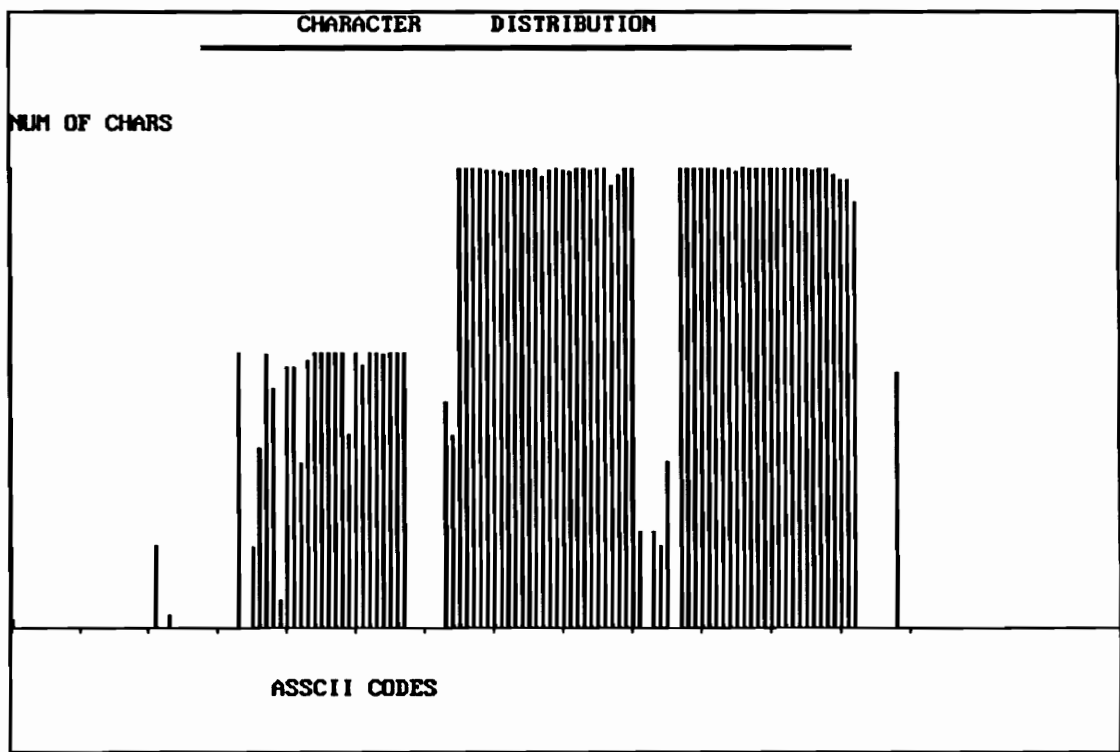


FIGURE 7.12. Preclassification table size (Roman set).



