# OPTICAL CHARACTER RECOGNITION USING MATLAB

**A PROJECT REPORT**

**Submitted by**

**AAKASH SOOD [10608001]**

**ABHYANK SRINET [10608005]**

**AMERENDRA DAULAT [10608010]**

**RANAJAY SIT [10608078]**

**Under the guidance of**

**MRS. JEYASHREE**

**(Assistant Professor, Department of Instrumentation & Control Engineering)**

**In the partial fulfilment for the award of the degree**

**Of**

**BACHELOR OF TECHNOLOGY**

**In**

**INSTRUMENTATION & CONTROL ENGINEERING**

**Of**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**S.R.M Nagar, Kattankulathur, Kancheepuram District**

**APRIL 2012**

**SRM UNIVERSITY**

**(Established under the section 3 of UGC Act, 1956)**

## BONAFIDE CERTIFICATE

Certified that this project report titled **"OPTICAL CHARACTER RECOGNITION USING MATLAB"** is the bonafide work of **AAKASH SOOD(10608001), ABHYANK SRINET(10608005), AMERENDRA DAULAT(10608010) & RANAJAY SIT(10608078)**, who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on these or any other candidates.

**MRS. JEYASHREE**

Assistant Professor

Instrumentation and Control Engg.

**DR. VIMALA JULIET**

Head of Department

Instrumentation & Control Engg.

**Internal Examiner**

**External Examiner**

**Date:**

# ABSTRACT

Handwriting recognition has been one of the active and challenging research areas in the field of image processing and pattern recognition. It has numerous applications which include, reading aid for blind, bank cheques and conversion of any hand written document into structural text form. In this project an attempt is made to recognize handwritten characters for English alphabets and numbers using feature extraction of characterization loci. Each character data set contains 26 alphabets. A number of data sets are used for training the SVM Database. The trained network is used for classification and recognition. In the proposed system, each character is resized into 36x27 pixels, which is directly subjected to training. That is, each resized character has 972 pixels and these pixels are taken as features for training the neural network. The results show that the proposed system yields good recognition rates which are comparable to other similar schemes for handwritten character recognition.

# ACKNOWLEDGEMENT

Our project is a confluence of several varied thoughts harmoniously integrated into a resourceful product. It is natural that we feel indebted to several people for having made this project possible.

We would like to extend our sincere appreciation and gratitude to Dr. Vimala Juliet Head of the department of Instrumentation and Control engineering for her constant encouragement by providing the necessary infrastructure and resources.

We are extremely grateful to Professor Y Jeyashree of Instrumentation and Control department SRM University for the deluge of ideas and assistance that she has provided to us all through the project .We are indeed fortunate to obtain constant guidance and inspiration from her.

We would also like to make use of this opportunity to thank our Class In-charge Mrs. Prema for providing us her keen in-sight throughout our project tenure and providing valuable suggestions to improve our project.

We would also like to take this opportunity to thank all the people who have helped us in completing this project and who have stood by us and constantly motivated us to complete this project.

# CONTENTS

# LIST OF FIGURES

# INTRODUCTION

## 1 Optical Character Recognition

Optical Character Recognition deals with the problem of recognizing optically processed Characters. Optical recognition is performed off-line after the writing or printing has been completed, as opposed to on-line recognition where the computer recognizes the characters as they are drawn. Both hand printed and printed characters may be recognized, but the performance is directly dependent upon the quality of the input documents.

