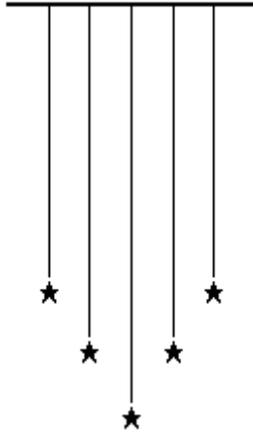
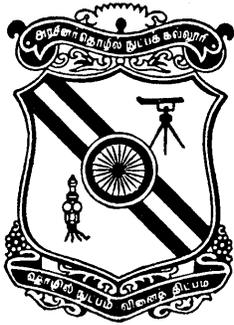


**DESTINATION ADDRESS INTERPRETATION FOR
AUTOMATING THE SORTING PROCESS OF
INDIAN POSTAL SYSTEM**

PROJECT WORK

SUBMITTED IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE AWARD OF THE DEGREE OF
BACHELOR OF ENGINEERING
(ELECTRONICS AND COMMUNICATION ENGINEERING)



SUBMITTED BY

BALAJI S

SRI RAMA PRASANNA P

THEJAVOR HARALU KHEZHIE

GUIDED BY

2002 - 2003

MRS. C. VASANTHANAYAKI M . E.

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
GOVERNMENT COLLEGE OF TECHNOLOGY
COIMBATORE - 641 013**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
GOVERNMENT COLLEGE OF TECHNOLOGY**

COIMBATORE – 641 013



BONAFIDE CERTIFICATE

**This is to certify that the project titled
“Destination Address Interpretation For Automating The Sorting
Process Of Indian Postal System”**

Submitted by

NAME	Reg. No.
BALAJI S	9914103
SRI RAMA PRASANNA P	9914143
THEJAVOR HARALU KHEZHIE	9914150

**in partial fulfilment of the requirements for the award of the degree of
Bachelor of Engineering in Electronics and Communication is a bonafide record of
the work done by them during the year 2002-2003.**

.....

Place : Coimbatore

Guide

Date :

(Mrs. C. VASANTHANAYAKI)

Submitted for the project viva-voce examination held on _____

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We wish to thank our honourable Principal Dr. S. Arumugam who has been a constant source of encouragement for all the students.

We sincerely express our deep sense of gratitude and profound thanks to our Head of the Department, Prof. V. Lakshmi Prabha for the unflinching interest she has shown in our success.

With immense pleasure, we express our heartfelt thanks to our beloved guide, Mrs. C. Vasanthanayaki, for her constant encouragement, expert guidance, creative criticisms and timely help to successfully complete this project.

Finally, we wish to extend our thanks to Mr. K.D. Vishwanathan, Retd. Postmaster, Mr. Chinnadurai, PMG's Office, R.S. Puram and the staff and students of National Institute of Technology Surathkal, Thiagarajar college of engineering Madurai and Govt. College of Engineering Salem for encouraging us with their valuable suggestions.

SYNOPSIS

While most of the Indian industries are in the process of automation, it is a bitter fact that the Indian Postal System is still using manual intervention for its mail sorting and processing. In this project, we intend to propose an Automated Postal System which would reduce the mail sorting time besides ruling out human errors. The Automatic Mail Processor, which we have designed, scans a mail and interprets the imperative fields of the destination address such as the Pin Code, City name, Locality name and the Street name. The interpreted address is subsequently converted into a Delivery Point Code, which is a 12 digit number. The Delivery Point Code is printed on to the mail in the form of a barcode which can be read by a low-cost machine. By converting the destination address into a barcode (which is printed on the mail), all of the future sorting processes can be accomplished by using a mechanical machine sorter, that can sort the mails according to the barcode present on them.

The AMP comprises of five modules namely Pre-Processing unit, Address Block Location unit, Line and Word separation Unit, Address Parsing and Recognition Unit and Delivery Point Code generation unit. The Address Parser detects the location of the fields by using a heuristic procedure. The recognition system is accomplished by representing the characters in the form of chain codes, and by using their Fourier Descriptors for alphanumeric matching with the aid of a Neural Network. The Delivery Point Code is then generated by using the Pin Code (6 digits) and the Street Code. A database is used in this step to assign 5 digit codes for street names. The DPC is then bar-coded and printed on to the mail.

CONTENTS

CHAPTER 1	Introduction	1
	1.1 Implementation Considerations	
	1.2 The Chapters Ahead	
CHAPTER 2	Automatic Mail Processor	3
	2.1 Modules of AMP	
	2.2 Block Diagram	
CHAPTER 3	Pre – Processing	6
	3.1 Digitization	
	3.2 Binarization	
CHAPTER 4	Address Block Location	8
	4.1 Redundancy Correction (RC) Algorithm	
	4.2 Flowchart of RC algorithm	
	4.3 Implementation of RC Algorithm	
CHAPTER 5	Line and Word Separation	13
	5.1 Line Separation Logic and Constraints	
	5.2 Horizontal Scanning (HS) Algorithm	
	5.3 Word Separation Logic	
	5.4 Vertical Scanning (VS) Algorithm	
	5.5 Flowcharts for HS and VS Algorithms	
CHAPTER 6	Address Parsing and Recognition	19
	6.1 Character Separation	
	6.2 Recognition of Characters and Numbers	
	6.3 Pre-Processing the Character image	
	6.4 Contour Detection	

	6.5 Chaincoding	
	6.6 DFT Components	
	6.7 Neural Network for Character Matching	
	6.8 A Final Note on Address Parsing and Recognition	
CHAPTER 7	Delivery Point Code Generation	34
	7.1 Control Digit Combinations	
	7.2 Real-Time Implementation	
	7.3 Universal Networking and Hierarchical Routing	
CHAPTER 8	Bar-Coding	38
	Implementation Results	41
	Conclusion and Future Enhancements	49
	Appendix (MATLAB Codes)	50
	References	104

LIST OF FIGURES

- Figure 1 : Block diagram
- Figure 2 : Digitized image
- Figure 3 : De-noised and Binarized image
- Figure 4 : RC Algorithm
- Figure 5 : Low Pass filtered image
- Figure 6 : Re-binarized image
- Figure 7 : Redundancy Corrected image
- Figure 8 : Extracted Address
- Figure 9 : Line separation
- Figure 10: Underline Removal
- Figure 11: A line of the extracted address
- Figure 12: a) Horizontal Scanning
- Figure 12: b) Vertical Scanning
- Figure 13: Word Separation
- Figure 14: Character Separation
- Figure 15: Contour Detection
- Figure 16: Chain Coding
- Figure 17: Chain Coding Logic
- Figure 18: The Back Propagation Neural Network
- Figure 19: 177 Character Patterns Used for Matching
- Figure 20: 20 Components for the Character 'a'
- Figure 21: Neural Network Output
- Figure 22: Delivery Point Code Format
- Figure 23: An Address Format
- Figure 24: Barcode Format
- Figure 25: Barcode for DPC = 641013000001
- Figure 26: Universal Networking Method
- Figure 27: Hierarchical Routing Method
- Figure 28: Inland Letter and Post Card Samples
- Figure 29: Front End demonstration of the implementation in MATLAB

- Figure 30: Lowpass Filtering Output
- Figure 31: Re-Binarization Output
- Figure 32: Redundancy Correction Output
- Figure 33: Address Block Location Output
- Figure 34: Underline Removal Output
- Figure 35: Line Separation Output
- Figure 36: Word Separation output
- Figure 37: Character Separation output
- Figure 38: Recognition Output
- Figure 39: Database Creation Output
- Figure 40: Samples Used for Testing
- Figure 41: DPC and Barcoding Output

LIST OF TABLES

- Table 1: CTRL combinations

1. Introduction

The Indian Postal System (IPS) is one among the few public sector units which deserve to be fully automated. The time taken for a mail delivery can be viewed as the sum of the transit time and processing time. The transit time is something which is inevitable. But, the processing time, which includes the time for mail address interpretation and sorting, can be reduced to a great deal if the sorting process is automated. Consider a mail that is posted from Coimbatore to Shimla. The existing postal system involves human intervention in the processing of this mail at least four times: Coimbatore, Chennai, Delhi and Shimla. At each of these points, a human reads the mail and manually interprets it before sorting it according to its destination address. This process of mail sorting is highly time consuming besides being error prone. In this project we intend to propose an Automatic Mail Processor (AMP) which scans and interprets the destination address and converts it into a Delivery Point Code, which is printed on the mail in the form of a bar-code. Now that the destination address is in the form of a barcode, it can be interpreted by using a low-cost barcode reader for future sorting processes.

1.1 Implementation Considerations

The Automatic Mail Processor (AMP) is designed to be compatible with all kinds of mails such as Inland letters, Post Cards, Post Covers, and Envelopes etc. The job of AMP is simplified in the first three cases due to the presence of predefined address locations. In the case of an Envelope, there is an additional work of locating the position of the destination address. Let us consider such an Envelope with **NO** predefined grids as

an example. All the stages in this project are described with Envelope as an instance. The same can be implemented with much ease for all other types of mails due to their inherent simplicities.

It should also be ensured that sufficient space is provided in each mail for printing the bar-code. Though space would not be a constraint in large Envelopes, sufficient care has to be taken in the case of small Post Cards and Covers.

1.2 The Chapters Ahead

Chapter 2 gives the various modules of the Automatic Mail Processor along with the general block diagram. Chapter 3 discusses the Pre-Processing stages performed on the scanned mail before going on to further modules. Chapter 4 concentrates on the Address Block Location module and explains the various steps taken to locate the destination address.

The destination address is extracted and further divided into lines and words as explained in Chapter 5. Chapter 6 explains the way in which the words are separated into characters and also the recognition process undertaken in this project. The process of generating the Delivery Point Code is explained in Chapter 7 and the Chapter 8 describes the procedure of converting the Delivery Point Code into a Barcode.

2. Automatic Mail Processor

2.1 Modules of AMP

The five main modules of Automatic Mail Processor (AMP) are

1. Pre-Processing unit
 - a) Digitization
 - b) De-noising
 - c) Binarization
2. Address Block Location (ABL) unit
 - a) Image Averaging
 - b) Binarization
 - c) Redundancy Correction (RC)
 - d) Address Extraction
3. Line and Word Separation unit
 - a) Horizontal Scanning (HS)
 - b) Vertical Scanning (VS)
4. Address Parsing and Recognition unit
 - a) Parser – Field Identification
 - b) Pin Code Recognition
 - c) City, Locality Recognition
 - d) Consistency Checker
 - e) Street Recognition
5. Delivery Point Code (DPC) generation unit
 - a) Add-On search
 - b) DPC creation
 - c) Bar-Coding

2.2 Block Diagram

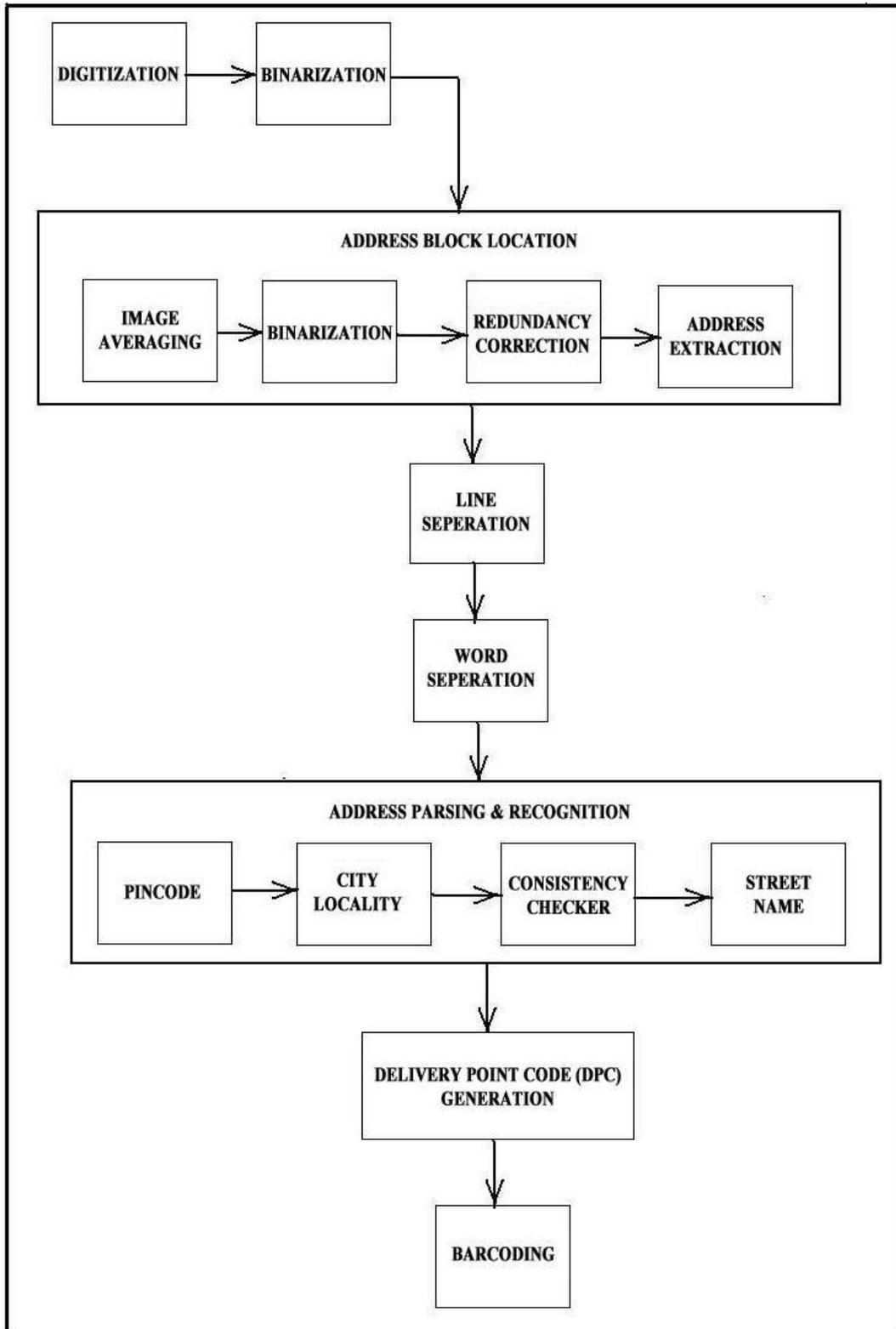


Figure 1: Block Diagram

The Block Diagram of AMP begins with a Pre-Processing unit which comprises of two stages namely Digitization and Binarization. Denoising can be considered to be a part of the Digitization stage. The Address Block Location module determines the location of destination address in the mail. The Line and Word separation units divide the extracted address into constituent lines and words by employing Horizontal and Vertical Scanning algorithms.

The Address Parsing and Recognition module parses and recognizes the imperative fields of the destination address. From the recognized fields, the Delivery Point Code (DPC) is generated by the DPC unit. The DPC is subsequently converted into a barcode that will be printed on the mail.

3. Pre-Processing

3.1 Digitization

The first step of AMP is to obtain a digitized image of the Envelope. Digitization is performed by scanning the Envelope with a high-resolution scanner to produce an 8-bit gray scaled image as shown below.

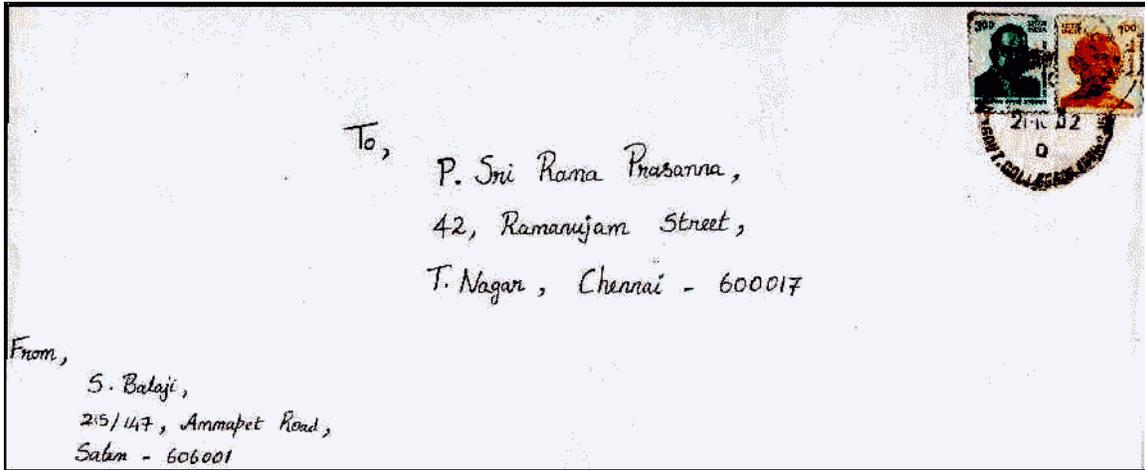


Figure 2: Digitized image

3.2 Binarization

The digitized image usually contains certain noisy spurious pixels which can be removed by de-noising the image. The image de-noising is performed by subjecting the image to an averaging filter.

The mask is suitably set so as not to blur the image. A 3 X 3 mask would be sufficient to remove the discrete spurious noisy pixels from the scanned image of the mail. Median filtering may also be employed for this purpose.

Every pixel of the averaged image is represented by 8-bits. But, for interpreting the destination address, a binary image with two colours is sufficient.