**FIGURE 7.13.** Character distribution in Roman set.

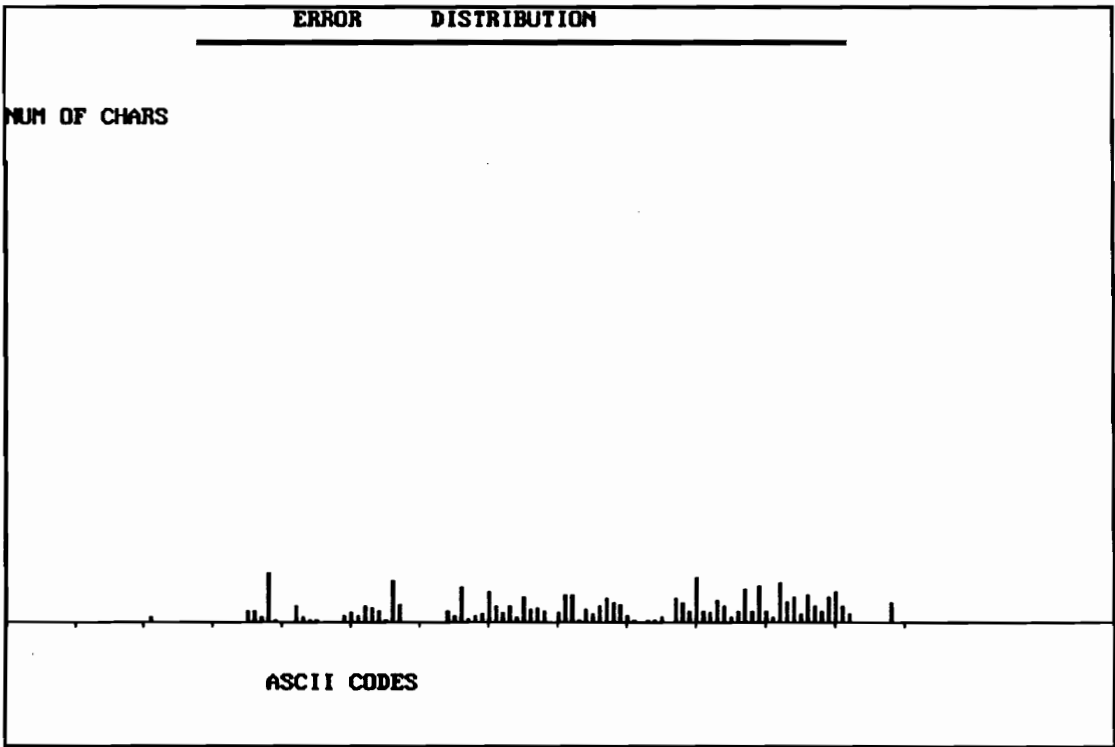
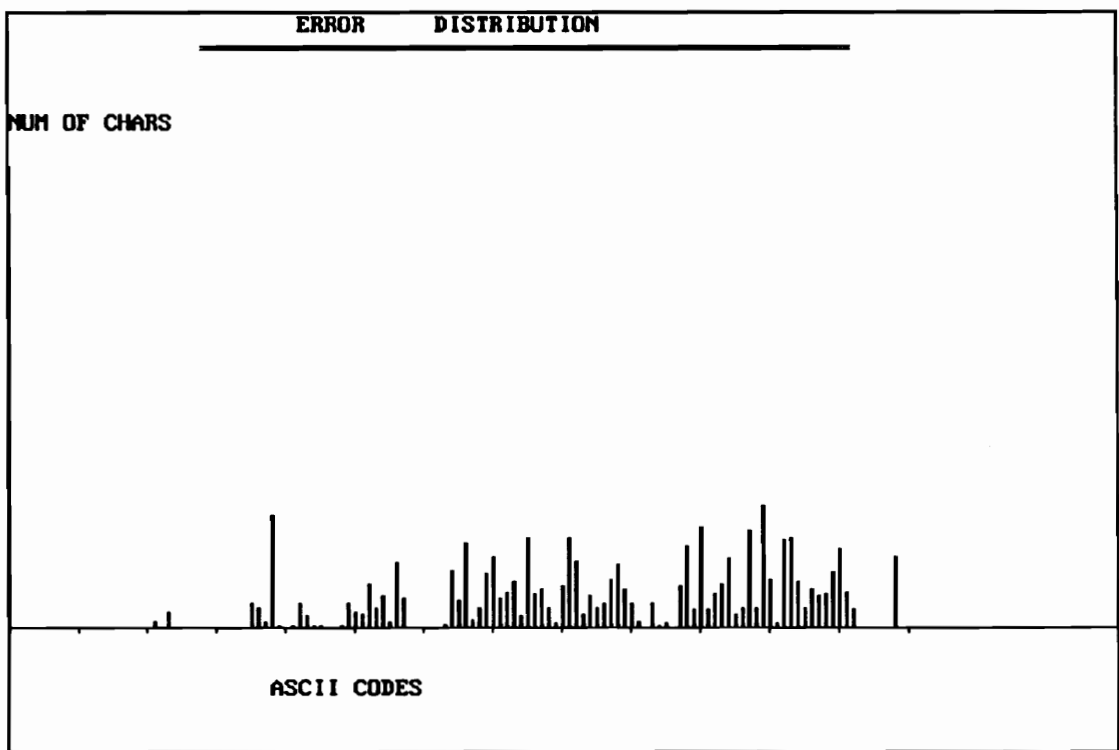
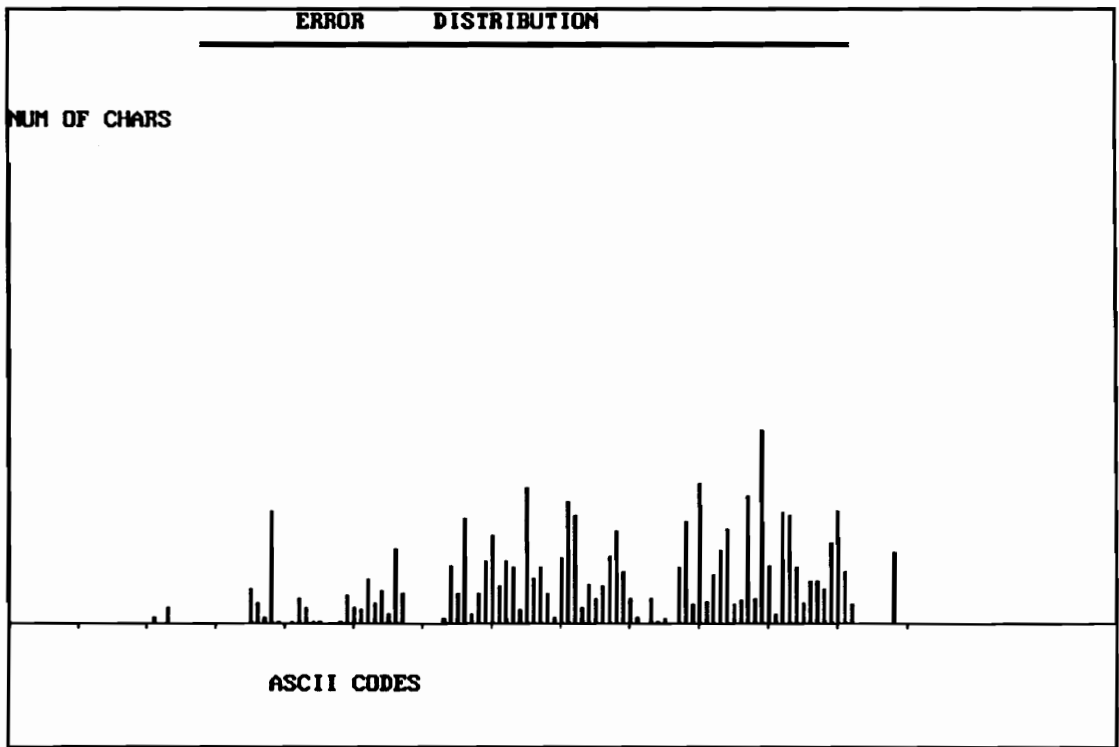