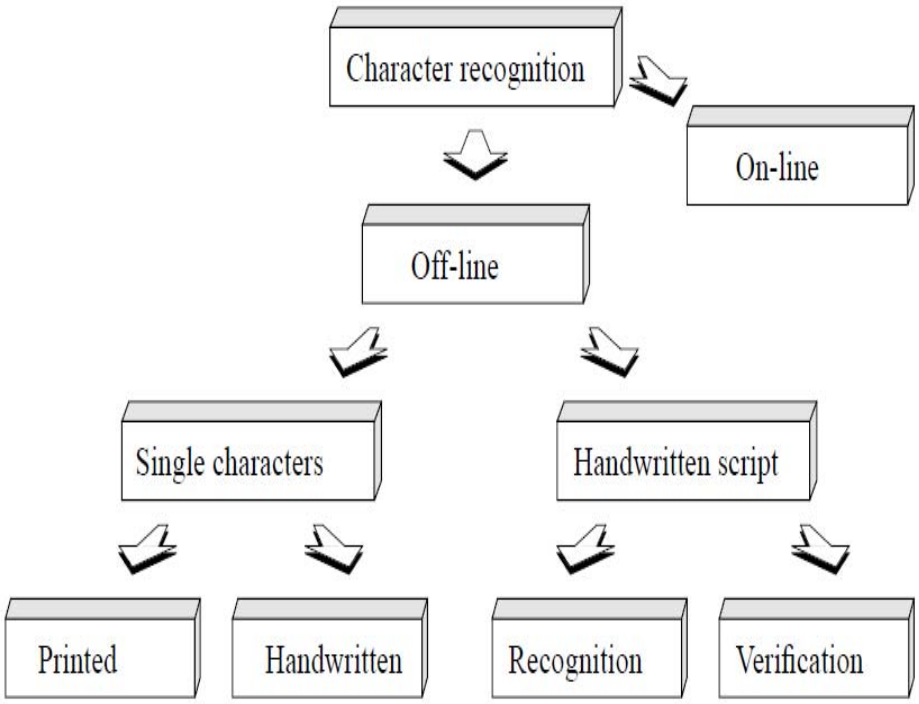
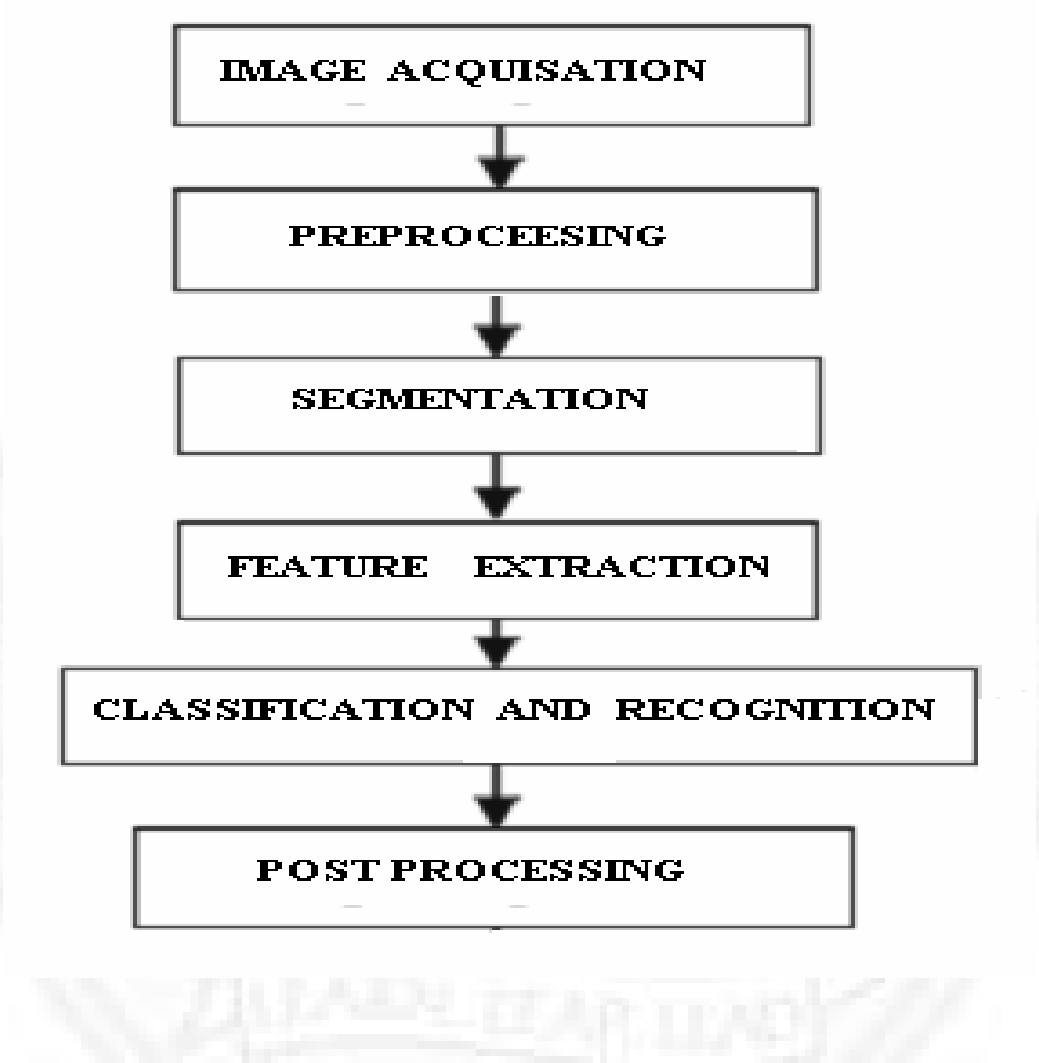