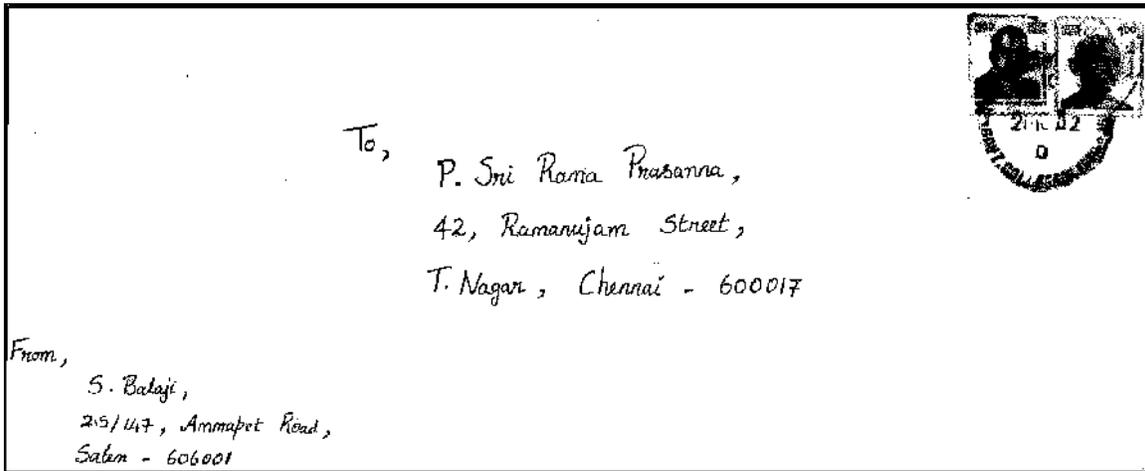


Figure 3: De-noised and Binarized image

Therefore, the averaged image is binarized by setting a threshold level, which is determined adaptively depending on the foreground and background colours. The binarized image is shown in Figure 3.

4. Address Block Location

The location of the destination address in an envelope is not pre-determined. This necessitates the need for an Address Block Location module (ABL). The ABL takes the binarized image from the preprocessor and determines the position of the destination address in the envelope. In other words, the ABL returns the coordinates of a rectangle with least area that can be drawn around the destination address after leaving a tolerance level. As the ABL distorts the original image while processing, a backup of the original image is stored for future retrieval.

The logic behind the working of ABL is the Redundancy Correction (RC) algorithm. The algorithm functions in such a way as to remove the explicit redundancies in the image such as the sender's address and stamp impressions. The steps of this algorithm are illustrated below.

4.1 Redundancy Correction (RC) Algorithm:

1. Obtain the image from preprocessing unit
2. Low pass filter the image using a 20 X 20 spatial mask using point processing techniques.
3. As the low-pass filtered image is a non-binary image, re-binarize it by setting a high threshold value.
4. The Re-Binarization would cause the fields on the envelope to form black patches.

5. As the redundant information such as the sender's address and the stamp impressions are usually present only in the corner of an envelope, they can be removed by scanning through the image.
6. The resultant image now has a white back ground with back patches only in the destination address.
7. After leaving a tolerance, a rectangle is suitably positioned over the destination address patch and its coordinates are returned.

4.2 Flowchart of RC algorithm:

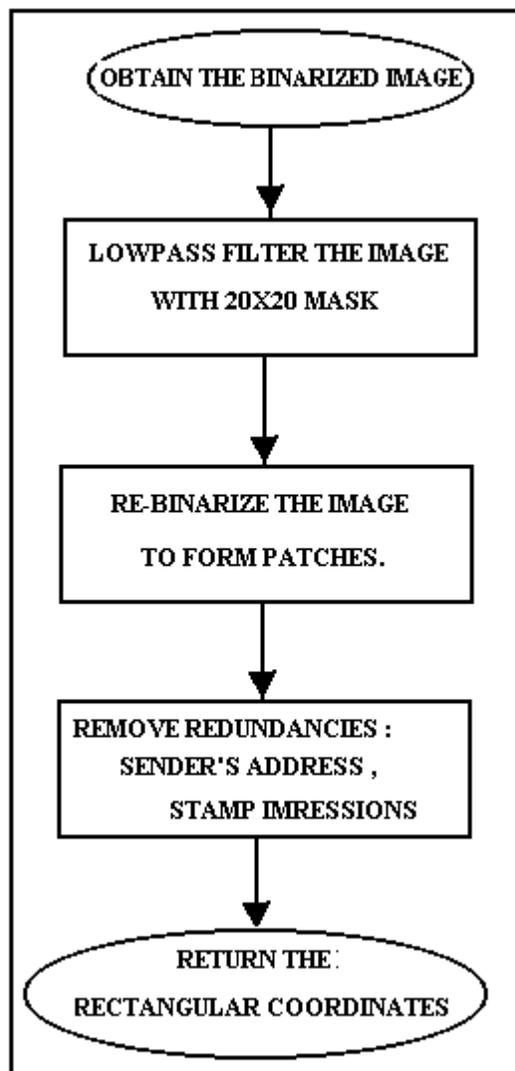


Figure 4: RC Algorithm

4.3 Implementation of RC Algorithm

The original image in Figure 3 is Low Pass Filtered using a 20 x 20 mask spatial mask. The spatial mask is nothing but a matrix with 20 rows and 20 columns with all of its elements set to 1. A 20 x 20 mask is chosen as it provides sufficient blurring effect which can be converted into patches by re-binarization. The Low Pass Filtering is accomplished by rolling the 20 x 20 mask through each pixel of the original image and determining its response. The value of the pixel is subsequently transformed into the response value.

$$R = 0.0025 * (x1 + x2 + x3 + x4..... + x17 + x18 + x19 + x20)$$

The equation for computing the response by point processing is given above. In this equation x_1, x_2, \dots, x_{20} represent the value of the pixels of the original image over which the 20 x 20 spatial mask is placed.



Figure 5: Low Pass Filtered image

It can be clearly observed from the image that the Low Pass Filtering has blurred it sufficiently for patch formation. Another important point to be noted here is that the size of Figure-5 is around eight times the size of Figure-3 as 8 bits are required representing each pixel of Figure-5.

The next step is to re-binarize the Low Pass Filtered image (Figure 5) to obtain an image with patches. The degree of patch formation can be adjusted by varying the threshold level during binarization.



Figure 6: Re-Binarized image

In our application, it is desirable to form continuous patches so that all the words are joined together. Consequently, a high threshold value of the order of (0.8 to 1.0) is set during binarization .The above Binary image (Figure 6) is obtained by using a threshold of 1.0

The Redundancy Corrected image (Figure 7) is obtained from Figure 6 by removing the unwanted information in the corners and boundaries of the image. As it is conventional that the Destination address is written somewhere in the centre and certainly not in the corners, such logic has been employed.

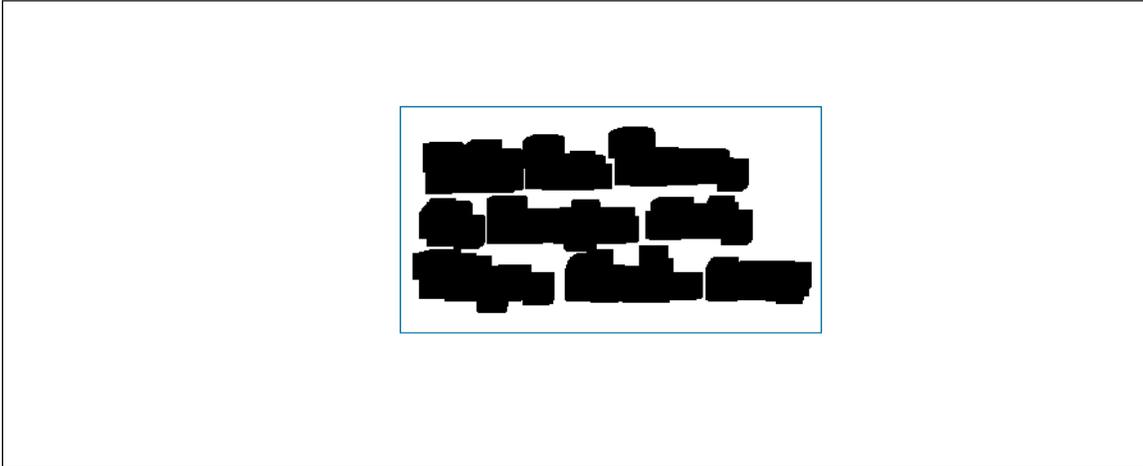
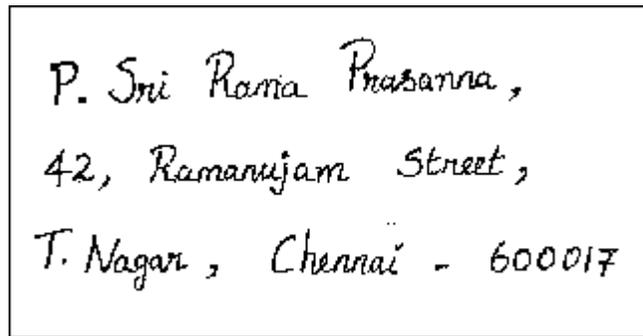


Figure 7: Redundancy Corrected image

After redundancy correction, the image is left only with the patches of the destination address. By horizontal scanning from the first pixel row, the location of the Destination address can be accurately identified. A tolerance level is adopted and a rectangle is drawn over the destination address. The coordinates of this rectangle is returned to the Line and Word Separation module.

5. Line and Word Separation

As stated earlier a copy of the original image (Figure 3) is stored separately as the ABL module distorts the image given to it. The Coordinates returned by the ABL module is used to extract the destination address from the saved copy of Figure 3. The extracted destination address is shown below.



P. Sni Rama Prasanna,
42, Ramanujam Street,
T. Nagar, Chennai - 600017

Figure 8: Extracted Address

5.1 Line Separation Logic and Constraints

The line separation procedure consists of scanning the image (Figure 8) row by row. The row in the preceding line represents the pixel row and not the lines of the address. The simplest way of separating the lines is to set a threshold value for the number of white pixel rows between two address lines. Two lines are separated if the number of white pixel rows between them is greater than the threshold value.

Such a logic would be futile when letters such as 'y', 'g' etc occur in the first line and letters like 'f', 'd', etc occur in the second line without having white pixel rows in-between. Such a bottle neck can be averted by designing the algorithm in such a way that

it is tolerant to a certain minimum number of black pixels in a white row. In other words, a row with very few black pixels can be considered as a white pixel row. Such an algorithm will be fault tolerant.

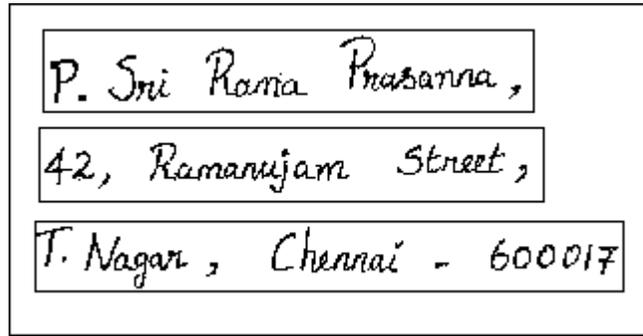


Figure 9: Line separation.

5.2 Underline Removal Technique

In many cases, the words in handwritten addresses are written on printed guide lines that are provided to assist the writer. Often handwritten text intersects these guide lines. The Underline removal algorithm designed ensures that the underlines are removed without tampering the image. It scans through the image horizontally and detects the presence of continuous lines.

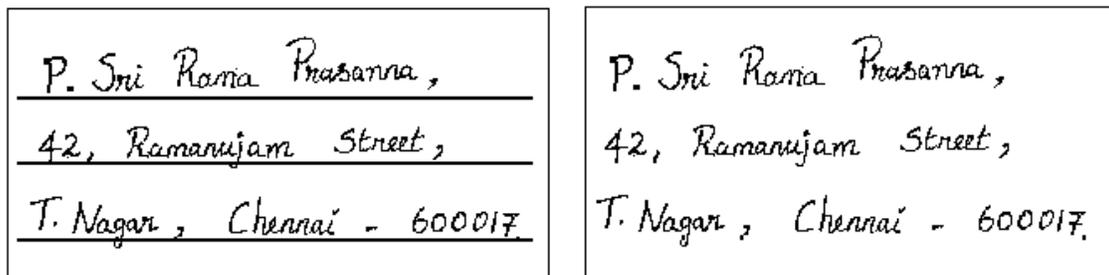


Figure 10: Underline Removal

While removing the lines, it takes into consideration the presence of black pixels in the row, above the row taken into consideration. If there happens to be a black pixel,

then the current pixel is not converted into white and if there is no black pixel in the previous row then the current black pixel is converted into white. Thus the destination address is not affected much by removing the underlines.

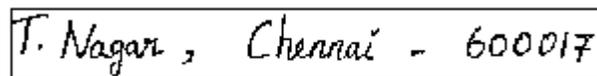
5.3 Horizontal Scanning Algorithm

The Horizontal scanning algorithm that is used for line separation is described below.

1. Scan the extracted image (Figure 8) pixel-row by pixel-row
2. Set a threshold value for the minimum number of white pixel rows to be present between two address lines.
3. A predefined amount of tolerance is provided while deciding whether a pixel-row is a white pixel-row for avoiding 'y' – 'f' problem.
4. Count the number of consecutive white pixel rows.
5. If the number of white pixel rows is greater than the threshold, then separate the two address lines.
6. Repeat steps 4 and 5 until all the address lines have been separated.

5.4 Word Separation Logic

The word separation is performed analogous to the line separation. But, instead of horizontal scanning, vertical scanning is employed here.



T. Nagar, Chennai - 600017

Figure 11: A line of the extracted address

It is very important to set a proper threshold value for horizontal and vertical scanning methods. An optimum threshold value is decided only after performing a statistical analysis of human handwriting.

5.5 Vertical Scanning Algorithm

The vertical scanning algorithm is described below.

1. Scan the extracted line (Figure 11) column by column.
2. Set a threshold value for the minimum number of white pixel columns to be present between two words.
3. Count the number of consecutive white pixel columns.
4. If the number of white pixel columns is greater than the threshold, then separate the two words.
5. Repeat steps 3 and 4 until all the words have been separated.

5.6 Flow Charts for Horizontal and Vertical Scanning

The Flowchart given for the line separation using horizontal scanning algorithm will work perfectly only when the address is written in a horizontal manner. The line separation technique adopted for non-horizontal addresses is described later in this section.

The tolerance value EL has to be set for avoiding high precision errors. In other words, while scanning a horizontal pixel row, a white row need not mean that all the pixels in it are white. A maximum of EL number of pixels can be black in colour. By

using such a tolerance, even if few characters of two address rows cross a same pixel row, line separation algorithm would work efficiently.

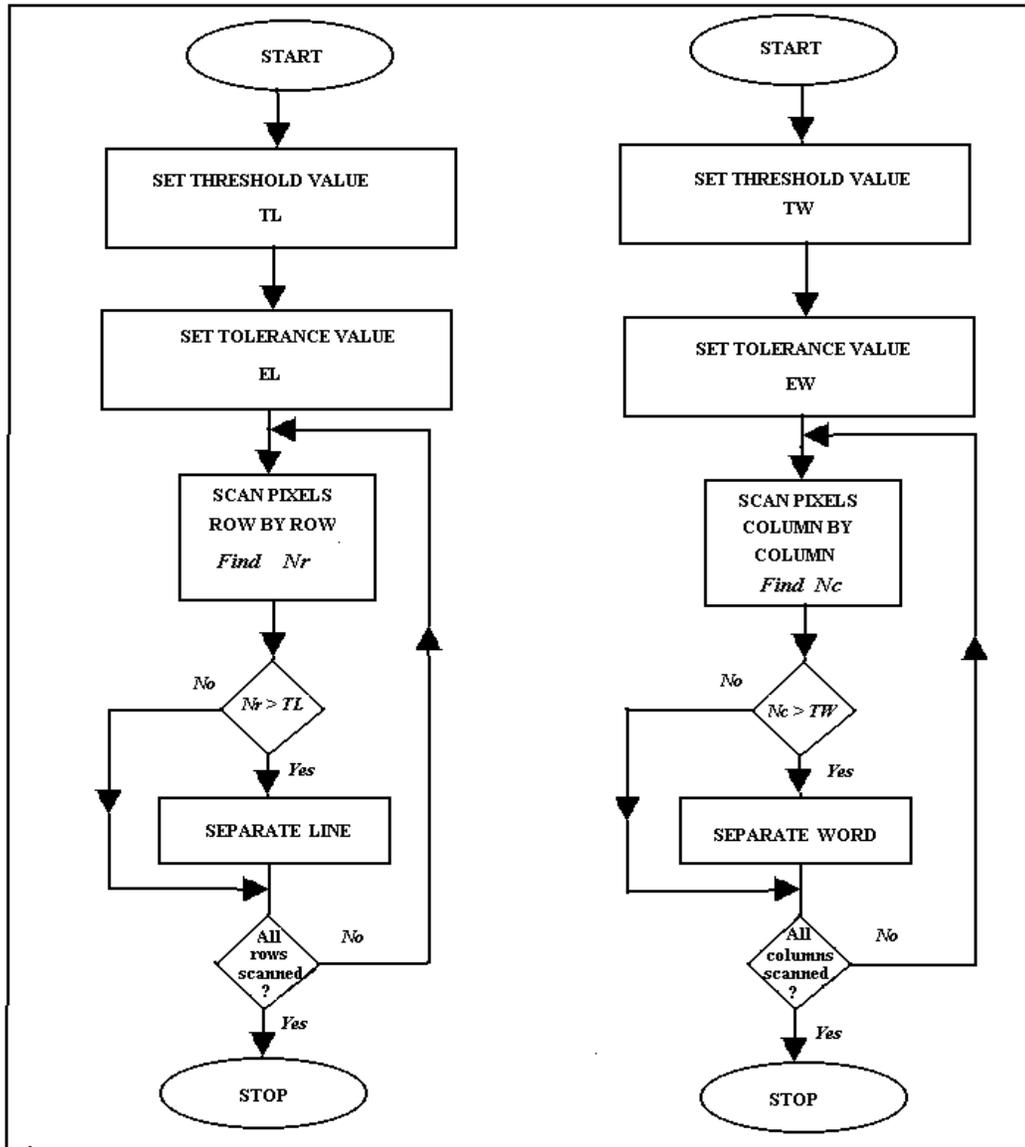


Figure 12: a) Horizontal Scanning

b) Vertical Scanning

The VS algorithm separates the words of Figure 11 as illustrated below.