FIGURE 7.14a. New Roman Prototypes at 4th iteration.



**FIGURE 7.14b.** New Roman Prototypes at 14th iteration.



**FIGURE 7.14c.** New Roman Prototypes at 18th iteration.

## **7.5 Iterative Design Process of Metric Tests**

In Chapter 6 we discussed the second step of the recognition process, which is based on metric comparisons of the coordinates associated with each QTC. Here we discuss the iterative process with which these tests were designed.

An initial set of metric tests was designed using one prototype for each member of a confusion class and the initial ranges were set to make the tests as tight as possible. For each feature present in a particular class, we computed the minimum and maximum values of the associated tests. For example, referring to the 'a' and 'd' shown in Figs 6.1 and 6.2, we used the first instance of an 'a' and the first instance of 'd' to compute the initial range of x and y coordinates associated with the Fv code. As with the preclassification tables, a set of characters was tested until a certain number of characters failed the current metric tests. The action that followed each iteration depended upon the types of errors that were encountered.

### **7.5.1 Automatic Adjustment of Strong Tests**

If a character was completely rejected, then some of the metric tests in the particular class were unnecessarily tight; therefore, the addition of the rejected character to the design set would eliminate the redundant constraints. If a character was misrecognized, then some of the metric tests were faulty and adding the character to the design set would eliminate the faulty constraints. Based on the above, all the characters that were rejected or misrecognized were added to the design set and the metric tests were updated. Referring to Figs 6.1 and 6.2 again, we used the misrecognized and rejected instances of 'a' and 'd' to update the range of the x and y coordinates associated with the code Fv. Similarly we updated the ranges of all the other metric tests. The resulting ranges are the ones shown in Table 6.6.

### **7.5.2 Manual Adjustment of Weak Tests**

If no unique decision could be made for some character, then the tests in the particular confusion class were considered weak. That meant that either there were some erroneously

labeled characters in the set or that the present set of features was not adequate. In either case, addition of the character to the design set would not change the result of the particular metric test. Therefore, when a weak test was detected, manual intervention was necessary. A visual inspection could show whether the metric tests could be modified or if the confused characters were virtually indistinguishable. If the characters could be separated, the proper changes were manually made and the tests related to the particular confusion class were redesigned. Efforts to automate this procedure were not very successful, especially because of the computational complexity required. Fortunately, though, the required manual intervention was minimized and limited to a very small percent of characters because of the good selection of features and tests.

## **7.6 Results with Russian and Roman Fonts**

Figures 7.15 and 7.16 show the metric test error rates per iteration for the Cyrillic and Roman set, respectively, while Table 7.4 and Table 7.5 list the types of errors per iteration for the same sets of characters.