FIGURE 1: The different areas of character recognition

# 2 BLOCK DIAGRAM




The above block diagram shows the dataflow pattern according to which a character is recognised optically using the optical character recognition in MATLAB. The sensor used is basically a CMOS sensor which can be found in a VGA camera followed by a the image acquisition into the MATLAB system where the image is acquired into the system by the method of using the various relay systems .This is then followed by the various pre-processing techniques such as edge detection thresholding conversion to grayscale from rgb etc. The next step is feature extraction which is used to extract the features of the images using the characteristic loci method and then finally the image is recognised by the system via classification and comparing with the database. and can be given as output to the user.

# 3 THE HISTORY OF OCR

Methodically, character recognition is a subset of the pattern recognition area. However,it was character recognition that gave the incentives for making pattern recognition and image analysis matured fields of science.

## 3.1 THE VERY FIRST ATTEMPTS

To replicate the human functions by machines, making the machine able to perform tasks like reading, is an ancient dream. The origins of character recognition can actually be found back in 1870. This was the year that C.R.Carey of Boston Massachusetts invented the retina scanner which was an image transmission system using a mosaic of photocells.

Two decades later the Polish P. Nipkow invented the sequential scanner which was a major breakthrough both for modern television and reading machines.During the first decades of the 19'th century several attempts were made to develop devices to aid the blind through experiments with OCR. However, the modern version of OCR did not appear until the middle of the 1940's with the development of the digital computer. The motivation for development from then on, was the possible applications within the business world.

## 3.2 THE START OF OCR

By 1950 the technological revolution was moving forward at a high speed, and electronic data processing was becoming an important field. Data entry was performed through punched cards and a cost-effective way of handling the increasing amount of data was needed. At the same time the technology for machine reading was becoming sufficiently mature for application, and by the middle of the 1950's OCR machines became commercially available. The first true OCR reading machine was installed at Reader's Digest in 1954. This equipment was used to convert typewritten sales reports into punched cards for input to the computer.

### 3.2.1 First Generation OCR

The commercial OCR systems appearing in the period from 1960 to 1965 may be called the first generation of OCR. This generation of OCR machines were mainly characterized by the constrained letter shapes read. The symbols were specially designed for machine reading, and the first ones did not even look very natural. With time multifont machines started to appear, which could read up to ten different fonts. The number of fonts were limited by the pattern recognition method applied, template matching, which compares the character image with a library of prototype images for each character of each font.

### 3.2.2 Second Generation OCR

The reading machines of the second generation appeared in the middle of the 1960's and early 1970's. These systems were able to recognize regular machine printed characters and also had hand-printed character recognition capabilities. When hand-printed characters were considered, the character set was constrained to numerals and a few letters and symbols.

The first and famous system of this kind was the IBM 1287, which was exhibited at the World Fair in New York in 1965. Also, in this period Toshiba developed the first automatic letter sorting machine for postal code numbers and Hitachi made the first OCR machine for high performance and low cost.

In this period significant work was done in the area of standardization. In 1966, a thorough study of OCR requirements was completed and an American standard OCR character set was defined; OCR-A. This font was highly stylized and designed to facilitate optical recognition, although still readable to humans. A European font was also designed, OCR-B, which had more natural fonts than the American standard. Some attempts were made to merge the two fonts into one standard, but instead machines being able to read both standards appeared.

### 3.2.3 THIRD GENERATION OCR

For the third generation of OCR systems, appearing in the middle of the 1970's, the challenge was documents of poor quality and large printed and hand-written character sets.

Low cost and high performance were also important objectives, which were helped by the dramatic advances in hardware technology.

Although more sophisticated OCR-machines started to appear at the market simple OCR devices were still very useful. In the period before the personal computers and laser printers started to dominate the area of text production, typing was a special niche for OCR.