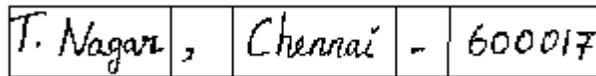


Figure 13: Word Separation

It has to be carefully noted that even the unwanted fields such as ',' and '-' are considered as separate words by the VS algorithm and they are also sent to the address parsing and recognition module where they are detected and removed.

If the destination address is written in a non-horizontal manner, then the address lines must be made horizontal by rotation before the horizontal scanning is performed. Yet another method for tackling this problem is to undertake a sloped scanning method. The slope of scanning should be equal to the slope with which the address line has been written.

6. Address Parsing and Recognition

The input to APR is the series of words of Figure 13 and the objective of APR is to parse and recognize these word ‘images’. Address parsing is nothing but identifying the Pin Code, City, Locality etc from the word ‘images’ that are provided to APR. This is done partly on the basis of convention and partly on the type of the strings. For instance, if the word image happens to be a 6-digit number, and then there is a high possibility for it to be the Pin Code.

Conventionally, the name of the City and Pin Code is written only in the last line of the address. Though the above cases may not be strictly followed, it would be beneficial in terms of speed of processing to make such wise assumptions. If a particular mail does not follow conventions, then the APR is forced to interpret and identify (parse) each field before recognition. The address parsing commences from the bottom most line of the destination address. The words of the bottom most lines are recognized and are compared with databases of countries, States of India, Districts in India and Localities. As such a system is employed; the parsing won’t fail even if additional details like Name of the State are written in the destination address.

6.1 Character Separation

Before sending the word images directly to recognition unit, they should be split up into component characters or digits. The basic idea here is to identify the individual characters and ligatures.

Ligatures are the small horizontal lines which connect two characters in a word. A ligature can also be vertical/slanted as in the case of a transition from ‘g’ to ‘f’. The division of a word without ligatures can be done easily by detecting the blank white columns between two characters. A word with ligatures can be divided by carefully specifying the ‘cut’ points at the two ends of ligatures.

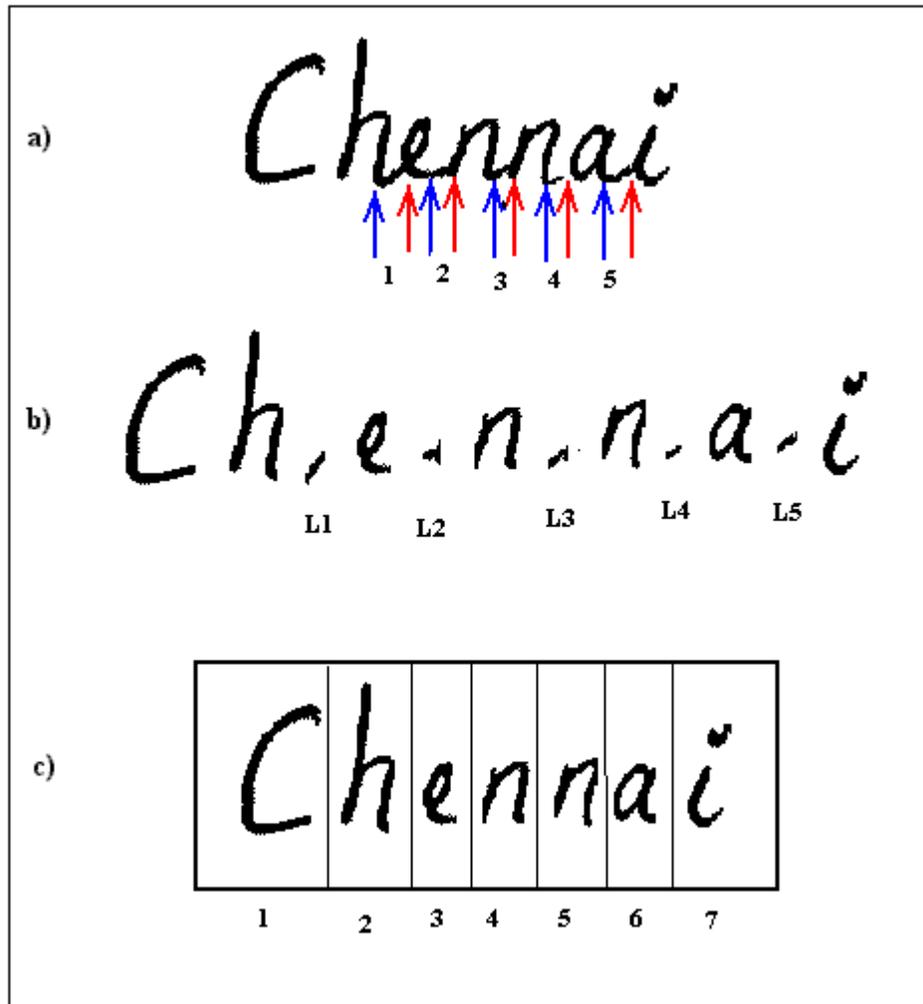


Figure 14: Character Separation

The above figure shows how the word image “Chennai” is separated into characters and ligatures. Statistics of hand-written recognition claims that ligatures are

invariably straight lines with constant slope. For general alpha-numeric recognition algorithms, it would be beneficial to consider ligatures as lines between two steep risings. Such a system is adopted by the APR to detect lines with fewer slopes ($\tan\Theta$) that are between two steep curves and these entities are removed. The characters are individually separated and they are passed to the recognition unit as shown in Figure 14(c).

6.2 Recognition of Characters and Numbers

The recognition unit uses 4 connectivity for back-ground and 8 connectivity for foreground. The key steps to be performed in recognition are enumerated below.

1. Normalizing the character image.
2. Boundary detection.
3. Extracting the chain code of the boundary.
4. DFT Computation
5. Feeding the DFT components into a trained Neural Network
6. Identifying the Character on the basis of the best match.

6.3 Pre-Processing the Character image

For Fault-Tolerant character recognition, it is imperative to avoid the size effects of the character images. Similarly, slant correction should also be undertaken to straighten up the characters. The slant correction is performed by measuring the average slant angle of the characters of the word and by tilting the character image in opposite direction

through the same angle. The slant corrected character image is normalized to size of 128 x 128 Sq pixels.

6.4 Contour Detection

The boundary detection is one in which the contours of the character image are detected. Any standard edge detection algorithm can be used for this purpose. But for the sake of accuracy, contour detection by point processing is undertaken. The detection logic is described below.



Figure 15: Contour Detection

Consider a character image with a black background and a white fore ground. The detection procedure encompasses a horizontal scanning technique. Every pixel of the character image is scanned horizontally. If the pixel under consideration is white, then the colours of its four connected pixels are taken into account.

If all of the four connected pixels are white, then the pixel under consideration is not a part of the contour. On the other hand, even if one of the four connected pixels are not white, then it can be concluded that the pixel under consideration is a part of the contour.

6.5 Chain Coding

The next step of Recognition unit is to extract the chain codes from the boundary of character image. Chain coding of outer contour alone is enough for recognizing alpha numeric characters with reasonably high degree of precision. While traversing through the outer contour, care is taken so as to avert the influence of noise. If a loop is formed due to noise, the algorithm turns backward and proceeds until a correct path is found. In Figure16 chain coding of the character 'C' is illustrated. It can be observed that chain codes are recursive in nature as the outer contour is itself a loop. The tracing ends when the initial point is reached again. Similarly, chain codes for all the other characters can be formed. A single character can have numerous 8-directional chain codes depending on its starting point. Conventionally, the chain code with the least magnitude is taken into consideration.

The Figure 16 shows the boundary detection of 'C' and the 8 – directional chain code convention. The starting point for determining the chaincode can be in any part of the outer contour. For the sake of simplicity, the starting point is considered to be the topmost pixel in the contour. Then, by proceeding in the counter clockwise direction, the direction of the next pixel in the contour is determined. The directivity determination is implemented as a series of eight functions in Matlab, as eight directional chaincode is

used. The following paragraph explains how exactly the direction of traversal is determined.

The starting point of the contour is determined as the topmost pixel in the contour. The topmost pixel can be identified by performing a horizontal scanning from the first row. As it has been already stated that the chaincoding has to proceed in a counter clockwise direction, there are only two directions in which the chain coding can proceed after the first pixel.

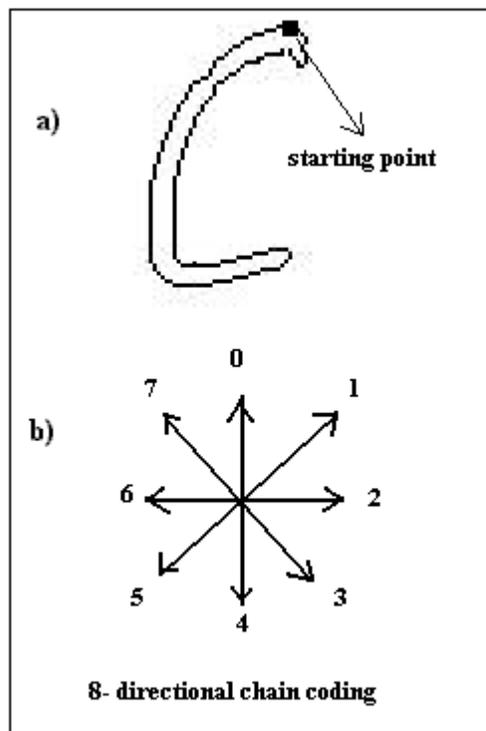


Figure 16: Chain Coding

This concept can be better understood by observing the Figure 17. The pixel arrangement is shown in Figure 17(a). The pixel 'P' represents the pixel under consideration. As it can be seen, there are eight pixels surrounding it. Coming back as to

why there are only two directions for the chaincode to proceed, it has to be noted that when pixel 'P' is under consideration, the pixels 1, 2, 3 and 4 would have already been scanned. As we are proceeding in the counter clockwise direction, we cannot consider pixel 5. A careful analysis of Figure 17 with all the character contours would reveal that there is no way what pixel 8 can be white when 'P' is the starting pixel. So, we are now left with only two pixels 6 and 7. From Figure 16, there directivities can be found to be 5 and 4 respectively. By explicitly proceeding in the directions 4 or 5, the next pixel in the contour can be identified.

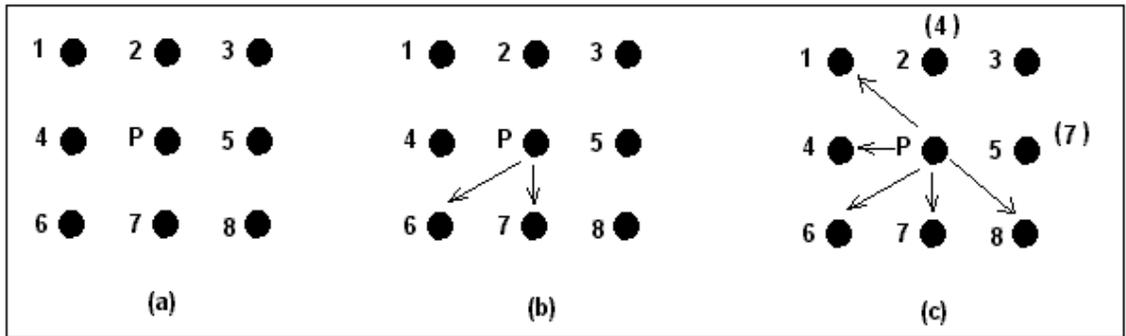


Figure 17: Chaincoding Logic

The coordinates of the starting pixel and the next pixel in the contour form the first two rows of the chaincode matrix with 'n' rows and 2 columns. The number 'n' denotes the number of pixels in the contour. Unlike the case for the second pixel in the contour, for all other pixels, there are five directions in which the chaincode can proceed.

In Figure 17(b) consider that the pixel 6 is white which implies that the chaincode moves in direction 5. The Figure 17(c) has been structured assuming such a possibility. The old pixel numbers are depicted within brackets. The pixel 3 in Figure 17(c) is the

pixel 'P' in Figure 17(b). So, In Figure 17(c) it can be rightly said that the chaincoding has proceeded from pixel 3 to pixel 'P'.

Now, to determine the next pixel direction in the chaincode, it is enough if we scan through five pixels namely 1, 4, 6, 7, 8 as shown in Figure 17(c). It is only in these directions the chaincoding can advance. The other three pixels 2, 3 and 5 need not be taken into account as the chaincode came to pixel 'P' from 3.

Similarly, for all other directions also, it is sufficient to consider only 5 pixels instead of 8. This chaincoding proceeds until the starting pixel is reached after making a full traversal through the outer contour. After detecting every successive pixel in the contour, its coordinates have to be appended to the chaincode matrix.

6.6 DFT Components

The Fourier descriptors (FDs) of the character are computed by applying DFT on the chain code points. DFT is applied in such a way that the x-coordinate $x(k)$ of the chain code is considered as the real part and the y-coordinate $y(k)$ is considered as the imaginary part.

$$S(k) = x(k) + jy(k)$$

$$a(u) = \frac{1}{N} \sum_{k=0}^{N-1} S(k) \exp[-j2\pi uk/N]$$

The complex coefficients $a(u)$ for $u = 0, 1, 2, 3, 4, 5, \dots$ are called the Fourier descriptors of the boundary. The number of DFT components obtained depends on the size of the contour (i.e.) the number of pixels in the contour.

For identifying a character, all the DFT components are not required. A satisfactory description can be obtained by choosing few lower frequency and few higher frequency components. Larger the number of components considered, higher is the accuracy. But, as the number of components considered increases, computational complexity is observed. So, a tradeoff has to be made between accuracy and complexity while choosing the number of DFT components considered for recognition. In this project, 10 low frequency components and 10 high frequency components are considered for each character. To summarize, every character will be represented by a set of 20 DFT components.

6.7 Neural Network for Character Matching

Before character matching can be performed, matching tables containing the Fourier descriptors of all the alpha-numeric characters are drafted. A Back Propagation Neural Network is used for identifying the input character pattern.

The BPN learns a predefined set of input-output example pairs by using a two phase propagate-adapt cycle. After an input pattern has been applied as a stimulus to the first layer of network units, it is propagated through each upper layer until an output is generated. This output pattern is then compared to the desired output, and an error is computed for each output unit.

The error signals are then transmitted backward from the output layer to each node in the intermediate layer that contributes directly to the output. However, each unit in the intermediate layer receives only a portion of the total error signal, based roughly on the relative contribution the unit made to the original output.

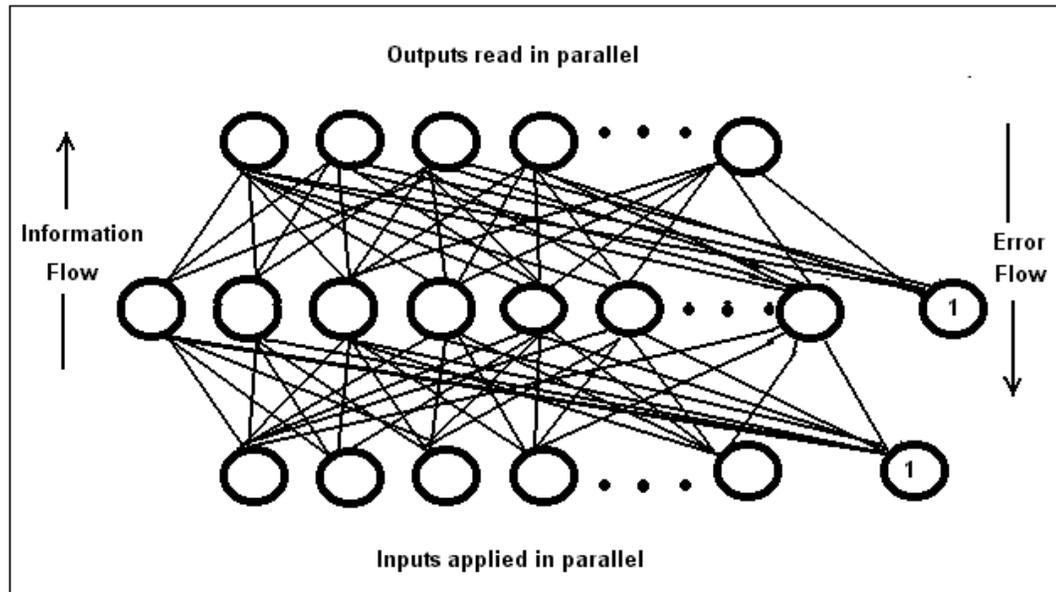


Figure 18: The Back Propagation Neural Network

This process repeats, layer by layer, until each node in the network has received an error signal that describes the relative contribution to the total error. Based on the error signal received, connection weights are then updated by each unit to cause the network to converge toward a state that allows all the training patterns to be encoded.