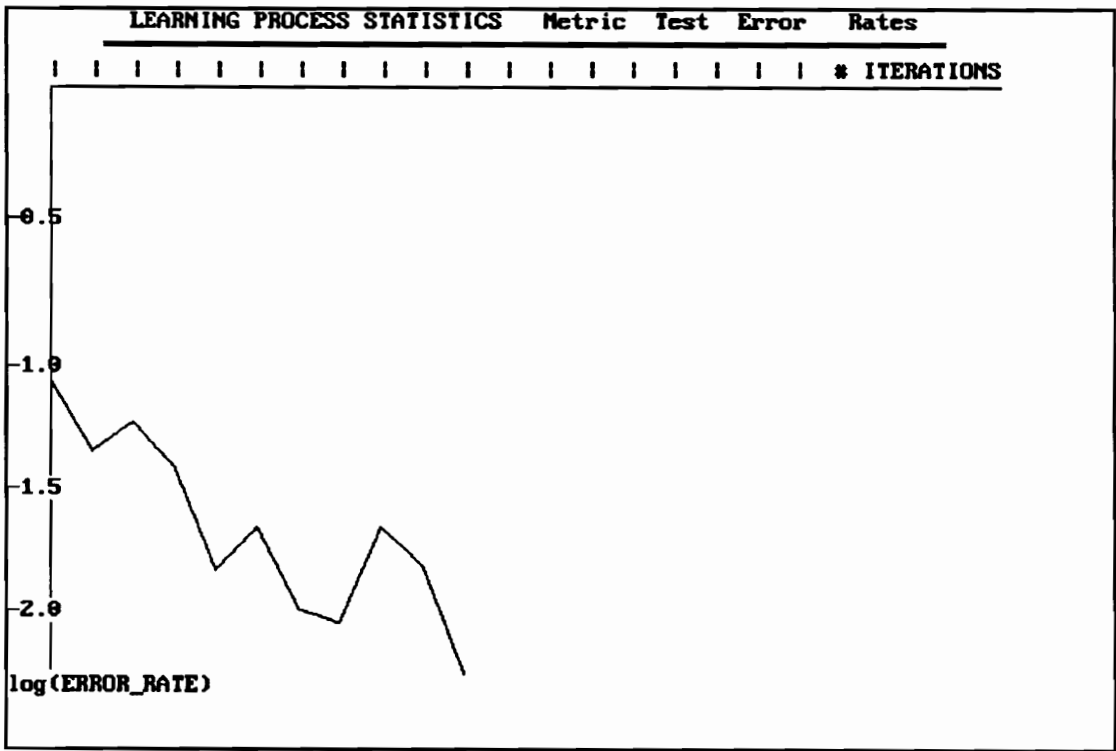


FIGURE 7.15. Metric test error rates (Cyrillic set).

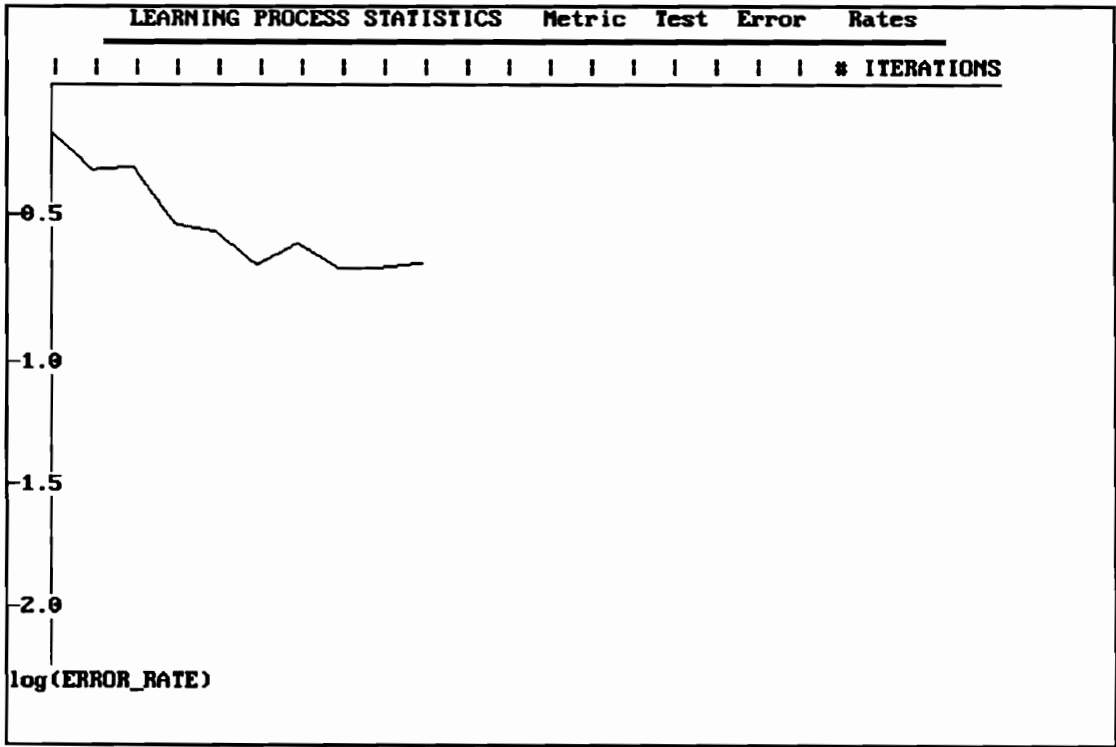


FIGURE 7.16. Metric test error rates (Roman set).