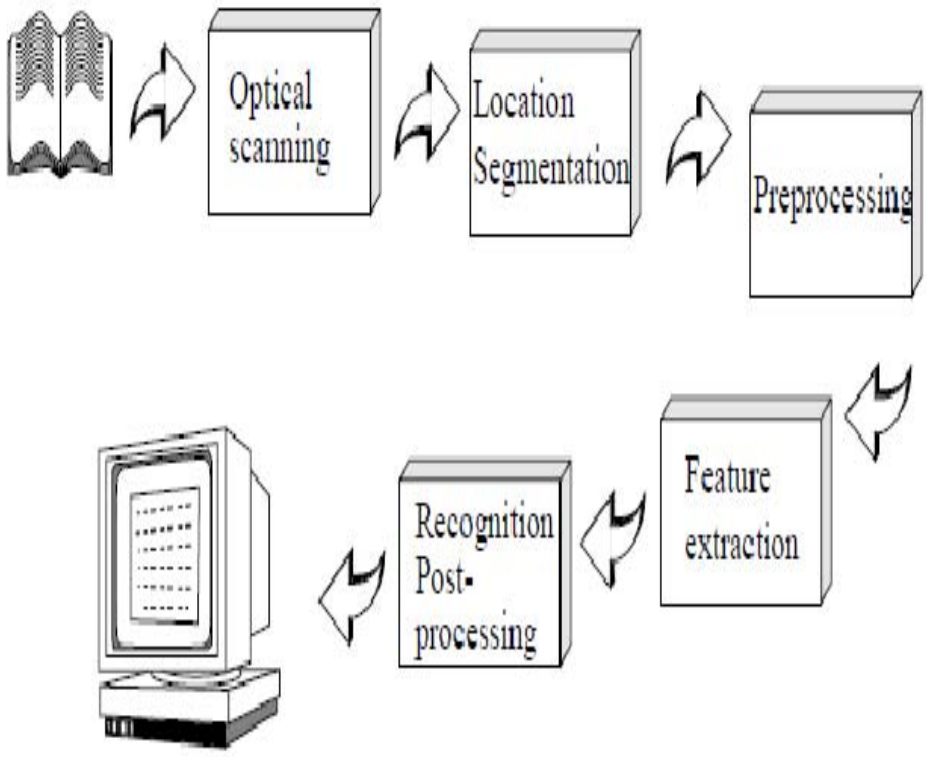
The uniform print spacing and small number of fonts made simply designed OCR devices very useful. Rough drafts could be created on ordinary typewriters and fed into the computer through an OCR device for final editing. In this way word processors, which were an expensive resource at this time, could support several people and the costs for equipment
could be cut.

### 3.2.4 OCR TODAY

Although, OCR machines became commercially available already in the 1950's, only a few thousand systems had been sold world wide up to 1986. The main reason for this was the cost of the systems. However, as hardware was getting cheaper, and OCR systems started to become available as software packages, the sale increased considerably. Today a few thousand is the number of systems sold every week, and the cost of an omnifont, OCR has dropped with a factor of ten every other year for the last 6 years.

# 4 COMPONENTS OF AN OCR SYSTEM

A typical OCR system consists of several components. In figure 3 a common setup is illustrated. The first step in the process is to digitize the analog document using an optical scanner. When the regions containing text are located, each symbol is extracted through a segmentation process. The extracted symbols may then be pre-processed, eliminating noise, to facilitate the extraction of features in the next step. The identity of each symbol is found by comparing the extracted features with descriptions of the symbol classes obtained through a previous learning phase. Finally contextual information is used to reconstruct the words and numbers of the original text.




Components of an ocr system

# 5 METHODS OF OCR

The main principle in automatic recognition of patterns, is first to teach the machine which classes of patterns that may occur and what they look like. In OCR the patterns are letters, numbers and some special symbols like commas, question marks etc., while the different classes correspond to the different characters. The teaching of the machine is performed by showing the machine examples of characters of all the different classes. Based on these examples the machine builds a prototype or a description of each class of characters.

Then, during recognition, the unknown characters are compared to the previously obtained descriptions, and assigned the class that gives the best match.

In most commercial systems for character recognition, the training process has been performed in advance. Some systems do however; include facilities for training in the case of inclusion of new classes of characters. In the next sections these steps and some of the methods involved are described in more detail.