The significance of this process is that, as the network trains, the nodes in the intermediate layers organize themselves such that different nodes learn to recognize different features of the total input space. After training, when presented with an arbitrary input pattern that is noisy or incomplete, the units in the hidden layers of the network will

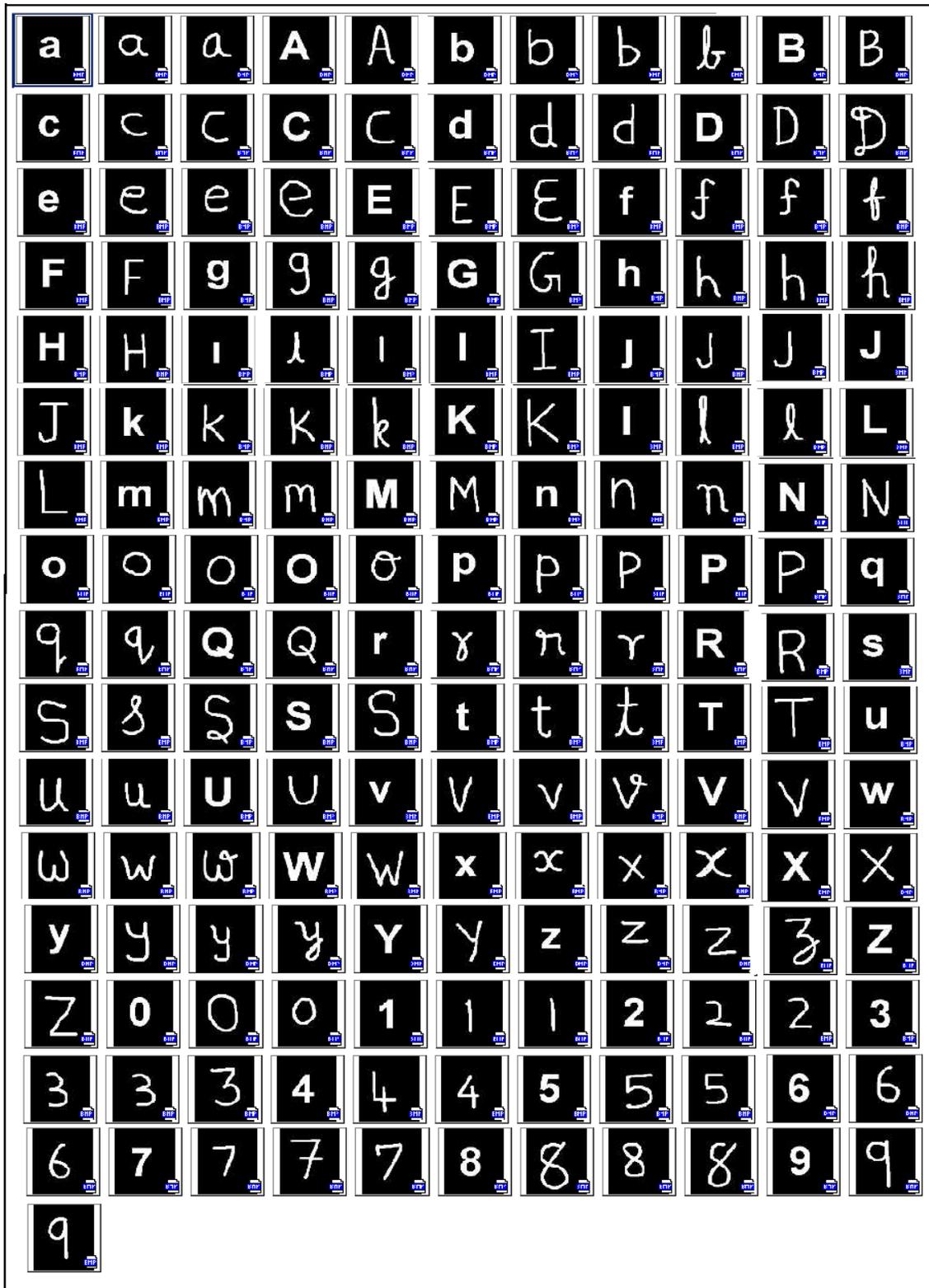


Figure 19: 177 Character Patterns used for training

respond with an active output if the new input contains a pattern that resembles the feature individual units learned to recognize during training.

The neural network is trained with 177 character patterns shown in Figure 19. The 177 character patterns are formed by selecting all the different forms of representing the 26 alphabets and 10 numbers. The structural configuration of the neural network used and the complexities involved are discussed in the following lines.

```
a1 =  
  
1.0e+004 *  
  
1.1288 + 1.1091i  
-0.0300 - 0.0299i  
-0.0584 - 0.0025i  
0.0197 + 0.0503i  
0.0142 + 0.0145i  
0.0005 - 0.0192i  
-0.0015 - 0.0059i  
-0.0046 + 0.0063i  
0.0037 + 0.0017i  
0.0030 - 0.0040i  
0.0007 - 0.0004i  
-0.0001 + 0.0069i  
0.0044 - 0.0023i  
0.0116 - 0.0006i  
0.0100 + 0.0074i  
-0.0039 - 0.0166i  
0.0101 + 0.0031i  
-0.0233 - 0.0631i  
-0.1512 - 0.0090i  
0.0927 - 0.2816i
```

Figure 20: 20 complex components for the character 'a'

The inputs to be fed to the Neural Network are DFT components which are complex numbers. As, it is not possible to feed a complex quantity directly to an input node, the complex DFT components have to be resolved further into Polar or Rectangular form. The Rectangular form of representing each complex component by its real and imaginary part is used in this project. If Polar form is used, each complex number is

represented in terms of its magnitude and angle. In both of these two cases, there is a need to represent a complex component by two entities. As 20 complex components are used for representing each character, the total number of input nodes required turns out to be 40. As 177 character patterns are used, the number of output nodes is obviously equal to 177.

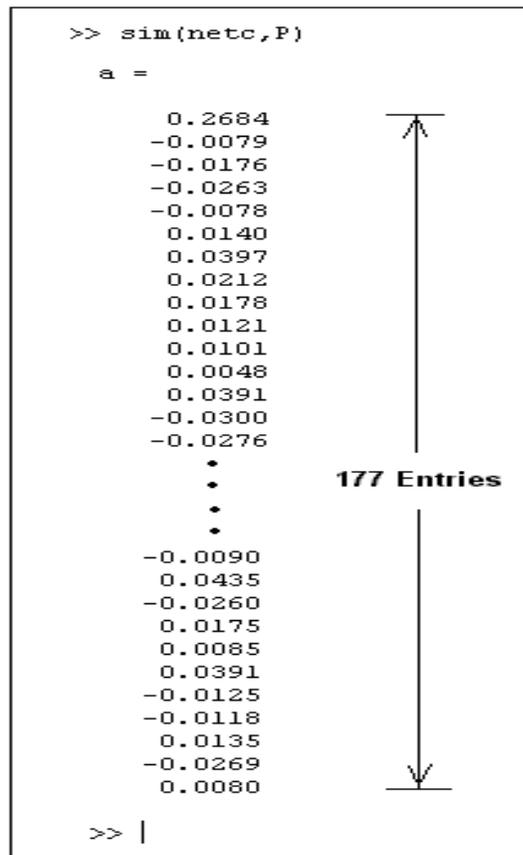


Figure 21: Neural Network Output

Deciding on the number of hidden layers and number of nodes in each layer is a very significant issue. As the number of hidden nodes is increased, the accuracy of the recognition system is found to increase considerably. It is imperative to note that the size

of the neural network used increases as the number of hidden nodes are increased which would prevent the recognition system from being time efficient in low memory systems.

An adaptive learning rate gradient descent training function (traingdx) is used to train the network. The speciality of this training function is that the learning rate can be adaptively varied as the training progresses. In other words if the training is proceeding in the required direction then the learning rate gets increased and vice versa. Once the network is fully trained, it can be instantly used for recognition.

The input character to be recognized is sent to the recognition module. The character's contour is determined before forming the chaincode matrix. It may be recollected that the chaincode matrix comprises of the coordinates of the pixels in the contour. The 20 DFT Components are computed and the components are divided into real and imaginary parts so that 40 entities are obtained. This vector containing 40 entries is fed into the trained neural network. The output of the neural network is a vector with 177 entries.

By carefully analyzing the output vector, it can be seen that only one entry clearly dominates others in terms of its value. This node is said to be the winner and the character corresponding to the winning node is returned as the output of the character recognition module.

6.8 A Final Note on Address Parsing and Recognition

Once a character is recognized, it is appended with the previously recognized character of the word. Thus, the destination address has now been converted into system interpretable words. As the fields of the words have already been identified by address parsing, the next step is to check if all the information provided by these fields are concurrent with each other. The concurrency checker module plays a major role in fool-proof address interpretation. The objective of the concurrency checker module is to check if the Pincode, City, Locality, District and state are concurrent with each other. By making such a check, an address can be interpreted even if it contains some explicit spelling mistakes. Further, such a module is highly helpful when the destination address does not contain certain imperative fields for DPC generation such as the Pincode.

The database that is created should be resilient to the different forms of representing the name of a City or Place. For instance, Coimbatore can be written also as CBE and Kovai. So, the database also contains the different forms of an entry in addition to the predominantly used form.

7. Delivery Point Code (DPC) Generation:

The DPC is a 12 digit code which comprises of

1. A 6 digit Pin Code.
2. A Control Digit.
3. A 5 digit Add-On.

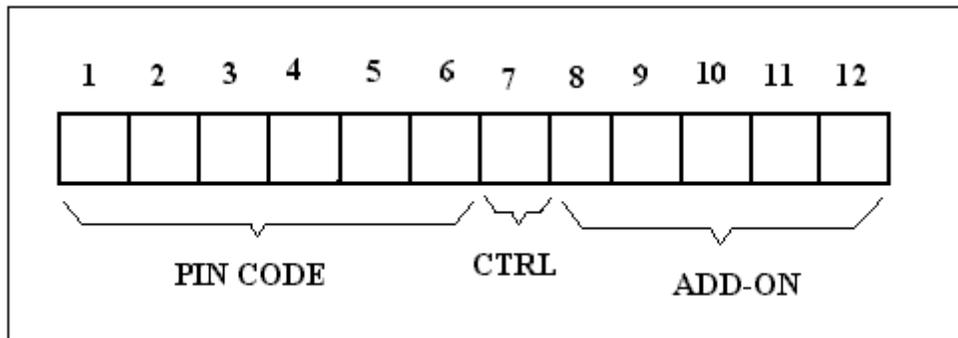


Figure 22: Delivery Point Code Format

In majority of the cases, the Add-On refers to the street code assigned. In special cases, such as the one shown in Figure 23, there is no street name in the address. Instead a field called PO BOX number is included. In this case, the add-on will be this PO BOX number.

7.1 Control Digit Combinations

The control digit (Digit 7 of DPC) is used to differentiate between street code and PO Box code. CTRL is also used to specify any error conditions. The various digit combinations of CTRL are enumerated in Table 1.

CTRL	Function
0	Add-On represents Street code
1	Add-On represents PO BOX number
2	Error: Unable to determine Add-On
3 to 9	Expansion Slots

Table 1: CTRL combinations

<p>S.Balaji PO Box No: 2431 Salem - 636 001</p>
--

Figure 23: An Address format

7.2 Real Time Implementation

Two databases should be employed by the Automatic Mail Processor (AMP). One is the database containing all the pin codes and their corresponding locations. This database is absolutely essential in cases where pin codes are not / partly specified in the destination address. Even if the Pin Code is specified, this database would be used by the Concurrency checker module of APR to check if the Pin Code is concurrent with the city

and locality. The second database contains the Street names of the particular City (where the AMP is located) and the corresponding street codes.

It must be noted that the AMP is not required at every post office in India. The idea is to assign a head post office for a particular city/region and send all the outgoing mails there. The head post office which is equipped with AMPs, converts the destination address into barcodes which are printed on to the mail. Every post office is provided with a low-cost barcode reader for automated sorting. Hence, AMPs are located only in head post offices while barcode readers are present in all post offices. This would prove that the system is highly efficient besides being cost effective.

7.3 Universal Networking and Hierarchical Routing

There are primarily two different ways for DPC assignment using AMP.

1. Universal Networking method
2. Hierarchical Routing method

In both of the above methods, the first database having all the Pin Codes and the second database with Street Codes are present in all the head post offices. A particular head post office will have the street codes pertaining to its limits *ONLY*.

In the Universal Networking method, all the head post offices (HPO) are networked with high speed leased lines. The AMP of a particular HPO recognizes/determines the pin code of a mail (with its own first database). Now, to determine the street code, it uses the universal network to communicate with the

destination HPO. Thus the entire DPC is generated in the source HPO and the barcode for 12 DPC digits is printed on the mail. Unlike Universal networking method, in hierarchical routing, only the Pin Code is bar-coded in the source HPO. As the pin code is bar coded, automated sorting is possible until it reaches the destination HPO. In the destination HPO, the Street name is recognized (by another AMP) and the street code is assigned. Thus there is no network between the HPOs.

The advantage of Universal networking method is that every mail is processed by only one AMP. The disadvantage is that a high-cost network is required for its operation. While the hierarchical routing is advantageous as it avoids the use of a network, it uses two AMPs (one is source HPO and another one in destination HPO) for its operation. *The need for including the street code in the DPC is to further sort down the mails in the destination post office according to the each post man's coverage area.*

In both of the above methods, as the Pin Code is bar coded right in the source HPO, in all further transits, automated sorting is made possible. The process of sorting requires only a barcode reader and **NOT** an AMP. Thus, the need for the presence of high cost equipment at every post office is averted.

8. Bar-Coding

The objective of this module is to dynamically generate a barcode corresponding to the Delivery Point Code (DPC) provided to it. As the range of digits in DPC lie between 0 and 9, four bits would suffice to represent each digit. The barcode format for each of the 10 numbers is illustrated in Figure 24.



Figure 24: Barcode Format



Figure 25: Barcode for DPC = 641013000001

Figure 25 shows the output of the bar coding module when the DPC given to it is 641013000001. Once, the barcode image pattern is created, the next step is to print it on the Envelope.

The final bar coded Envelope at SOURCE HPO for both cases (Universal Networking and Hierarchical Routing) is shown below.

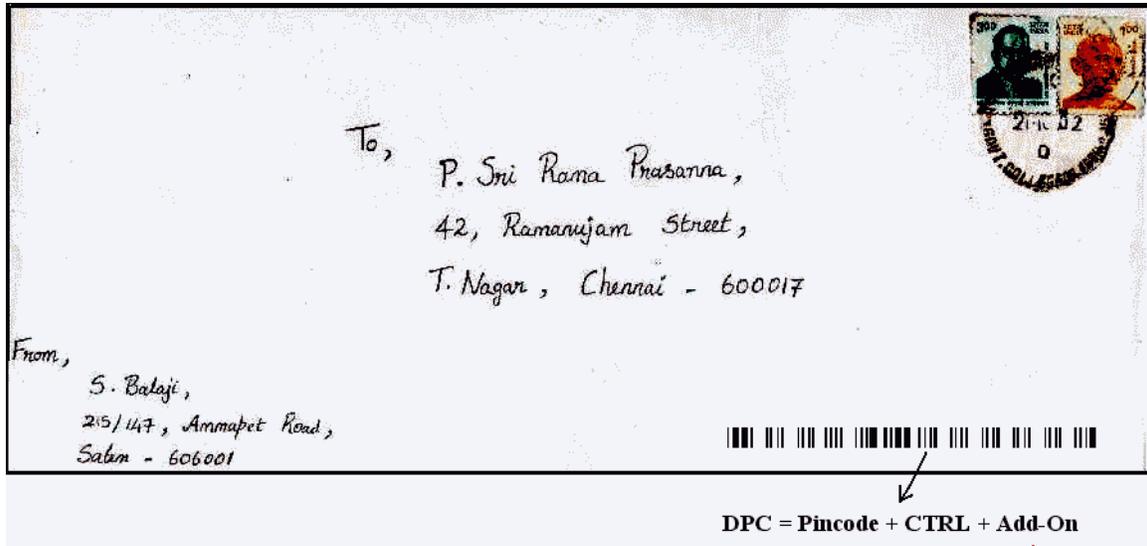


Figure 26: Universal Networking method.

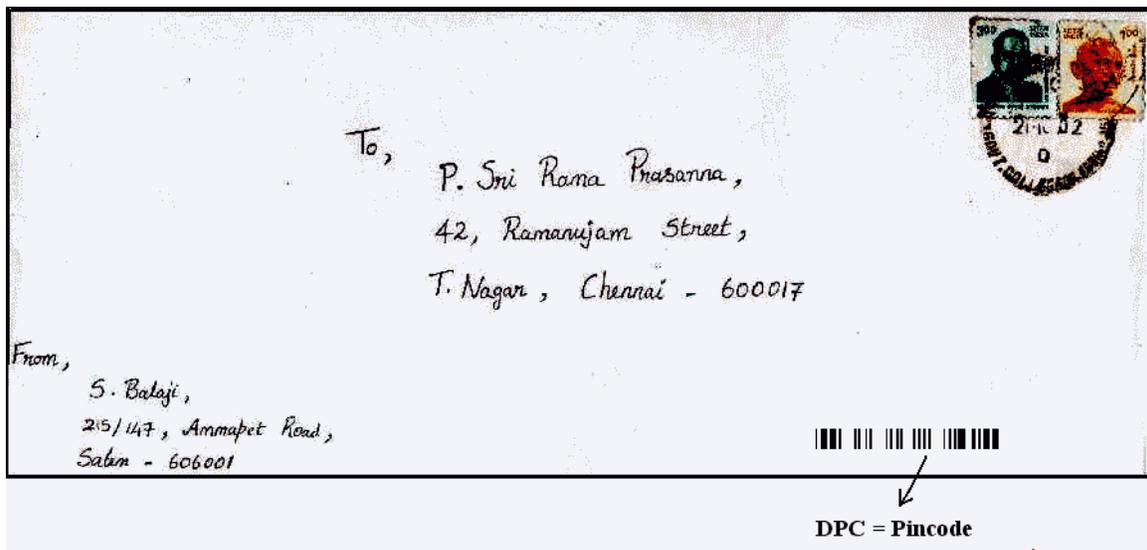


Figure 27: Hierarchical Routing method

The same procedure can be adopted for all other kinds of letters such as the Inland letter, Post card, Post cover etc. As explained before in the case of Inland letters and postal covers, the processing becomes very simple due to the location of predefined spaces to write the destination address. There is no role for the Address Block Location module in such cases.