**TABLE 7.4. Error types per metric test iteration (Cyrillic Set).**

#ITER	TOTAL	REJ	SUB	WEAK	ERRORS	RATE
1	321	10	15	0	25	0.0778
2	595	16	8	0	25	0.0420
3	461	14	9	1	25	0.0542
4	682	12	13	0	25	0.0366
5	1683	14	10	0	25	0.0148
6	1149	12	12	0	25	0.0217
7	2327	15	9	1	25	0.0107
8	2640	8	15	2	25	0.0095
9	1153	9	16	0	25	0.0217
10	1621	12	7	3	25	0.0154
11	4076	9	4	9	25	0.0061

**TABLE 7.5. Error types per iteration (Roman Set).**

ITER#	TOTAL	REJ	SUBST	WEAK	ERRORS	ERROR RATE
1	38	16	5	4	25	0.657
2	53	12	6	6	25	0.471
3	52	4	5	16	25	0.471
4	85	10	12	2	25	0.294
5	91	9	12	4	25	0.274
6	121	5	12	7	25	0.206
7	101	8	9	8	25	0.247
8	125	6	7	12	25	0.200
9	124	6	3	14	25	0.200
10	120	1	9	14	25	0.208

## **7.7 Comments**

The initial results we obtained both from the set of Cyrillic characters and the set of Roman characters demonstrated that the recognition process we developed behaves in a healthy manner and can be trained to achieve very high recognition rates. We did not see any apparent saturation of the learning curves and, at the same time, the size of the design set of characters seems to remain finite.

## 8. SUMMARY AND CONCLUSIONS

In this work we investigated the various items that compose a complete handheld optical character recognition system. We examined the problems that are unique to such a device and developed solutions that, we believe, can be combined into useful and reliable reading aid for visually impaired users. Starting from the scanning device itself and ending with a reliable and real time recognition subsystem, we faced a series of unique problems that dictated the solutions that we proposed.

The requirement for good quality images, presented us with the need to develop a high speed image processing board [58] which allows accurate capture of poor quality text. This board, which was developed in parallel to this investigation, is currently used as a commercial product for document restoration and can process up to one hundred typical documents a minute. In addition to the high speed image processing board, we had to evaluate commercially available handheld scanners and select the one with the best optical and electrical characteristics. We concluded that existing scanning heads did not meet the requirements for a reliable text capturing unit, but, as our expertise and means did not allow us to develop a better scanner, we decided to modify one commercially available scanner for our prototype. If the need arises, we plan to build a scanning head that will generate better bitmaps of the scanned text.

The velocity compensation algorithm that was developed was successfully implemented in a small board that could fit in a portable scanning unit. The results with a variety of documents were very satisfactory. When we used a high quality linear sensor to capture data we achieved excellent velocity compensation. For actual velocities ranging from 10 to 1 of the maximum allowable scanning velocity (nominal velocity), we obtained apparent (compensated) velocities 1.5 to 1 of the nominal velocity. This means that the user can scan a document without being concerned about the speed with which the handheld wand is moved, a feature which makes manual scan very comfortable. As the velocity compensation algorithm is based on the assumption that the print is locally uniform across the document, electrical noise can cause

undesirable artifacts to the velocity compensation. The part of the scanning unit which is more sensitive to noise, is the analog to digital converter which translates analog electrical signals to binary numbers. If, for example, the power supply produces a dc voltage that fluctuates a small percentage around the ideal voltage, then the analog to digital converter tends to interpret those changes as changes on the input analog signal and the result is a digitized signal that contains the fluctuations of the power supply. These fluctuations are, in turn, interpreted by the velocity compensation hardware as motion of the scanning head and the final result is distorted bitmaps of the scanned characters. Experiments performed with noisy scanners showed that fluctuations of 5 percent or more of the power supply voltage can considerably affect the developed velocity compensation unit.

Real time segmentation and feature extraction was a necessary feature for the developed system. In order to provide useful feedback to a visually impaired user, the unit must analyze the text as it is being captured. This task was accomplished by developing hardware that can extract the Quasi-Topological Codes at pixel rates. The actual detection of each QTC must be completed within a pixel cycle. It was, therefore, necessary to design high speed hardware to perform all the required operations. In the developed prototype, real time analysis of these codes and proper segmentation is achieved by a high speed personal computer, that acts as a dedicated microcontroller (all interrupts are disabled); in a production version, however, these tasks will be performed by an actual dedicated microcontroller. The required sequence of operations is short and consists of simple data transfer instructions and a small number of comparisons; it is, therefore, ideally suited for a low cost microcontroller or RISC.

The selection of QTCs as segment describing features proved very successful. In addition to accurately describing characters, they enabled us to develop a very fast and simple skew detection algorithm. The segmentation process has been enhanced so that the physical position of each segment within the scanned document becomes available to the skew detection as soon as the segment is completely acquired; a simple comparison of its position with previously acquired segments can provide adequate information for reliable skew detection and, later, correct reconstruction of the text. The exact format of the acoustic feedback has not yet been defined,

but multiple experiments showed that normal printed or typed text can be correctly traced with skews of up to 15 degrees. We need, though, to further examine some exceptional cases of artistic text lay-outs, where a large variety of character sizes coexist in the same area of a document. In these cases, more complicated line tracing procedures may be required.