## 5.1 OPTICAL SCANNING

Through the scanning process a digital image of the original document is captured. In OCR optical scanners are used, which generally consist of a transport mechanism plus a sensing device that converts light intensity into gray-levels. Printed documents usually consist of black print on a white background. Hence, when performing OCR, it is common practice to convert the multilevel image into a bilevel image of black and white. Often this process, known as thresholding, is performed on the scanner to save memory space and computational effort.

The thresholding process is important as the results of the following recognition is totally dependent of the quality of the bilevel image. Still, the thresholding performed on the scanner is usually very simple. A fixed threshold is used, where gray-levels below this threshold is said to be black and levels above are said to be white. For a high-contrast document with uniform background, a prechosen fixed threshold can be sufficient. However, a lot of documents encountered in practice have a rather large range in contrast. In these cases more sophisticated methods for thresholding arerequired to obtain a good result.

# 5.2 PREPROCESSING DATA

## 5.2.1 LOCATION AND SEGMENTATION

Segmentation is a process that determines the constituents of an image. It is necessary to locate the regions of the document where data have been printed and distinguish them from figures and graphics. For instance, when performing automatic mail-sorting, the address must be located and separated from other print on the envelope like stamps and company logos, prior to recognition.

Applied to text, segmentation is the isolation of characters or words. The majority of optical character recognition algorithms segment the words into isolated characters which are recognized individually. Usually this segmentation is performed by isolating each connected component that is each connected black area. This technique is easy to implement, but problems occur if characters touch or if characters are fragmented and consist of several parts. The main problems in segmentation may be divided into four groups:

• *Extraction of touching and fragmented characters.*

Such distortions may lead to several joint characters being interpreted as one single character, or that a piece of a character is believed to be an entire symbol. Joints will occur if the document is a dark photocopy or if it is scanned at a low threshold. Also joints are common if the fonts are serifed. The characters may be split if the document stems from a light photocopy or is scanned at a high threshold.

• *Distinguishing noise from text.*

Dots and accents may be mistaken for noise, and vice versa.

• *Mistaking graphics or geometry for text.*

This leads to nontext being sent to recognition.

• *Mistaking text for graphics or geometry.*

In this case the text will not be passed to the recognition stage. This often happens if characters are connected to graphics.

## PREPROCESSING

The image resulting from the scanning process may contain a certain amount of noise. Depending on the resolution on the scanner and the success of the applied technique for thresholding, the characters may be smeared or broken. Some of these defects, which may later cause poor recognition rates, can be eliminated by using a preprocessor to smooth the digitized characters.

The smoothing implies both filling and thinning. Filling eliminates small breaks, gaps and holes in the digitized characters, while thinning reduces the width of the line. The most common techniques for smoothing, moves a window across the binary image of the character, applying certain rules to the contents of the window.

In addition to smoothing, preprocessing usually includes normalization. The normalization is applied to obtain characters of uniform size, slant and rotation. To be able to correct for rotation, the angle of rotation must be found. For rotated pages and lines of text, variants of Hough transform are commonly used for detecting skew. However, to find the rotation angle of a single symbol is not possible until after the symbol has been recognized.

## 5.3 FEATURE EXTRACTION

The objective of feature extraction is to capture the essential characteristics of the symbols, and it is generally accepted that this is one of the most difficult problems of pattern recognition. The most straight forward way of describing a character is by the actual raster image. Another approach is to extract certain features that still characterize the symbols, but leaves out the unimportant attributes. The techniques for extraction of such features

are often divided into three main groups, where the features are found from:

• The distribution of points.

• Transformations and series expansions.

• Structural analysis.

The different groups of features may be evaluated according to their sensitivity to noise and deformation and the ease of implementation and use. The criteria used in the comparison and evaluation are the following:

• **Robustness.**

1) *Noise.*

Sensitivity to disconnected line segments, bumps, gaps, filled loops etc.

2) *Distortions.*

Sensitivity to local variations like rounded corners, improper protrusions, dilations and shrinkage.

3) *Style variation.*

Sensitivity to variation in style like the use of different shapes to represent the same character or the use of serifs, slants etc.

4) *Translation.*

Sensitivity to movement of the whole character or its components.

5) *Rotation.*

Sensitivity to change in orientation of the characters.