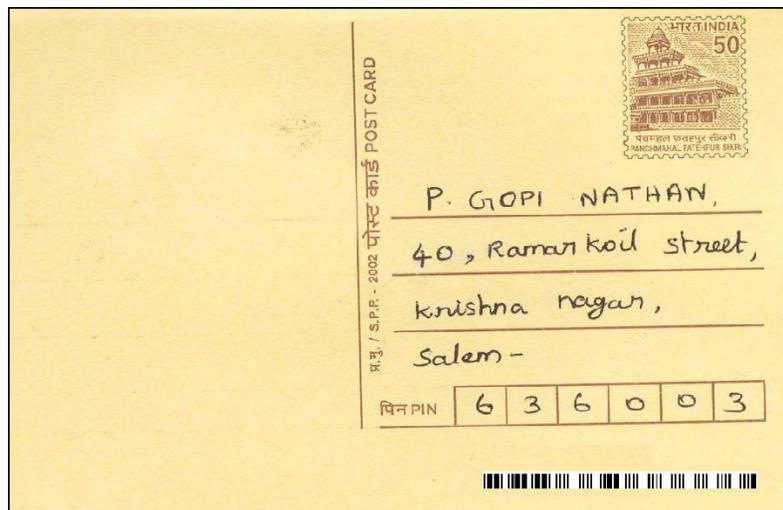
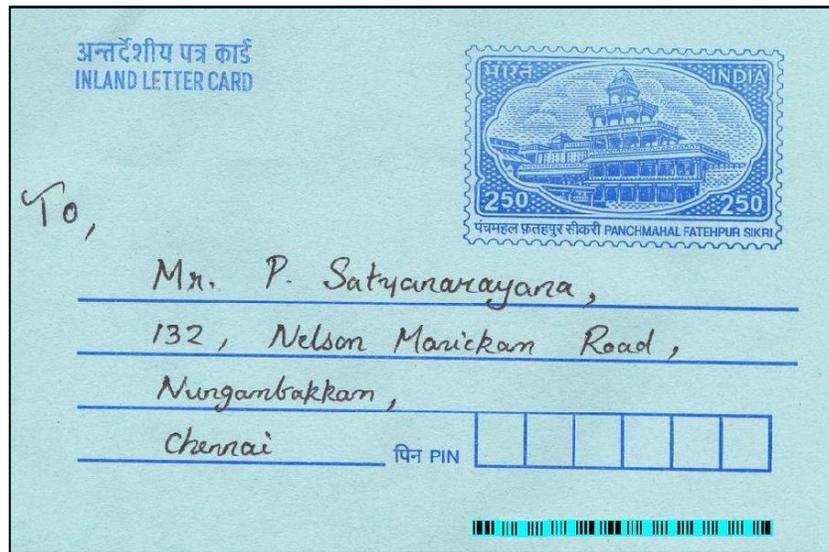


Figure 28: Inland letter and Postcard Samples

Implementation Results

The Figure 29 shows the front end of the implementation in MATLAB. This front end encompasses all the significant steps of our project. The different stages can be executed by clicking the corresponding push-buttons provided in the sides.

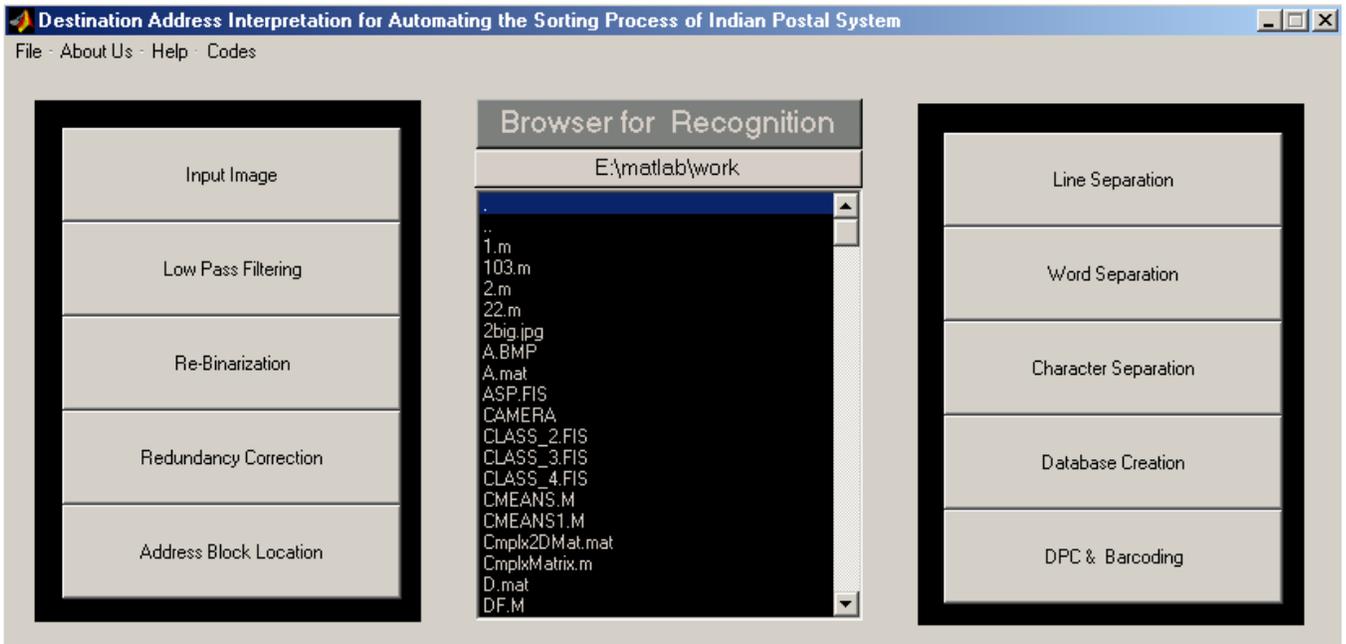


Figure 29: Front End demonstration of the implementation in MATLAB

A Browser has been provided for the recognition module. The input character image to be recognized is selected from the browser list box. Finally, the DPC & Barcoding module generates the DPC corresponding to the destination address along with the barcode corresponding to it. The barcode is generated dynamically in accordance with the digits of the delivery point code. The output for each module is shown in the following pages.