A related problem which needs further investigation is graphics handling. When the entire bitmap is available, it is possible to globally analyze the document and isolate the areas that contain graphics. In the case, though, of a handheld scanner with a small scanning window it is not possible to reliably separate graphic objects from text. A more global analysis of the document is required. One possible solution is to store global information about the document with each consecutive manual scan and reconstruct the text and graphics from the individual pieces; however, such a solution would require significant computing power which cannot as yet be made available to a small portable scanning device. For all practical purposes, we consider as graphics any segment whose complexity and/or size are significantly greater than previously acquired characters.

The heart of any optical recognition system is the recognition algorithms that it employs and their implementation. As we have discussed in this dissertation, a hierarchical recognition method which utilizes structural, statistical and contextual analysis can achieve very high recognition rates with a multiplicity of fonts and sizes. In this work, we developed an attributed grammar in which the QTCs serve as the syntactic primitives and their coordinates are the metric attributes that provide additional information when necessary. The QTC sequence of an unknown segment is first tested by a syntactic acceptor (preclassifier) and, if it cannot be unambiguously identified, a statistical analysis of the QTC coordinates (metric tests) is performed to resolve the ambiguities. If, after the first two steps of the above hierarchical recognition process, a segment has not been identified, a final contextual analysis (postprocessing) step is performed.

During the development of this recognition process we encountered several interesting challenges which dictated some of the solutions we proposed. We had to evaluate the amount of information contributed by each primitive and attribute so that at each recognition step we retained only useful information, simplifying thus the algorithms and speeding up the process.

In the case of syntactic analysis we found out that the presence of Dh and Fh codes does not add any significant structural information; therefore, we eliminated their analysis in the preclassification step. On the other hand, their relative positions with respect to the remaining codes can provide very useful information when we must distinguish among characters with the same topologies; the majority of metric tests that were developed are based on metric differences of coordinates associated with Dv, Fv, Dh and Fh codes. In order to achieve high recognition rates that would be independent of character stretching, we limited the metric tests to metric differences along one direction, main scan direction or subscan direction, making thus the tests independent of the actual character size and aspect ratio.

In order to accurately estimate the error rates, we used real characters, sampled over a large number of fonts in order to train the recognition logic. The grammar rules were inferred from these characters and implemented in a finite state automaton; they were then generalized in order to include characters with topologies similar to those in the training set. Several generalization algorithms were evaluated, ranging from formal state reduction to random addition of next states. Finally, we decided to implement two simple generalization rules that were based on statistical data obtained from the large training set: We observed that several pairs of codes tend to permute due to small skews or print defects; for this reason we implemented a generalization algorithm which examines acquired sequences of QTCs and adds to the preclassification table sequences that are obtained by permuting certain codes. The second generalization technique we employed is based on generalization of unique suffixes; if after a number of QTCs a sequence leads to a unique character, then a decision is made immediately after the last branch. These simple generalization techniques provided significant reduction of the learning period and of the size of the preclassification table.

Similar generalization techniques were designed for the second recognition step, the metric tests. Several training techniques were evaluated, including manual specification of the metric features to be considered with each class of similar characters. The final decision was to specify a set of possible structural features (for example number of horizontal finishes) and then automatically detect the features present in a given class and update the metric tests associated

with these features. Generalization was achieved in the following manner: during training, the maximum range of each metric test is monitored. If the range of a test is identical for all the members of a confusion class, then that specific test is considered redundant and is eliminated from the list of metric tests for the class under consideration. Similarly, if the measurements of a certain tests range over a large area of a character space, the test is considered unreliable and is also eliminated from the list.

Tests with a variety of fonts in two alphabets, Roman and Cyrillic, showed very healthy performance of the proposed recognition system and demonstrated its omnifont recognition capability. We are currently developing high speed recognition systems that will achieve error rates less than  $10^{-4}$  with Roman and Cyrillic characters.

Other areas that are still under investigation include reconstruction of broken characters, a problem that is very common in many applications. Bad photocopiers or defective printing devices tend to break characters in certain spots where the lines are thin. These breaks alter the topology of a character and result in erroneous character recognition. We are currently analyzing the nature of these breaks, to find a reliable method of reconnecting the pieces.

In summary, we have investigated the unique problems associated with a handheld optical character recognition device that can be used as a reading aid for the visually impaired. We analyzed each step of the process, from the handheld scanning head to the final recognition processing unit, and proposed solutions to all related problems. Some of the proposed solutions need further investigation, which we hope will be stimulated by this dissertation.