• **Practical use.**

*1) Speed of recognition.*

*2) Complexity of implementation.*

3) *Independence.*

The need of supplementary techniques.

| Feature extraction technique | Robustness 1 2 3 4 5 | Practical use 1 2 3 |
|---|---|---|
| Template matching | ● ● ○ ○ ○ | ○ ● ○ |
| Transformations | ○ ● ● ● ● | ○ ○ ● |
| Distribution of points: Zoning | ○ ● ○ ○ ● | ● ● ○ |
| Moments | ● ● ○ ● ● | ○ ● ○ |
| n-tuple | ● ○ ● ○ ● | ● ● ● |
| Characteristic loci | ○ ● ● ● ● | ● ● ○ |
| Crossings | ○ ● ● ● ● | ● ● ○ |
| Structural features | ○ ● ● ● ● | ● ○ ● |

● High or easy ● Medium ○ Low or difficult

Comparison of various feature extraction techniques

The above mentioned techniques are described in detail as below:

### 5.3.1 TEMPLATE-MATCHING AND CORRELATION TECHNIQUES.

These techniques are different from the others in that no features are actually extracted. Instead the matrix containing the image of the input character is directly matched with a set of prototype characters representing each possible class. The distance between the pattern and each prototype is computed, and the class of the prototype giving the best match is assigned to the pattern.

The technique is simple and easy to implement in hardware and has been used in many commercial OCR machines. However, this technique is sensitive to noise and style variations and has no way of handling rotated characters.

### 5.3.2 FEATURE BASED TECHNIQUES

In these methods, significant measurements are calculated and extracted from a character and compared to descriptions of the character classes obtained during a training phase.

The description that matches most closely provides recognition. The features are given as numbers in a feature vector, and this feature vector is used to represent the symbol.

### 5.3.3 DISTRIBUTION OF POINTS

This category covers a technique that extracts features based on the statistical distribution of points. These features are usually tolerant to distortions and style variations. Some of the typical techniques within this area are listed below.

### 5.3.4 Zoning

The rectangle circumscribing the character is divided into several overlapping, or no overlapping, regions and the densities of black points within these regions are computed and used as features.

### 5.3.5 Moments

The moments of black points about a chosen centre, for example the centre of gravity, or a chosen coordinate system, are used as features.

### 5.3.6 Crossings and distances

In the crossing technique features are found from the number of times the character shape is crossed by vectors along certain directions. This technique is often used by commercial systems because it can be performed at high speed and requires low complexity. When using the distance technique certain lengths along the vectors crossing the character shape are measured. For instance the length of the vectors within the boundary of the character.

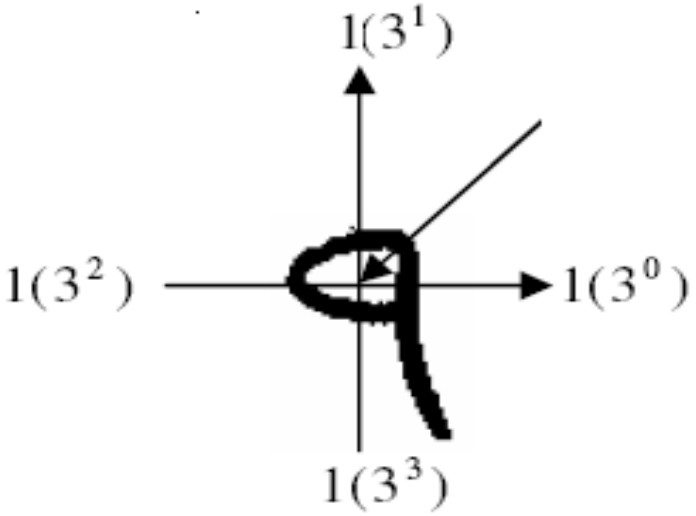
### 5.3.7 N-tuples

The relative joint occurrence of black and white points (foreground and background) in certain specified orderings, are used as features.

## 5.4 Characterization Loci Feature

Characteristic loci features are usually defined in horizontal and vertical orand directions (Glucksman, 1967). In computing feature vectors, we assign a number to each background pixel as shown in Figure 1. The features are computed according to the number of intersections with the sub word body in right, upward, left and downward directions. Then, for each background pixel, a four digit number of base 3 are obtained. For instance, the locus number of point P in Figure 1 is. The locus numbers are between 0 and 80. This is done for all background pixels. In this case, dimension of the feature vectors becomes 81. Each element of this vector represents the total number of background pixels that have locus number corresponding to that element. For example, 56th element of this vector represents the number of background pixels with locus number of 56. Features are normalized by dividing them by the total number of background pixels. The proposed method is described as follow: First, we done feature extraction using Characteristic loci features then recognition of hand written digits is done using mixture of experts.