Output for Lowpass Filtering

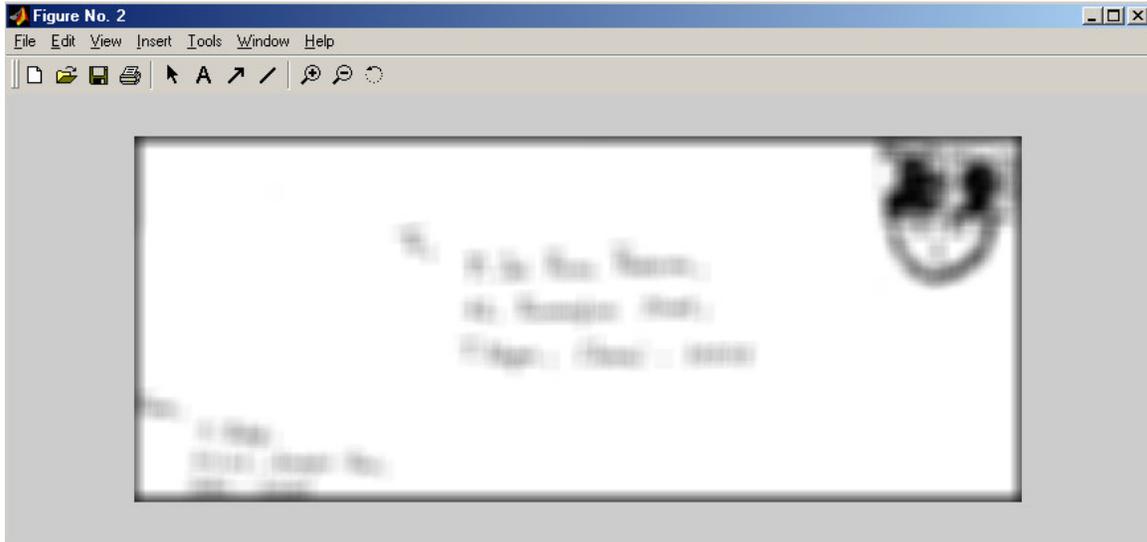


Figure 30: Lowpass Filtering output

Output for Re-Binarization

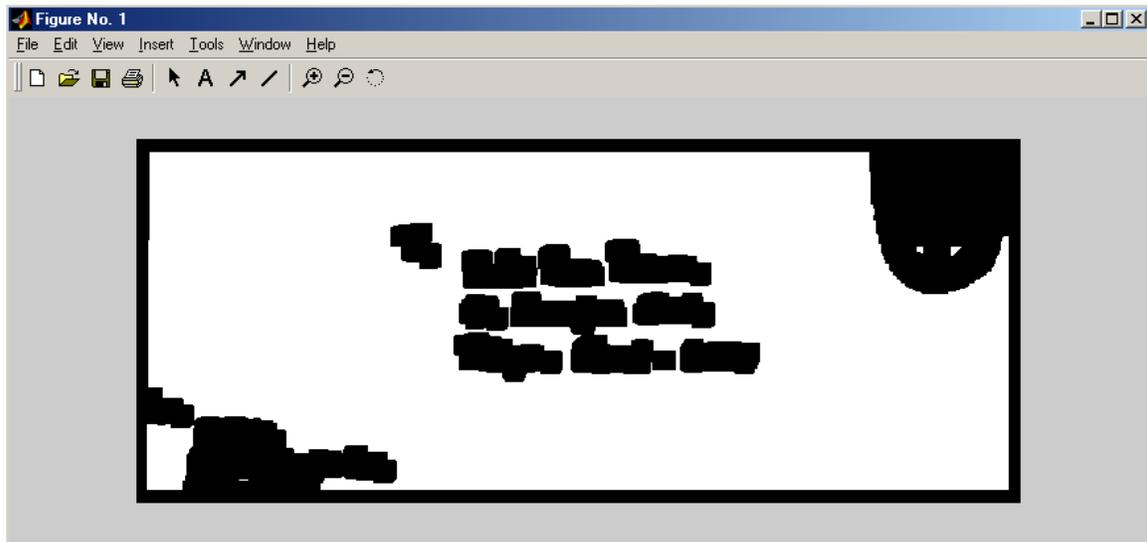


Figure 31: Re-Binarization output

Output for Redundancy Correction

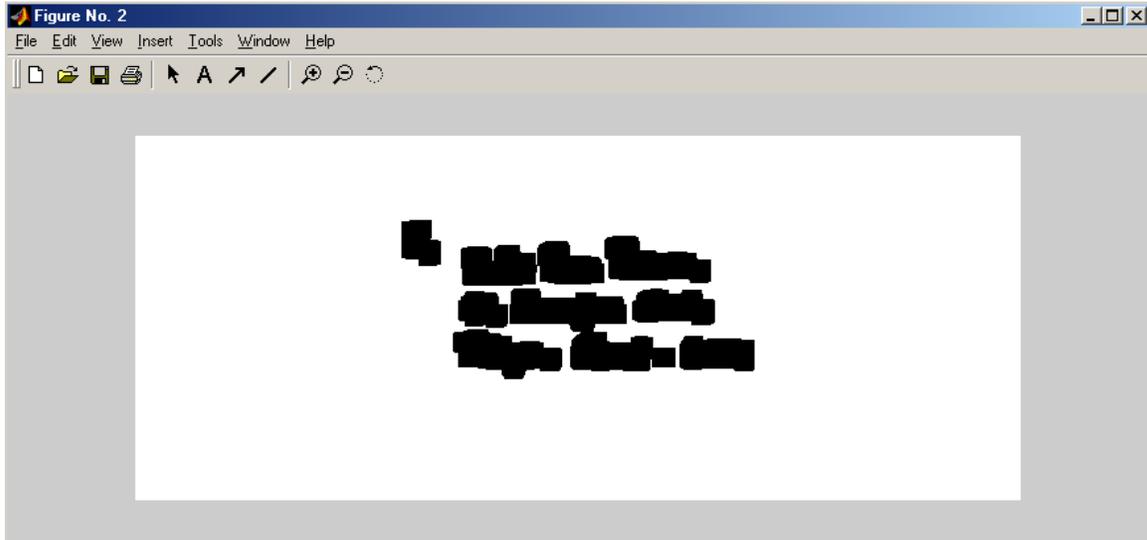


Figure 32: Redundancy Correction Output

Outputs for Address Block Location

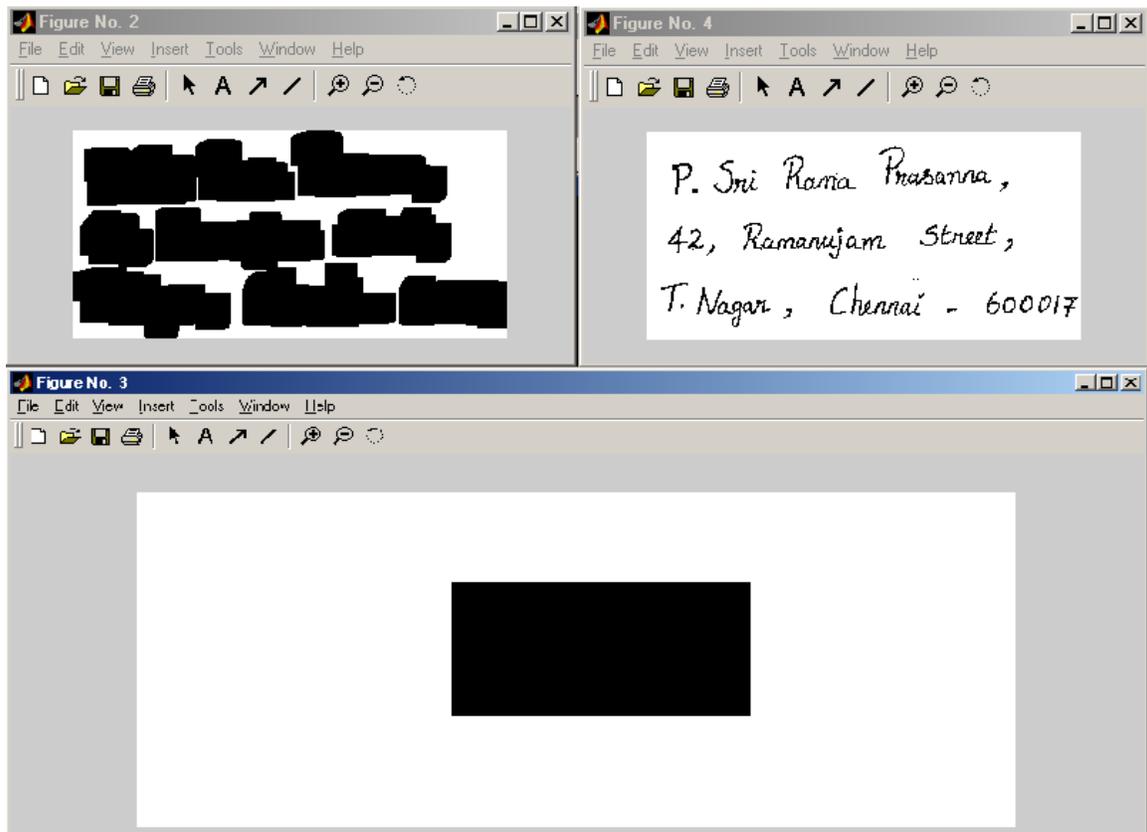


Figure 33: Address Block Location Output

Output for Underline Removal

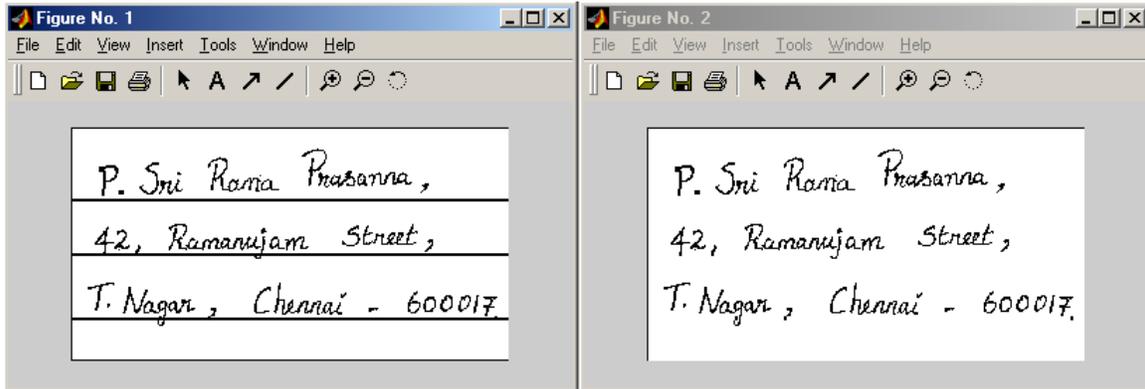


Figure 34: Underline Removal Output

Output for Line Separation

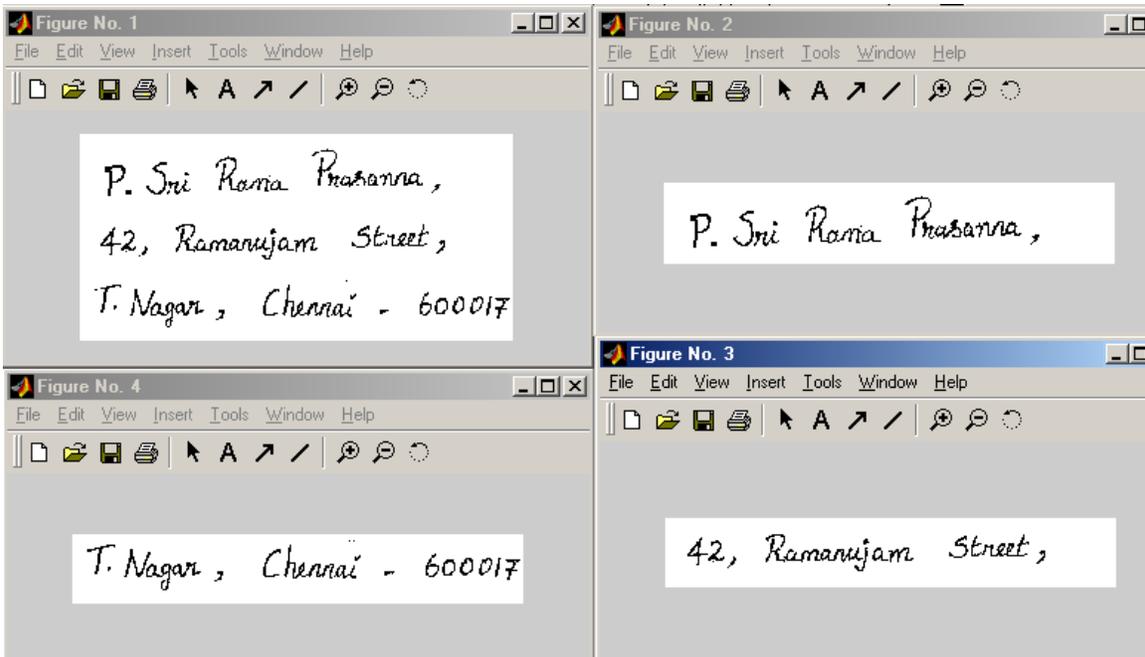


Figure 35: Line Separation Output

Output for Word Separation

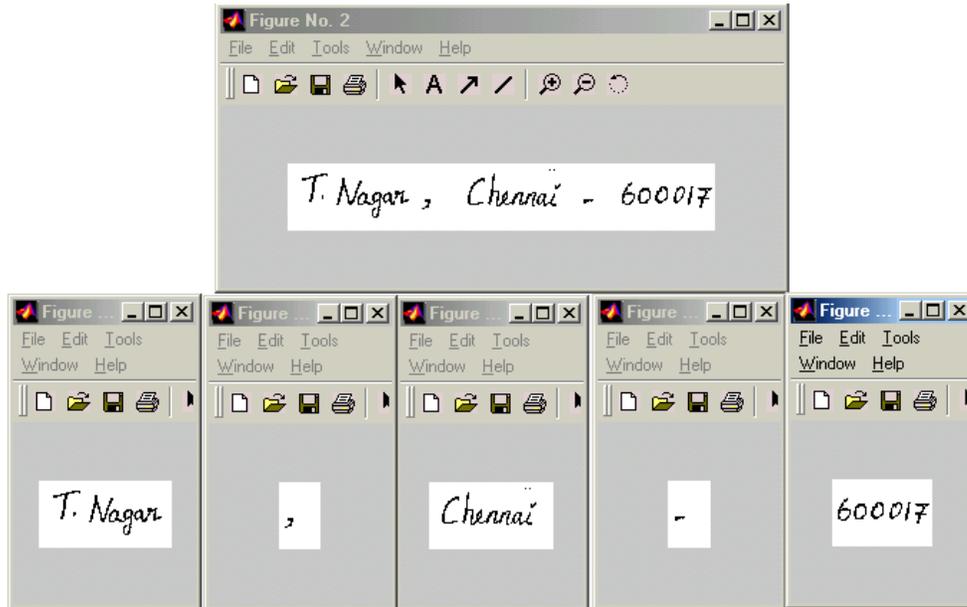


Figure 36: Word Separation output

Output for Character Separation



Figure 37: Character Separation output.

Output for Character Recognition

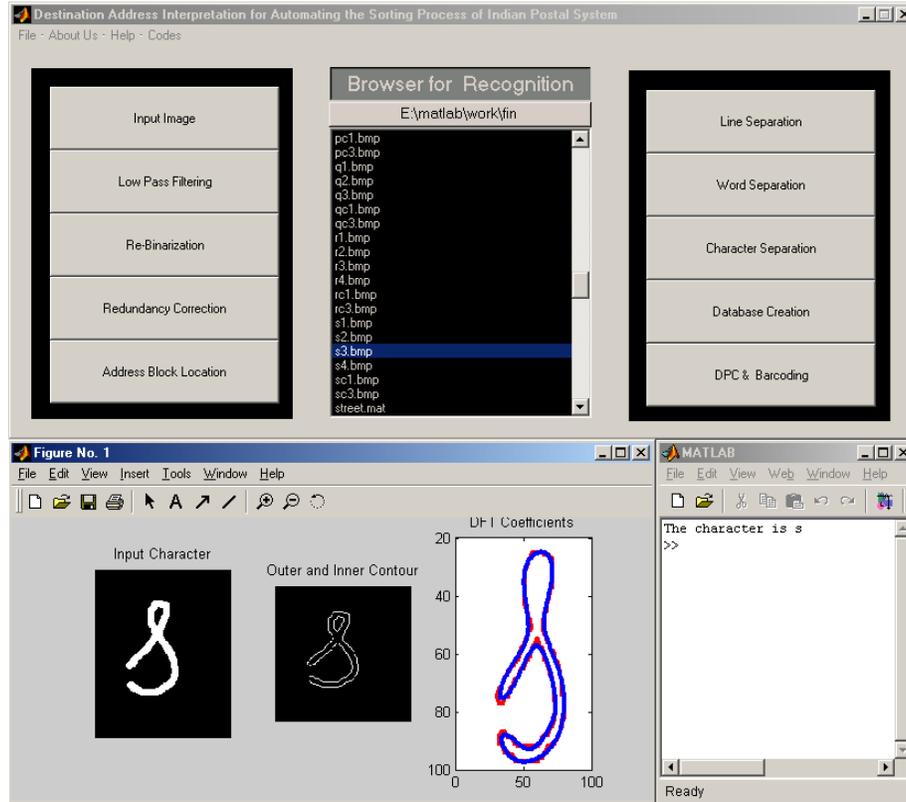


Figure 38: Recognition Output

Output for Database Creation

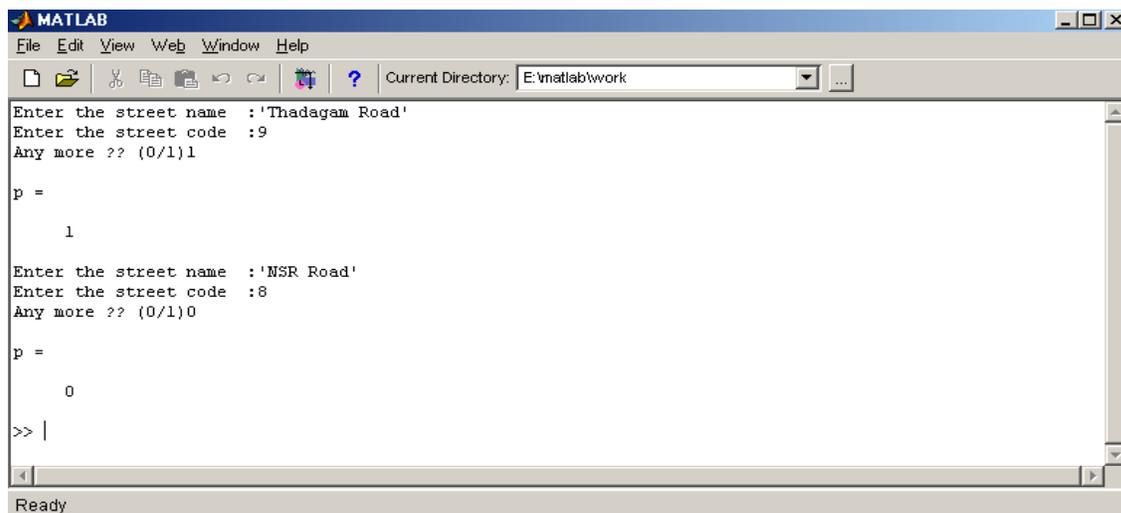


Figure 39: Database Creation Output

Testing

This project was tested with the following different kind of address samples.

S. Balaji 147, Ammapet main road, Salem - 1	<u>C. RAJA</u> PUDUVALAYU <u>MALAIVEPPANKUTTAI (PO)</u> <u>NAMAKKAL (DC) 637404</u>	C. KIRUTHIKA 12B, Agasthya samathy street, Ambasamudram, Tirunelveli District PIN: 627401
R. Sathish R-178 P. or. Hostel Krishna Kol Virudhunagar - 627 190	G. GIRI 114-B, OMANTHURAR ALWAR NAGAR COLONY, NAGAMALAI, MADURAI. TAMIL NADU.	<u>R. Poomima</u> <u>#11, Sugandhi layout</u> <u>United Nagar</u> <u>Coimbatore - 7</u>
<u>R. Sundararajan.</u> New No: 44, Old No: 277/1 <u>Lakshmanaswamy Sabai,</u> <u>K.K. Nagar, Chennai - 78</u>	M. MURALI NO: 8, FOURTH CROSS SARATHINAGAR PONDICHERY - 605011	M. Venmathi 7F, Chokka Nother Kol street, Valluvar - 621117
M. Izam Prabanna No. 20, Arasappa St, Purasaiwalkam, Chennai - 600007.	J. SUNDARAYANAN, H-12, PALLAVAN NAGAR, KANCHIPURAM, 631501	M. HARINI 46/3 I MAIN ROAD, THIRUVANMIUR, CHENNAI-41.
T. Karthikeyan, F4, Pallewarai Apts, Kaliappan street, K.K. Nagar, Coimbatore - 38	<u>A. Anand,</u> <u>10, V main street</u> <u>Kalyanaundaram</u> <u>Nagar</u> <u>Tiruchy - 620001.</u>	E. SANGITA, 21/75, PALANIYAPPA NAGAR, ASTHAMPATTI, SALEM-7.
D. DEEPAK PALDAND, 1/1B, 9TH STREET (WEST), BRAYANT NAGAR, TUTICORIN - 628008	C. Diwan, S/o. Mr. N. Chandra Bose, 49, Nadar street, North, Tuticorin - 628001	T. SUGANYA, 5/110, MAKKINAMPATTI, POLLACHI - 641013.

Figure 40: Samples Used for Testing

Output for DPC and Barcoding

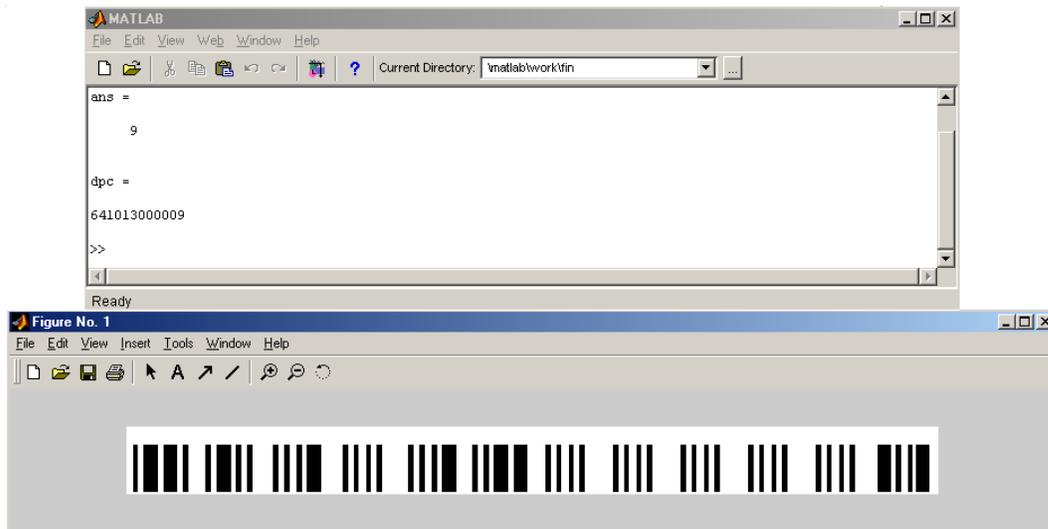


Figure 41: DPC and Barcoding Output

Conclusion and Future Enhancements

The Automatic Mail Processor, whose design and working was explained in the preceding pages, is practically implementable. Cost efficiency is ensured as only barcode readers are required at sorting stations. The speed of operation would be exponentially high when compared to manual sorting. As everything is automated, the chances for errors in interpretation are very less. We have already discussed this project with the Post Master General's (PMG) office, Coimbatore and we certainly believe that this project will revolutionize the Indian Postal System.

Though every module of this project completely serves its purpose, enhancements can be made in the recognition process we have used. Instead of using only the outer contour of each character, the inner contour can also be taken into consideration. If the Back Propagation Neural network is replaced by an Adaptive network like ART, then the training process for new samples would not be cumbersome.

We have implemented this project for the English language. But for real use, this must be extended to other Indian languages too. The process of extending this project to other languages is not difficult as the recognition process undertaken here is independent of the language used. The Neural Network used in the final stage of recognition should be trained with character patterns of other Indian languages too.

MATLAB Codes

Image Denoising and Binarization

```
imfile = 'c:\mail.jpg';
```

```
I = imread (imfile);
```

```
% De-noising
```

```
H = fspecial ('average', 2);
```

```
AI = filter2 (H,I);
```

```
% Binarization
```

```
th = 0.5;
```

```
BI = im2bw (AI, th);
```

Address Block Location

```
% Low pass filter with 20x20 mask
```

```
h = ones (20);
```

```
LI = filter2 (h, BI);
```

```
LI = LI/400;
```

```
% Re-Binarization
```

```
BI = im2bw (LI, 1.0);
```

```
[r c] = size (BI);
```

```
fl = 0;
```

Redundancy Correction

```
% Remove redundant patches in corners
```

```
I = imread ('c:\desti\rb.tif');
```

```
[r c] = size (I);
```

```
figure;
```

```
imshow (I)
```

```
for i = 1 : 30
```

```
    for j = 1:896
```

```
        k = 370-i;
```

```
        I(i,j) = 1;
```

```
        I(k,j) = 1;
```

```
    end
```

```
end
```

```
for i = 1 : 30
```

```
    for j = 1:369
```

```
        k = 897-i;
```

```
        I(j,i) = 1;
```

```
        I(j,k) = 1;
```

```
    end
```

```
end
```

```
for i=1:170
```

```
    for j =1:270
```

```
        I(i,j)=1;
```

```
I(369-i,j)=1;
I(i,897-j) = 1;
I(369-i,897-j)=1;
end
end
figure;
imshow(I)
```

Address Block Extraction

```
I = imread('c:\desti\rc3.tif');
figure;
imshow(I);
[r c] = size(I);
r2=0;c2=0;r1=r;c1=c;
for i = 1:r
    for j = 1:c
        if(I(i,j)== 0)
            if(i<r1)
                r1 = i;
            end
            if(j<c1)
                c1 = j;
            end
        end
    end
end
```

```

    if(i>r2)
        r2 = i;
    end
    if(j>c2)
        c2 = j;
    end
end
end
end
J = imcrop(I,[c1 r1 c2-c1 r2-r1]);
figure;
imshow(J);
for i = r1:r2
    for j = c1:c2
        I(i,j)=0;
    end
end
figure;
imshow(I)
P = imread('f:\fin1_bin.bmp');
E = imcrop(P,[c1 r1 c2-c1 r2-r1]);
figure;
imshow(E)

```

Underline Removal

```
I = imread('g:\a.bmp');  
  
imshow(I);  
  
[r c] = size(I);  
  
n=0;  
  
for i = 2:r  
  
    fl = 1;  
  
        for j = 1:c  
  
            if I(i,j) == 1  
  
                fl = 0;  
  
                break;  
  
            end  
  
        end  
  
    end  
  
    if fl == 1  
  
        for k = 1:c  
  
            if I(i-1,k) == 1  
  
                I(i,k) = 1;  
  
            end  
  
        end  
  
        %I(i,1:end) = 1;  
  
        %n = n+1;  
  
    end  
  
end  
  
end
```

```

figure;

imshow(I)

% n = n+1;

% n = n/3

% for i = 1:c

%   for j = n+1:r-n

%     if I(j,i) == 1

%       if I(j-n,i) == 0

%         if I(j+n,i) == 0

%           I(j,i) = 0;

%         end

%       end

%     end

%   end

% end

%end

%end

%figure;

%imshow(I)

```

Line and Word separation

```

I = imread('c:\desti\rc4.tif');

figure;

imshow(I);

[r c] = size(I);

```

```

n=0;m=1;n=1;
for i = 1:r
    fl = 0;
    for j = 1:c
        if(I(i,j)==0)
            fl =1;
        end
    end
end
if(fl==0)
    n = n+1;
    if(n>20)
        r4(m)= i;
        m = m+1;
        n=0;
    end
end
end
end
I1 = imcrop(I,[1 1 305 r4(1)]);
I2 = imcrop(I,[1 r4(1) 305 r4(2)-r4(1)]);
I3 = imcrop(I,[1 r4(2) 305 146-r4(2)]);
figure;
imshow(I1);figure;
imshow(I2);figure;

```

```

imshow(I3);

% Word Separation

I = imread('c:\desti\ls.tif');

figure;imshow(I);

[r c] = size(I);

n = 0;m = 1;

for j = 1:c

    fl = 0;

    for i = 1:r

        if(I(i,j)==0)

            fl = 1;

        end

    end

    if(fl==0)

        n = n+1;

        if(n>21)

            c4(m)=j;

            n=0;

            m = m+1;

        end

    end

end

m = m-1;

```

```

figure;

imshow( imcrop(I,[1,1,c4(1),r]));

for i = 2:m

    figure;imshow(imcrop(I,[c4(i-1),1,c4(i)-c4(i-1),r]));

end

figure;

imshow( imcrop(I,[c4(m),1,c-c4(m),r]));

```

Character Separation

```

I = imread('c:\desti\ws1.tif');

figure;imshow(I);

[r c] = size(I);

n = 0;m = 1;rr=1;

for j = 1:c

    fl = 0;

    for i = 1:r

        if(I(i,j)==0)

            fl = fl+1;

            rr = i;

        end

    end

    if(fl<7)

        n = n+1;

```

```

    if(n>1)
        if(rr>70)
            c4(m)=j;
            n=0;
            m = m+1;
        end
    end
end
end
end
m = m-1;
figure;
imshow( imcrop(I,[1,1,c4(1),r]));
for i = 2:m
    figure;imshow(imcrop(I,[c4(i-1),1,c4(i)-c4(i-1),r]));
end
figure;
imshow( imcrop(I,[c4(m),1,c-c4(m),r]));