## REFERENCES

1. G. Tauscheck, Reading Machine, U. S. Patent 2 026 329, Dec 1935.
2. J. W. Bryce, Reading Machine, U. S. Patent 2 240 546, May 1941.
3. Philco Corporation, Character Reading Apparatus, U.K. patent 1 124 454, Aug 1968.
4. R. B. Hennis, "The IBM 1975 Optical Page Reader," IBM Journal of Research and Development, vol 12, no 5, Sept 1968.
5. H. S. Baird, K. Thompson, "Reading Chess," IEEE Trans on PAMI, vol 12, no 6, pp 552-559, June 1990.
6. K. S. Fu, Syntactic Pattern Recognition and Applications, Prentice-Hall, Englewood Cliffs, N.J. 1982.
7. A. C. Shaw, "A Formal Picture Description Scheme as a Basis for Picture Processing Systems," Information and Control, 14, pp 9-52, 1969.
8. J. Feder, "Plex Languages," Info. Sci. 3, pp 225-241, 1971.
9. R. C. Gonzalez, P. Wintz, Digital Image Processing, Addison-Wesley, Nov 1987.
10. J.R. Ullman, Pattern Recognition Techniques, Butterworth, London, 1973.
11. C.H. Cox, P. Coueignoux, B. Blesser, M. Eden, "Skeletons: A Link Between Theoretical and Physical Letter Descriptions," Pattern Recognition, Vol 15, No 1, pp 11-22, 1982.
12. T.Y. Zhang, C.Y. Suen, "A Fast Parallel Algorithm for Thinning Digital Patterns," Comm. ACM, vol 27, no 3, pp 236-239, 1984.
13. H.E. Lu, P.S.P. Wang, "A Comment on a Fast Parallel Algorithm for Thinning Digital Patterns," Comm. ACM, vol 29, no 3, pp 239-242, 1986.
14. T. Pavlidis, "A Vectorizer and Feature Extractor for Document Recognition," Computer Vision Graphics and Image Processing, vol 35, pp 111-127, 1986.
15. B. K. Jang, R. T. Chin, "Analysis of Thinning Algorithms Using Mathematical Morphology," IEEE Trans on PAMI, vol 12, no 6, pp 541-551, June 1990.
16. S. Kahan, T. Paulidis, H. S. Baird, "On the Recognition of Printed Characters of Any Font and Size," ATT Bell Laboratories, Jan 1986.

17. B. Blesser, R. Shillman, C. Cox, T. Kuklinski, J. Ventura, M. Eden, " A Theoretical Approach to Character Recognition Based on Phenomenological Attributes," Proceedings of 1st Int. Joint Conference on Pattern Recognition, pp 33-40, 1973.
18. L. Uhr, "Flexible Linguistic Pattern Recognition," Pattern Recog., vol 3, pp 363-383, 1971.
19. H. Freeman, "Picture Processing and Psychopictorics," ed. by Lopkin, Rosenfeld, Academic Press, New York, pp 241-266, 1970.
20. H. Freeman, "Computer Processing of Line Drawings," Comput. Surveys, vol 6, pp 57-59, 1974.
21. Pinstov, Koay, Stone, Tan, Tuttle, Buck, "High speed pattern Recognition System for alphanumeric handprint characters," Scan-Optics Corp, 1982.
22. T. Pavlidis, "Segmentation of Pictures and Maps through Functional Approximation," Comp. Graph. Image Proc., vol 1, pp 360-372, 1972.
23. T. Pavlidis, "Structural Pattern Recognition," Springer-Verlag, New York, 1977.
24. J. Sklansky, R.L. Chazin, B.J. Hansen, "Minimum-Perimeter Polygons of Digitized Silhouettes," IEEE Trans. Comp., vol C-21, no 3, pp 260-268, 1972.
25. M. Nadler, "Sequentially-Local Picture Operators," 2th Int. Conf. on Pattern Recognition, pp 131-135, Aug 1974.
26. M. Nadler, "Structural Codes for Omnifont and Handprinted Characters," 3th Int. Conf. on Pattern Recognition, pp 135-139, Nov 1976.
27. M. Nadler, "Structural Codes for Omnifont and Handprinted Characters: II," 5th Int. Conf. on Pattern Recognition, pp 848-852, Dec 1980.
28. M. Nadler, "Hybrid Pattern Recognition: A Synthesis of the Numerical and Qualitative Approaches," Pattern Recognition Theory and Applications, Kittler, Fu, Pau eds, pp 177-187, 1982.
29. A. Vallet, M. Nadler, "Hierarchic Sequential Decision Logic for Optical Patern Recognition Based on Learning Procedures," IFIP Conf., New York, May 1965.
30. P.J. Nahim, "The Theory of Measurement of a Silhouette Description for Image Processing and Recognition," Pattern Recog., vol 6, no 2, pp 1338-1344, 1974.
31. D. H. Ballard, C.M. Brown, Computer Vision, Prentice-Hall, Englewood Cliffs, N.J. 1982.
32. Tadao Ichikawa, "On-Line Recognition of Handprinted Characters with Associative Read-out of Patterns in a Memory," pp 206-207, 1975.