Illustration of Characteristic loci.




$$(1111)_3 = (40)_{10}$$
$$(1 \times 3^0 + 1 \times 3^1 + 1 \times 3^2 + 1 \times 3^3 = 40)$$

## 5.5 STRUCTURAL ANALYSIS

During structural analysis, features that describe the geometric and topological structures of a symbol are extracted. By these features one attempts to describe the physical makeup of the character, and some of the commonly used features are strokes, bays, end-points, intersections between lines and loops. Compared to other techniques the structural analysis gives features with high tolerance to noise and style variations. However, the features are only moderately tolerant to rotation and translation. Unfortunately, the extractions of these features are not trivial, and to some extent still an area of research.

## CLASSIFICATION

The classification is the process of identifying each character and assigning to it the correct character class. In the following sections two different approaches for

classification in character recognition are discussed. First decision-theoretic recognition is treated.

These methods are used when the description of the character can be numerically represented in a feature vector.

We may also have pattern characteristics derived from the physical structure of the character which are not as easily quantified. In these cases the relationship between the characteristics may be of importance when deciding on class membership. For instance, if we know that a character consists of one vertical and one horizontal stroke, it may be either an “L” or a “T”, and the relationship between the two strokes is needed to distinguish the characters. A structural approach is then needed.

## 5.5 DECISION-THEORETIC METHODS

The principal approaches to decision-theoretic recognition are minimum distance classifiers, statistical classifiers and neural networks. Each of these classification techniques are briefly described below

### 5.5.1 MATCHING

Matching covers the groups of techniques based on similarity measures where the distance between the feature vector, describing the extracted character and the description of each class is calculated. Different measures may be used, but the common is the Euclidean distance. This minimum distance classifier works well when the classes are well separated, that is when the distance between the means is large compared to the spread of each class.

When the entire character is used as input to the classification, and no features are extracted (template-matching), a correlation approach is used. Here the distance between the character image and prototype images representing each character class is computed.

### 5.5.2 OPTIMUM STATISTICAL CLASSIFIERS.

In statistical classification a probabilistic approach to recognition is applied. The idea is to use a classification scheme that is optimal in the sense that, on average, its use gives the lowest probability of making classification errors.

A classifier that minimizes the total average loss is called the Bayes' classifier. Given an unknown symbol described by its feature vector, the probability that the symbol belongs to class *c* is computed for all classes *c*=1...N. The symbol is then assigned the class which gives the maximum probability. For this scheme to be optimal, the probability density functions of the symbols of each class must be known, along with the probability of occurrence of each class. The latter is usually solved by assuming that all classes are equally probable. The density function is usually assumed to be normally distributed, and the closer this assumption is to reality, the closer the Bayes' classifier comes to optimal behaviour. The minimum distance classifier described above is specified completely by the mean vector of each class, and the Bayes classifier for Gaussian classes is specified completely by the mean vector and covariance matrix of each class. These parameters specifying the classifiers are obtained through a training process. During this process, training patterns of each class is used to compute these parameters and descriptions of each class are obtained.

## NEURAL NETWORKS

Recently, the use of neural networks to recognize characters (and other types of patterns) has resurfaced. Considering a back-propagation network, this network is composed of several layers of interconnected elements. A feature vector enters the network at the input layer. Each element of the layer computes a weighted sum of its input and transforms it into an output by a nonlinear function. During training the weights at each connection are adjusted until a desired output is obtained. A problem of neural networks in OCR may be their limited predictability and generality, while an advantage is their adaptive nature.

## Structural Methods

Within the area of structural recognition, syntactic methods are among the most prevalent approaches. Other techniques exist, but they are less general and will not be treated here.

## SYNTACTIC METHODS

Measures of similarity based on relationships between structural components may be formulated by using grammatical concepts. The idea is that each class has its own grammar defining the composition of the character.A grammar may be represented as strings or trees, and the structural components extracted from an unknown character is matched against the grammars of each class. Suppose that we have two different character classes which can be generated by the two grammars $G_1$ and $G_2$, respectively. Given an unknown character, we say that it is more similar to the first class if it may be generated by the grammar $G_1$, but not by $G_2$.