```

Character Recognition

Chain coding and DFT Representation of Samples

```

imfile='f:\fin\zn73.bmp';
tmp=imread(imfile);
tmp=imresize(tmp,[128,128]);

```

```

sub_im=zeros(size(tmp)+[2 2]);
sub_im(2:end-1,2:end-1)=tmp;
SE=ones(3,3);
sub_im=erode(sub_im,SE);
sub_im=double(dilate(sub_im,SE));
ct_im=zeros(size(sub_im));
for m=2:size(sub_im,1)-1
for n=2:size(sub_im,2)-1
if sub_im(m,n)==1
tmp=sub_im(m-1,n)+sub_im(m+1,n)+sub_im(m,n-1)+sub_im(m,n+1);
if tmp~=4
ct_im(m,n)=1;
end
end
end
end
figure;
imshow(ct_im);
im = ct_im;
[r c] = size(im);
fl = 0;
% To determine starting pixel (r1,c1)
for i = 1:r

```

```

for ii = 1:c
    if im(i,ii) == 1
        r1 = i;
        c1 = ii;
        fl = 1;
        break;
    end
end
if fl == 1
    break;
end
end
cc(1,1:2) = [r1,c1];
fr = r1;
fc = c1;
%disp('bottom left');
%im(r1+1,c1-1)
%disp('bottom');
%im(r1+1,c1)
if im(r1+1,c1-1) == 1
    r1 = r1+1;
    c1 = c1-1;
    cc(2,1:2) = [r1,c1];

```

```

    d = 5;
else
    r1 = r1 + 1;
    cc(2,1:2) = [r1,c1];
    d=6;
end
i=2;
while(1)
    if(r1 == fr)
        if(c1 == fc)
            break;
        end
    end
    i = i+1;
switch d
case 0
    [cc,d,r1,c1] = zero(r1,c1,cc,im,i);
case 1
    [cc,d,r1,c1] = one(r1,c1,cc,im,i);
case 2
    [cc,d,r1,c1] = two(r1,c1,cc,im,i);
case 3
    [cc,d,r1,c1] = three(r1,c1,cc,im,i);

```

```

case 4

    [cc,d,r1,c1] = four(r1,c1,cc,im,i);

case 5

    [cc,d,r1,c1] = five(r1,c1,cc,im,i);

case 6

    [cc,d,r1,c1] = six(r1,c1,cc,im,i);

case 7

    [cc,d,r1,c1] = seven(r1,c1,cc,im,i);

end

end

np = size(cc,1);

for l = 1:np-1

    ch(l,1:2) = cc(l,1:2);

end

np = size(ch,1);

p=ch(1:np,2)+ ch(1:np,1)*j;

N=np;

[k,n]=meshgrid(0:N-1,0:N-1);

E=exp(-2*pi*j*k.*n/N);

a=E*p;

ra=a;

ra(11:end-10)=0;

rp=inv(E)*ra;

```

```

figure;

plot(real(p),imag(p),'r.',real(rp),imag(rp),'b.');
```

axis ij;

```

l = size(ra,1);

for i = 1:10

    pp1(i,1) = ra(i);

end

j = 11;

for i = 1-9 : 1

    pp1(j,1) = ra(i);

    j = j + 1;

end

w=pp1;
```

Function Definitions

```
function [cc,d,r1,c1]=zero(r1,c1,cc,im,i)
```

```

if im(r1,c1+1) == 1

    c1 = c1+1;

    d = 0;

    cc(i,1:2) = [r1,c1];

    break;

end

if im(r1-1,c1) == 1

    r1 = r1-1;

    d = 2;
```

```

    cc(i,1:2) = [r1,c1];
    break;
end
if im(r1-1,c1+1) == 1
    r1 = r1-1;
    c1 = c1+1;
    d = 1;
    cc(i,1:2) = [r1,c1];
    break;
end
if im(r1+1,c1) == 1
    r1 = r1+1;
    d = 6;
    cc(i,1:2) = [r1,c1];
    break;
end
if im(r1+1,c1+1) == 1
    r1 = r1+1;
    c1 = c1+1;
    d = 7;
    cc(i,1:2) = [r1,c1];
    break;
end

```

```
function [cc,d,r1,c1]=one(r1,c1,cc,im,i)
```

```
    if im(r1-1,c1+1) == 1
```

```
        r1 = r1-1;
```

```
        c1 = c1+1;
```

```
        d = 1;
```

```
        cc(i,1:2) = [r1,c1]; break;
```

```
    end
```

```
    if im(r1-1,c1) == 1
```

```
        r1 = r1-1;
```

```
        d = 2;
```

```
        cc(i,1:2) = [r1,c1]; break;
```

```
    end
```

```
    if im(r1,c1+1) == 1
```

```
        c1 = c1+1;
```

```
        d = 0;
```

```
        cc(i,1:2) = [r1,c1]; break;
```

```
    end
```

```
    if im(r1-1,c1-1) == 1
```

```
        r1 = r1-1;
```

```
        c1 = c1-1;
```

```
        d = 3;
```

```
        cc(i,1:2) = [r1,c1]; break;
```

```
    end
```

```
if im(r1+1,c1+1) == 1
    r1 = r1+1;
    c1 = c1+1;
    d = 7;
    cc(i,1:2) = [r1,c1]; break;
end
```

```
function [cc,d,r1,c1]=two(r1,c1,cc,im,i)
```

```
if im(r1-1,c1) == 1
    r1 = r1-1;
    d = 2;
    cc(i,1:2) = [r1,c1]; break;
end
```

```
if im(r1,c1+1) == 1
    c1 = c1+1;
    d = 0;
    cc(i,1:2) = [r1,c1]; break;
end
```

```
if im(r1-1,c1+1) == 1
    r1 = r1-1;
    c1 = c1+1;
    d = 1;
    cc(i,1:2) = [r1,c1]; break;
```

```
end
if im(r1-1,c1-1) == 1
    r1 = r1-1;
    c1 = c1-1;
    d = 3;
    cc(i,1:2) = [r1,c1]; break;
end
```

```
end
if im(r1,c1-1) == 1
    c1 = c1-1;
    d = 4;
    cc(i,1:2) = [r1,c1]; break;
end
```

```
end
```

```
function [cc,d,r1,c1]=three(r1,c1,cc,im,i)
```

```
    if im(r1-1,c1-1) == 1
        r1 = r1-1;
        c1 = c1-1;
        d = 3;
        cc(i,1:2) = [r1,c1]; break;
end
```

```
end
if im(r1-1,c1) == 1
    r1 = r1-1;
    d = 2;
```

```

    cc(i,1:2) = [r1,c1]; break;
end
if im(r1+1,c1-1) == 1
    r1 = r1+1;
    c1 = c1-1;
    d = 5;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1-1,c1+1) == 1
    r1 = r1-1;
    c1 = c1+1;
    d = 1;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1,c1-1) == 1
    c1 = c1-1;
    d = 4;
    cc(i,1:2) = [r1,c1]; break;
end

```

```

function [cc,d,r1,c1]=four(r1,c1,cc,im,i)

```

```

    if im(r1,c1-1) == 1
        c1 = c1-1;

```

```

d = 4;
cc(i,1:2) = [r1,c1]; break;
end
if im(r1-1,c1) == 1
    r1 = r1-1;
    d = 2;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1+1,c1-1) == 1
    r1 = r1+1;
    c1 = c1-1;
    d = 5;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1+1,c1) == 1
    r1 = r1+1;
    d = 6;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1-1,c1-1) == 1
    r1 = r1-1;
    c1 = c1-1;
    d = 3;

```

```

    cc(i,1:2) = [r1,c1]; break;
end

function [cc,d,r1,c1]=five(r1,c1,cc,im,i)
if im(r1+1,c1-1) == 1
    r1 = r1+1;
    c1 = c1-1;
    d = 5;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1-1,c1-1) == 1
    r1 = r1-1;
    c1 = c1-1;
    d = 3;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1,c1-1) == 1
    c1 = c1-1;
    d = 4;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1+1,c1) == 1
    r1 = r1+1;

```

```

    d = 6;

    cc(i,1:2) = [r1,c1]; break;

end

if im(r1+1,c1+1) == 1

    r1 = r1+1;

    c1 = c1+1;

    d = 7;

    cc(i,1:2) = [r1,c1]; break;

end

```

function [cc,d,r1,c1]=six(r1,c1,cc,im,i)

```

    if im(r1+1,c1) == 1

        r1 = r1+1;

        d = 6;

        cc(i,1:2) = [r1,c1]; break;

    end

    if im(r1,c1-1) == 1

        c1 = c1-1;

        d = 4;

        cc(i,1:2) = [r1,c1]; break;

    end

    if im(r1,c1+1) == 1

        c1 = c1+1;

```

```

    d = 0;

    cc(i,1:2) = [r1,c1]; break;
end

if im(r1+1,c1-1) == 1

    r1 = r1+1;

    c1 = c1-1;

    d = 5;

    cc(i,1:2) = [r1,c1]; break;
end

if im(r1+1,c1+1) == 1

    r1 = r1+1;

    c1 = c1+1;

    d = 7;

    cc(i,1:2) = [r1,c1]; break;
end

```

```

function [cc,d,r1,c1]=seven(r1,c1,cc,im,i)

```

```

    if im(r1+1,c1+1) == 1

        r1 = r1+1;

        c1 = c1+1;

        d = 7;

        cc(i,1:2) = [r1,c1]; break;
    end

```

```

end
if im(r1-1,c1+1) == 1
    r1 = r1-1;
    c1 = c1+1;
    d = 1;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1,c1+1) == 1
    c1 = c1+1;
    d = 0;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1+1,c1-1) == 1
    r1 = r1+1;
    c1 = c1-1;
    d = 5;
    cc(i,1:2) = [r1,c1]; break;
end
if im(r1+1,c1) == 1
    r1 = r1+1;
    d = 6;
    cc(i,1:2) = [r1,c1]; break;
end

```

Neural Network Training

```
load('E:\finmat\a1.mat');
```

```
load('E:\finmat\b1.mat');
```

```
load('E:\finmat\c1.mat');
```

```
load('E:\finmat\d1.mat');
```

```
load('E:\finmat\e1.mat');
```

```
load('E:\finmat\f1.mat');
```

```
load('E:\finmat\g1.mat');
```

```
load('E:\finmat\h1.mat');
```

```
load('E:\finmat\i1.mat');
```

```
load('E:\finmat\j1.mat');
```

```
load('E:\finmat\k1.mat');
```

```
load('E:\finmat\l1.mat');
```

```
load('E:\finmat\m1.mat');
```

```
load('E:\finmat\n1.mat');
```

```
load('E:\finmat\o1.mat');
```

```
load('E:\finmat\p1.mat');
```

```
load('E:\finmat\q1.mat');
```

```
load('E:\finmat\r1.mat');
```

```
load('E:\finmat\s1.mat');
```

```
load('E:\finmat\t1.mat');
```

```
load('E:\finmat\u1.mat');
```

```
load('E:\finmat\v1.mat');
load('E:\finmat\w1.mat');
load('E:\finmat\x1.mat');
load('E:\finmat\y1.mat');
load('E:\finmat\z1.mat');
load('E:\finmat\zn01.mat');
load('E:\finmat\zn11.mat');
load('E:\finmat\zn21.mat');
load('E:\finmat\zn31.mat');
load('E:\finmat\zn41.mat');
load('E:\finmat\zn51.mat');
load('E:\finmat\zn61.mat');
load('E:\finmat\zn71.mat');
load('E:\finmat\zn81.mat');
load('E:\finmat\zn91.mat');
load('E:\finmat\a2.mat');
load('E:\finmat\b2.mat');
load('E:\finmat\c2.mat');
load('E:\finmat\d2.mat');
load('E:\finmat\e2.mat');
load('E:\finmat\f2.mat');
load('E:\finmat\g2.mat');
load('E:\finmat\h2.mat');
```

```
load('E:\finmat\i2.mat');
load('E:\finmat\j2.mat');
load('E:\finmat\k2.mat');
load('E:\finmat\l2.mat');
load('E:\finmat\m2.mat');
load('E:\finmat\n2.mat');
load('E:\finmat\o2.mat');
load('E:\finmat\p2.mat');
load('E:\finmat\q2.mat');
load('E:\finmat\r2.mat');
load('E:\finmat\s2.mat');
load('E:\finmat\t2.mat');
load('E:\finmat\u2.mat');
load('E:\finmat\v2.mat');
load('E:\finmat\w2.mat');
load('E:\finmat\x2.mat');
load('E:\finmat\y2.mat');
load('E:\finmat\z2.mat');
load('E:\finmat\zn02.mat');
load('E:\finmat\zn12.mat');
load('E:\finmat\zn22.mat');
load('E:\finmat\zn32.mat');
load('E:\finmat\zn42.mat');
```

```
load('E:\finmat\zn52.mat');
load('E:\finmat\zn62.mat');
load('E:\finmat\zn72.mat');
load('E:\finmat\zn82.mat');
load('E:\finmat\zn92.mat');
load('E:\finmat\a3.mat');
load('E:\finmat\b3.mat');
load('E:\finmat\c3.mat');
load('E:\finmat\d3.mat');
load('E:\finmat\e3.mat');
load('E:\finmat\f3.mat');
load('E:\finmat\g3.mat');
load('E:\finmat|h3.mat');
load('E:\finmat|i3.mat');
load('E:\finmat|j3.mat');
load('E:\finmat\k3.mat');
load('E:\finmat\l3.mat');
load('E:\finmat|m3.mat');
load('E:\finmat\n3.mat');
load('E:\finmat\o3.mat');
load('E:\finmat\p3.mat');
load('E:\finmat\q3.mat');
load('E:\finmat\r3.mat');
```

```
load('E:\finmat\s3.mat');
load('E:\finmat\t3.mat');
load('E:\finmat\u3.mat');
load('E:\finmat\v3.mat');
load('E:\finmat\w3.mat');
load('E:\finmat\x3.mat');
load('E:\finmat\y3.mat');
load('E:\finmat\z3.mat');
load('E:\finmat\zn03.mat');
load('E:\finmat\zn13.mat');
load('E:\finmat\zn23.mat');
load('E:\finmat\zn33.mat');
load('E:\finmat\zn43.mat');
load('E:\finmat\zn53.mat');
load('E:\finmat\zn63.mat');
load('E:\finmat\zn73.mat');
load('E:\finmat\zn83.mat');
load('E:\finmat\zn93.mat');
load('E:\finmat\ac3.mat');
load('E:\finmat\bc3.mat');
load('E:\finmat\cc3.mat');
load('E:\finmat\dc3.mat');
load('E:\finmat\ec3.mat');
```

```
load('E:\finmat\fc3.mat');
load('E:\finmat\gc3.mat');
load('E:\finmat\hc3.mat');
load('E:\finmat\ic3.mat');
load('E:\finmat\jc3.mat');
load('E:\finmat\kc3.mat');
load('E:\finmat\lc3.mat');
load('E:\finmat\mc3.mat');
load('E:\finmat\nc3.mat');
load('E:\finmat\oc3.mat');
load('E:\finmat\pc3.mat');
load('E:\finmat\qc3.mat');
load('E:\finmat\rc3.mat');
load('E:\finmat\sc3.mat');
load('E:\finmat\tc3.mat');
load('E:\finmat\uc3.mat');
load('E:\finmat\vc3.mat');
load('E:\finmat\wc3.mat');
load('E:\finmat\xc3.mat');
load('E:\finmat\yc3.mat');
load('E:\finmat\zc3.mat');
load('E:\finmat\ac1.mat');
load('E:\finmat\bc1.mat');
```

```
load('E:\finmat\cc1.mat');
load('E:\finmat\dc1.mat');
load('E:\finmat\ec1.mat');
load('E:\finmat\fc1.mat');
load('E:\finmat\gc1.mat');
load('E:\finmat\hc1.mat');
load('E:\finmat\ic1.mat');
load('E:\finmat\jc1.mat');
load('E:\finmat\kc1.mat');
load('E:\finmat\lc1.mat');
load('E:\finmat\mc1.mat');
load('E:\finmat\nc1.mat');
load('E:\finmat\oc1.mat');
load('E:\finmat\pc1.mat');
load('E:\finmat\qc1.mat');
load('E:\finmat\rc1.mat');
load('E:\finmat\sc1.mat');
load('E:\finmat\tc1.mat');
load('E:\finmat\uc1.mat');
load('E:\finmat\vc1.mat');
load('E:\finmat\wc1.mat');
load('E:\finmat\xc1.mat');
load('E:\finmat\yc1.mat');
```

```

load('E:\finmat\zc1.mat');
load('E:\finmat\b4.mat');
load('E:\finmat\dc4.mat');
load('E:\finmat\e4.mat');
load('E:\finmat\ec4.mat');
load('E:\finmat\f4.mat');
load('E:\finmat\h4.mat');
load('E:\finmat\k4.mat');
load('E:\finmat\r4.mat');
load('E:\finmat\s4.mat');
load('E:\finmat\v4.mat');
load('E:\finmat\w4.mat');
load('E:\finmat\x4.mat');
load('E:\finmat\y4.mat');
load('E:\finmat\z4.mat');
load('E:\finmat\zn34.mat');
load('E:\finmat\zn74.mat');
load('E:\finmat\zn84.mat');