33. A.A. Spanjerberg, "Combinations of Different Systems for the Recognition of Handwritten Digits," pp 208-209, 1975.
34. Gerard Gaillat, "An On-Line Character Recognizer with Learning Capabilities," pp 305-306, 1975.
35. J. R. Ward, T. Kuklinski, " A Model for Variability Effects in Hand-printing, with Implications for the Design of On-Line Character Recognition Systems, 1986.
36. W. Kuckuck, B. Rieger, K. Steinke, "Automatic Writer Recognition," 1979 Carnahan Conference on Crime Measurements, Lexington, Kentucky, May 1979.
37. D. J. Burr, "Designing a Handwriting Reader," Bell Laboratories, 1981.
38. R. W. Ehrich, K. J. Koehler, "Experiments in the Contextual Recognition of Cursive Script," IEEE Trans. Comp., C-24(2), pp 182-194, 1975.
39. K.M. Sayre, "Machine Recognition of Handwritten Words: A Project Report," Pattern Recog., vol 5, pp 213-228, 1973.
40. A.R. Hanson, E.M. Riseman, E. Fisher, "Context in Word Recognition," Pattern Recog., vol 8, pp 35-45, 1976.
41. R. M. Bozinovic, S. N. Srihari, " Off-Line Cursive Script Word Recognition," IEEE Trans. on PAMI, vol 11, no 1, Jan 1989.
42. J. R. Fruchterman, "Omnifont Text Recognition," Inform, Vol 2, no 5, pp 16-19, May 1988.
43. Caere Corp, OnmiPage(TM), AnyFont(TM), Omnifont Optical Character Recognition, User's Manual, 1989.
44. Xerox Imaging Systems, Inc, Accutext(TM), Multifont Recognition System, Techniacal Reference Manual, 1990.
45. CTA, TextPert(TM), Trainable Optical Character Recognition Software, User's Manual, March 1990.
46. Calera Recognition Systems, Inc, TrueScan(TM), High speed Omnifont OCR, User's Manual, 1988.
47. OCR Systems Inc., Readright(TM), Technical Reference manual, 1988
48. Recognition Equipment Incorporated, TARTAN XP 80, Optical Characater Recognition System, Product Definition Manual, 1988
49. Kurzweil Computer Products, Discover 7320, Intelligent Character Recognition System, 1987.

50. Soricon Corporation, DataSweep 2 (TM), Handheld Optical Character Recognition, 1990.
51. Inovatic, ReadStar III Plus, Omnifont Character Recognition System, User's Manual, 1990.
52. The Complete PC, Inc, The Complete Hand Scanner, Technical Reference Manual, 1988.
53. Oberon, Hand Held Optical Character Recognition, Technical Reference manual, 1986.
54. M. Nadler, "Output-Extended Sequential Machines," Proc. of the IEEE, vol 67, no 5, pp 864-867, May 1979.
55. M. Nadler, " 'Empyrean', an Alternative Paradigm for Pattern Recognition," Pattern Recog., vol 1, pp 147-163, 1968.
56. M. Nadler, Image Scanning Method and Device, US Patent #4418362, 29 Nov 1984.
57. M. Nadler, Process and Device for the Automatic Segmentation of a Scanned Image in an Image Pattern Recognition System, US Patent #4504971, 12 March 1985.
58. M. Nadler, Process and Device for the Binarization of a Pattern, US Patent #4509195, 2 April 1985.
59. Yoh-Ham Pao, Adaptive Pattern Recognition and Neural Networks, Addison-Wesley, N.Y, 1989.
60. A. Kandel, Fuzzy Mathematical Techniques with Applications, Addison-Wesley, N.Y, 1986.
61. J. Hopcroft, "An  $n \log n$  Algorithm for Minimizing States in a Finite Automaton," Technical Report, Stanford University, 1970.
62. A. M. Landraud, J. F. Avril, P. Chretienne, "An Algorithm for Finding a Common Structure Shared by a Family of Strings," IEEE Trans. on PAMI, Vol 11, no 8, pp 890-895, Aug 1989.

## VITA

Nikos D. Asimopoulos was born in Kozani, Greece, April 23, 1957. He attended the Aristotelian University of Thessaloniki, and graduated in 1980 receiving his Diploma in Electrical Engineering. In the Fall of 1981 he came to Virginia Polytechnic Institute and State University to pursue graduate studies. He received the M.Sc degree in Electrical Engineering in July 1983 and continued his studies towards a Ph.D. degree in Electrical Engineering. During his graduate studies he was engaged in several research programs in the areas of digital image processing and pattern recognition.