```

```

P = [a1 b1 c1 d1 e1 f1 g1 h1 i1 j1 k1 l1 m1 n1 o1 p1 q1 r1 s1 t1 u1 v1 w1 x1 y1 z1 zn01
zn11 zn21 zn31 zn41 zn51 zn61 zn71 zn81 zn91 a2 b2 c2 d2 e2 f2 g2 h2 i2 j2
k2 l2 m2 n2 o2 p2 q2 r2 s2 t2 u2 v2 w2 x2 y2 z2 zn02 zn12 zn22 zn32 zn42
zn52 zn62 zn72 zn82 zn92 a3 b3 c3 d3 e3 f3 g3 h3 i3 j3 k3 l3 m3 n3 o3 p3 q3
r3 s3 t3 u3 v3 w3 x3 y3 z3 zn03 zn13 zn23 zn33 zn43 zn53 zn63 zn73 zn83
zn93 ac1 bc1 cc1 dc1 ec1 fc1 gc1 hc1 ic1 jc1 kc1 lc1 mc1 nc1 oc1 pc1 qc1
re1 se1 te1 ue1 ve1 we1 xe1 ye1 ze1 ac3 bc3 cc3 dc3 ec3 fc3 gc3 hc3 ic3 jc3
kc3 lc3 mc3 nc3 oc3 pc3 qc3 rc3 sc3 tc3 uc3 vc3 wc3 xc3 yc3 zc3 b4 dc4 e4
ec4 f4 h4 k4 r4 s4 v4 w4 x4 y4 z4 zn34 zn74 zn84];

```

```

clear a1 b1 c1 d1 e1 f1 g1 h1 i1 j1 k1 l1 m1 n1 o1 p1 q1 r1 s1 t1 u1 v1 w1 x1 y1 z1 zn01
zn11 zn21 zn31 zn41 zn51 zn61 zn71 zn81 zn91 a2 b2 c2 d2 e2 f2 g2 h2 i2 j2
k2 l2 m2 n2 o2 p2 q2 r2 s2 t2 u2 v2 w2 x2 y2 z2 zn02 zn12 zn22 zn32 zn42
zn52 zn62 zn72 zn82 zn92 a3 b3 c3 d3 e3 f3 g3 h3 i3 j3 k3 l3 m3 n3 o3 p3 q3
r3 s3 t3 u3 v3 w3 x3 y3 z3 zn03 zn13 zn23 zn33 zn43 zn53 zn63 zn73 zn83
zn93 ac1 bc1 cc1 dc1 ec1 fc1 gc1 hc1 ic1 jc1 kc1 lc1 mc1 nc1 oc1 pc1 qc1
re1 se1 te1 ue1 ve1 we1 xe1 ye1 ze1 ac3 bc3 cc3 dc3 ec3 fc3 gc3 hc3 ic3 jc3
kc3 lc3 mc3 nc3 oc3 pc3 qc3 rc3 sc3 tc3 uc3 vc3 wc3 xc3 yc3 zc3 b4 dc4 e4
ec4 f4 h4 k4 l4 m4 n4 o4 p4 q4 r4 s4 t4 u4 v4 w4 x4 y4 z4 zn34 zn74 zn84;

```

```
PR = real(P);
```

```
PI = imag(P);
```

```
clear P;
```

```
%R(1:3,1:36) = PR(1:3,1:36);
```

```
%R(4:8,1:36) = PR(16:20,1:36);
```

```
%I(1:3,1:36) = PI(1:3,1:36);
```

```
%I(4:8,1:36) = PI(16:20,1:36);
```

```
P = [PR ; PI];
```

```
clear PR PI;
```

```
pr = minmax(P);
```

```
net = newff(pr,[2000 177],{'tansig','tansig'},'traingdx');
```

```
%net=init(net);
```

```
clear pr;
```

```
tg = eye(177);
```

```
% net.adaptParam.passes = 200;
```

```
% [net,no,ne]=adapt(net,P,tg);
```

```
net.trainParam.epochs = 7000; % Maximum number of epochs to train.
```

```
net.trainParam.goal = 0 ; % Maximum number of epochs to train.
```

```

%net.trainParam.lr = 0.05;

%net.trainParam.mc = 0.9;

%net.trainParam.max_perf_inc = 1.04;

net.trainParam.min_grad = 1e-18;

netn = train(net,P,tg);

save thenet netn;

save thep P;

```

Recognition

```

imfile='f:\tmp\a.bmp';

tmp=imread(imfile);

tmp=imresize(tmp,[128,128]);

sub_im=zeros(size(tmp)+[2 2]);

sub_im(2:end-1,2:end-1)=tmp;

SE=ones(3,3);

sub_im=erode(sub_im,SE);

sub_im=double(dilate(sub_im,SE));

ct_im=zeros(size(sub_im));

for m=2:size(sub_im,1)-1

for n=2:size(sub_im,2)-1

if sub_im(m,n)==1

tmp=sub_im(m-1,n)+sub_im(m+1,n)+sub_im(m,n-1)+sub_im(m,n+1);

if tmp~=4

```

```

ct_im(m,n)=1;

end

end

end

end

figure;

imshow(ct_im);

im = ct_im;

[r c] = size(im);

fl = 0;

% To determine starting pixel (r1,c1)

for i = 1:r

    for ii = 1:c

        if im(i,ii) == 1

            r1 = i;

            c1 = ii;

            fl = 1;

            break;

        end

    end

end

if fl ==1

    break;

end

```

```

end

cc(1,1:2) = [r1,c1];

fr = r1;

fc = c1;

%disp('bottom left');

%im(r1+1,c1-1)

%disp('bottom');

%im(r1+1,c1)

if im(r1+1,c1-1) == 1

    r1 = r1+1;

    c1 = c1-1;

    cc(2,1:2) = [r1,c1];

    d = 5;

else

    r1 = r1 + 1;

    cc(2,1:2) = [r1,c1];

    d=6;

end

i=2;

while(1)

    if(r1 == fr)

        if(c1 == fc)

            break;

```

```
    end
end
i = i+1;
switch d
case 0
    [cc,d,r1,c1] = zero(r1,c1,cc,im,i);
case 1
    [cc,d,r1,c1] = one(r1,c1,cc,im,i);
case 2
    [cc,d,r1,c1] = two(r1,c1,cc,im,i);
case 3
    [cc,d,r1,c1] = three(r1,c1,cc,im,i);
case 4
    [cc,d,r1,c1] = four(r1,c1,cc,im,i);
case 5
    [cc,d,r1,c1] = five(r1,c1,cc,im,i);

case 6
    [cc,d,r1,c1] = six(r1,c1,cc,im,i);
case 7
    [cc,d,r1,c1] = seven(r1,c1,cc,im,i);
end
end
```

```

np = size(cc,1);
for l = 1:np-1
    ch(l,1:2) = cc(l,1:2);
end
np = size(ch,1);
p=ch(1:np,2)+ ch(1:np,1)*j;
N=np;
[k,n]=meshgrid(0:N-1,0:N-1);
E=exp(-2*pi*j*k.*n/N);
a=E*p;
ra=a;
ra(11:end-10)=0;
l = size(ra,1);
% to extract five lower frequency and five higher frequency components
for i = 1:10
    P(i,1) = ra(i);
end
j = 11;
for i = l-9 : l
    P(j,1) = ra(i);
    j = j + 1;
end
PR = real(P);

```

```

PI = imag(P);
clear P;
P = [PR;PI];
load('sainet2.mat');
O = sim(netc,P);
max = O(1);
m = 1;
for(i = 2:177)
    if O(i)>max
        m = i;
        max = O(i);
    end
end
switch m
case {1,37,73,109,135}
    disp('The character is a');
case {2,38,74,110,136,161}
    disp('The character is b');
case {3,39,75,111,137}
    disp('The character is c');
case {4,40,76,112,138,162}
    disp('The character is d');
case {5,41,77,113,139,163,164}

```

```
disp('The character is e');
case {6,42,78,114,140,165}

disp('The character is f');
case {7,43,79,115,141}

disp('The character is g');
case {8,44,80,116,142,166}

disp('The character is h');
case {9,45,81,117,143}

disp('The character is i');
case {10,46,82,118,144}

disp('The character is j');
case {11,47,83,119,145,167}

disp('The character is k');
case {12,48,84,120,146}

disp('The character is l');
case {13,49,85,121,147}

disp('The character is m');
case {14,50,86,122,148}

disp('The character is n');
case {15,51,87,123,149}

disp('The character is o');
case {16,52,88,124,150}

disp('The character is p');
```

```
case {17,53,89,125,151}
    disp('The character is q');
case {18,54,90,126,152,168}
    disp('The character is r');
case {19,55,91,127,153,169}
    disp('The character is s');
case {20,56,92,128,154}
    disp('The character is t');
case {21,57,93,129,155}
    disp('The character is u');
case {22,58,94,130,156,170}
    disp('The character is v');
case {23,59,95,131,157,171}
    disp('The character is w');
case {24,60,96,132,158,172}
    disp('The character is x');
case {25,61,97,133,159,173}
    disp('The character is y');
case {26,62,98,134,160,174}
    disp('The character is z');
case {27,63,99}
    disp('The number is 0');
case {28,64,100}
```

```

    disp('The number is 1');
case {29,65,101}

    disp('The number is 2');
case {30,66,102,175}

    disp('The number is 3');
case {31,67,103}

    disp('The number is 4');
case {32,68,104}

    disp('The number is 5');
case {33,69,105}

    disp('The number is 6');
case {34,70,106,176}

    disp('The number is 7');
case {35,71,107,177}

    disp('The number is 8');
case {36,72,108}

    disp('The number is 9');
end

```

Concurrency Checker

```

function [pin, ct, lc, st , sr,dt] = concurrent(pin, ct, lc,st,sr,dt )

% Code for concurrency checker

%pin represents pincode; st - state ; sr - represents street name;

```

```

%lc = localityname; ct = city name; dt - district name

load('pin.mat');

load('st.mat');

load('sr.mat');

load('ct.mat');

load('lc.mat');

load('dt.mat')

%[pin, ct, lc, st , sr] = concurrent(pin, ct, lc,st,sr );

% The above statement passes control from the main program to this function

fl = conc(pin,lc);

% Conc is a function that checks is the pincode and locality name are concurrent

if fl == 0

    [pin, lc] = correct(pin,lc,st,dt)

    % This function corrects the pincode or locality

end

fl = conc(st,pin)

if fl == 0

    st = correct(st,pin)

end

% pin and st now represent the pincode anf street code

% These are the imperative fields required for generating the DPC

```

Database for Street Codes

```
clear;clc

load('street');

%n1 = 0;

while (1)

n1 = n1 + 1;

T1 = input('Enter the street name :');

T2 = input('Enter the street code :');

[m,len1] = size(T1);

[m,len2] = size(T2);

S(n1,1:len1) = T1(1,1:len1);

C(n1,1:len2) = T2(1,1:len2);

p = input('Any more ?? (0/1)')

if p == 0

    break;

end

end

save street S C n1;
```

Delivery Point Code Generation

```
load('street');

[m n] = size(S);

T = input('Enter the street name :');
```

```

[lent,n] = size(T);

t = 0;

for i = 1:m

flag =1;

    [lens,n] = size(S(i));

if lent > lens

    len = lens;

else

    len = lent;

end

for j = 1:len

    if S(i,j) ~= T(1,j)

        flag = 0;

        break;

    end

end

if flag == 1

    t = i;

    break;

end

end

C(i,1)

pin = '641013';

```

```

cd = '0';
sc=100000+C(i,1);
sc = num2str(sc);
[11,lensc] = size(sc);
lensc = lensc-1;
for i = 1:lensc
    sc(1,i) = sc(1,i+1);
end
dpc = strcat(pin,cd,sc);
dpc = dpc(1,1:12)

```

Barcode Generation

```

I = ones(60,1) ;
for i = 1:12
    chr = dpc(1,i);
    fn = strcat('e:\proj\barcodes\',chr, '.bmp');
    im = imread(fn);
    im = imresize(im,[60,60]);
    I = [I im];
end
figure;
imshow(I,[0 1]);

```

Front End Codes

```
function varargout = demo3(varargin)

% DEMO3 Application M-file for demo3.fig

% FIG = DEMO3 launch demo3 GUI.

% DEMO3('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 11-Apr-2003 00:03:25

%cd \

% cd fin

if nargin <= 1 % LAUNCH GUI

if nargin == 0

initial_dir = pwd;

elseif nargin == 1 & exist(varargin{1},'dir')

initial_dir = varargin{1};

else

error('Input argument must be a valid directory','Input Argument Error!')

return

end

% Open FIG-file

fig = openfig(mfilename,'reuse'); % Generate a structure of handles to pass to
callbacks, and store it.

% Use system color scheme for figure:

set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

handles = guihandles(fig);

guidata(fig, handles);
```

```

% Populate the listbox

load_listbox(initial_dir,handles)

% Return figure handle as first output argument

if nargin > 0

varargout{ 1 } = fig;

end

elseif ischar(varargin{ 1 }) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try

        [varargout{ 1:nargout }] = feval(varargin{:}); % FEVAL switchyard

    catch

        disp(lasterr);

    end

end

%| ABOUT CALLBACKS:

%| GUIDE automatically appends subfunction prototypes to this file, and

%| sets objects' callback properties to call them through the FEVAL

%| switchyard above. This comment describes that mechanism.

%|

%| Each callback subfunction declaration has the following form:

%| <SUBFUNCTION_NAME>(H, EVENTDATA, HANDLES, VARARGIN)

%|

%| The subfunction name is composed using the object's Tag and the

%| callback type separated by '_', e.g. 'slider2_Callback',

```

```

%l 'figure1_CloseRequestFcn', 'axis1_ButtondownFcn'.

%l

%l H is the callback object's handle (obtained using GCBO).

%l

%l EVENTDATA is empty, but reserved for future use.

%l

%l HANDLES is a structure containing handles of components in GUI using
%l tags as fieldnames, e.g. handles.figure1, handles.slider2. This
%l structure is created at GUI startup using GUIHANDLES and stored in
%l the figure's application data using GUIDATA. A copy of the structure
%l is passed to each callback. You can store additional information in
%l this structure at GUI startup, and you can change the structure
%l during callbacks. Call guidata(h, handles) after changing your
%l copy to replace the stored original so that subsequent callbacks see
%l the updates. Type "help guihandles" and "help guidata" for more
%l information.

%l

%l VARARGIN contains any extra arguments you have passed to the
%l callback. Specify the extra arguments by editing the callback
%l property in the inspector. By default, GUIDE sets the property to:
%l <MFILENAME>(<SUBFUNCTION_NAME>', gcbo, [], guidata(gcbo))
%l Add any extra arguments after the last argument, before the final
%l closing parenthesis.

```

```

% -----
function varargout = togglebutton1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.togglebutton1.
disp('togglebutton1 Callback not implemented yet.')
% -----

function varargout = pushbutton4_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton4.
disp('pushbutton4 Callback not implemented yet.')
% -----

function varargout = pushbutton5_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton5.
disp('pushbutton5 Callback not implemented yet.')
% -----

function varargout = pushbutton6_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton6.
disp('pushbutton6 Callback not implemented yet.')
% -----

function varargout = pushbutton7_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton7.
disp('pushbutton7 Callback not implemented yet.')
% -----

function varargout = pushbutton8_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.pushbutton8.

```

```

disp('pushbutton8 Callback not implemented yet.')

% -----

function varargout = pushbutton9_Callback(h, eventdata, handles, varargin)

% Stub for Callback of the uicontrol handles.pushbutton9.

disp('pushbutton9 Callback not implemented yet.')

% -----

function varargout = pushbutton10_Callback(h, eventdata, handles, varargin)

% Stub for Callback of the uicontrol handles.pushbutton10.

disp('pushbutton10 Callback not implemented yet.')

% -----

function varargout = pushbutton11_Callback(h, eventdata, handles, varargin)

% Stub for Callback of the uicontrol handles.pushbutton11.

disp('pushbutton11 Callback not implemented yet.')

% -----

% Callback for list box - open .fig with guide, otherwise use open

% -----

function varargout = listbox1_Callback(h, eventdata, handles, varargin)

if strcmp(get(handles.figure1,'SelectionType'),'open')

index_selected = get(handles.listbox1,'Value');

file_list = get(handles.listbox1,'String');

filename = file_list{index_selected};

if handles.is_dir(handles.sorted_index(index_selected))

```

```

        cd (filename)

        load_listbox(pwd,handles)
else
    [path,name,ext,ver] = fileparts(filename);

    switch ext

    case '.fig'

        guide (filename)

    otherwise

        try

            fn = strcat(pwd,'\',filename);

            I = imread(fn);

            % handles;

            %figure(get(handles.figure1));

            % figure('Destination Address Interpretation for Automating the Sorting Process of
                Indian Postal System') ;

            %axes(handles.frame10);

            %subplot(axes(handles.frame10));

            % axes(handles.axes1);

            subplot(1,3,1);

            imshow(I);

            %axis off;

            title('Input Character');%figure(1);

            fn = strcat(pwd,'\',filename);

```

```

    ra = tcf(fn);

    %open(filename)

    catch

        errordlg(lasterr,'File Type Error','modal')

    end

end

end

end

% -----
% Read the current directory and sort the names
% -----

function load_listbox(dir_path,handles)

cd (dir_path)

dir_struct = dir(dir_path);

[sorted_names,sorted_index] = sortrows({dir_struct.name}');

handles.file_names = sorted_names;

handles.is_dir = [dir_struct.isdir];

handles.sorted_index = [sorted_index];

guidata(handles.figure1,handles)

set(handles.listbox1,'String',handles.file_names,... 'Value',1)

set(handles.text1,'String',pwd)

```

References

1. Sargur N. Srihari, Venu Govindarajulu, Ajay Shekhawat, (CEDAR), NY 14228, “Interpretation of Handwritten addresses in US Mail Stream”
2. Rafael C. Gonzalez, Richard E.Woods, “Digital Image Processing”, Pearson Education Asia, 1992 edition.
3. Anil K. Jain, “Fundamentals of Digital Image Processing”, EEE, 1989 edition.
4. Dwayne Phillips, “Image Processing in C”, BPB Publications, 1995.
5. Nick Efford, “Digital Image Processing”, Pearson Education, 2000.
6. James A. Freeman, David M. Skapura, “Neural Networks Algorithms, Applications and Programming Techniques”, Pearson Education Asia 1991 edition.