# 5.6 POST PROCESSING

### 5.6.1 GROUPING

The result of plain symbol recognition on a document, is a set of individual symbols. However, these symbols in themselves do usually not contain enough information. Instead we would like to associate the individual symbols that belong to the same string with each other, making up words and numbers. The process of performing this association of symbols into strings, is commonly referred to as grouping. The grouping of the symbols into strings is based on the symbols' location in the document. Symbols that are found to be sufficiently close are grouped together.
For fonts with fixed pitch the process of grouping is fairly easy as the position of each character is known. For typeset characters the distance between characters are variable. However, the distance between words are usually significantly larger than the distance between characters, and grouping is therefore still possible. The real problems occur for handwritten characters or when the text is skewed.

### 5.6.2 Error-detection and correction

Up until the grouping each character has been treated separately, and the context in which each character appears has usually not been exploited. However, in advanced optical text recognition problems, a system consisting only of single-character recognition will not be sufficient. Even the best recognition systems will not give 100% percent correct identification of all characters, but some of these errors may be detected or even corrected by the use of context.

There are two main approaches, where the first utilizes the possibility of sequences of characters appearing together. This may be done by the use of rules defining the syntax of the word, by saying for instance that after a period there should usually be a capital letter. Also, for different languages the probabilities of two or more characters appearing together in a sequence can be computed and may be utilized to detect errors. For instance, in the English language the probability of a “k” appearing after an “h” in a word is zero, and if such a combination is detected an error is assumed.

Another approach is the use of dictionaries, which has proven to be the most efficient method for error detection and correction. Given a word, in which an error may be present, the word is looked up in the dictionary. If the word is not in the dictionary, an error has been detected, and may be corrected by changing the word into the most similar word. Probabilities obtained from the classification, may help to identify the character which has been erroneously classified. If the word is present in the dictionary, this does unfortunately not prove that no error occurred. An error may have transformed the word from one legal word to another, and such errors are undetectable by this procedure. The disadvantage of the dictionary methods is that the searches and comparisons implied are time-consuming.

# 6 HARDWARE USED

## 6.1 WEBCAM OR VGA CAMERA

A webcam is a video camera that feeds its images in real time to a computer or computer network, often via USB, ethernet, or Wi-Fi.

Their most popular use is the establishment of video links, permitting computers to act as videophones or videoconference stations. The common use as a video camera for the World Wide Web gave the webcam its name. Other popular uses include security surveillance, computer vision, video broadcasting and for recording social videos .

Webcams are known for their low manufacturing cost and flexibility,[1] making them the lowest cost form of videotelephony. They have also become a source of security and privacy issues, as some built-in webcams can be remotely activated via spyware.

Figure 3 : A VGA Camera

## TECHNOLOGY

Webcams typically include a lens, an image sensor, support electronics, and may also include a microphone for sound. Various lenses are available, the most common in consumer-grade webcams being a plastic lens that can be screwed in and out to focus the camera. Fixed focus lenses, which have no provision for adjustment, are also available. As a camera system's depth of field is greater for small image formats and is greater for lenses with a large f-number (small aperture), the systems used in webcams have a sufficiently large depth of field that the use of a fixed focus lens does not impact image sharpness to a great extent.

Image sensors can be CMOS or CCD, the former being dominant for low-cost cameras, but CCD cameras do not necessarily outperform CMOS-based cameras in the low cost price range. Most consumer webcams are capable of providing VGA resolution video at a frame rate of 30 frames per second. Many newer devices can produce video in multi-megapixel resolutions, and a few can run at high frame rates such as the PlayStation Eye, which can produce 320×240 video at 120 frames per second.

Support electronics read the image from the sensor and transmit it to the host computer. The camera pictured to the right, for example, uses a Sonix SN9C101 to transmit its image over USB. Some cameras, such as mobile phone cameras, use a CMOS sensor with supporting electronics "on die", i.e. the sensor and the support electronics are built on a single silicon chip to save space and manufacturing costs. Most webcams feature built-in microphones to make video calling and videoconferencing more convenient.

The USB video device class (UVC) specification allows for interconnectivity of webcams to computers without the need for proprietary device drivers. Microsoft Windows XP SP2, Linux[12] and Mac OS X (since October 2005) have UVC support built in and do not require extra device drivers, although they are often installed to add additional features.

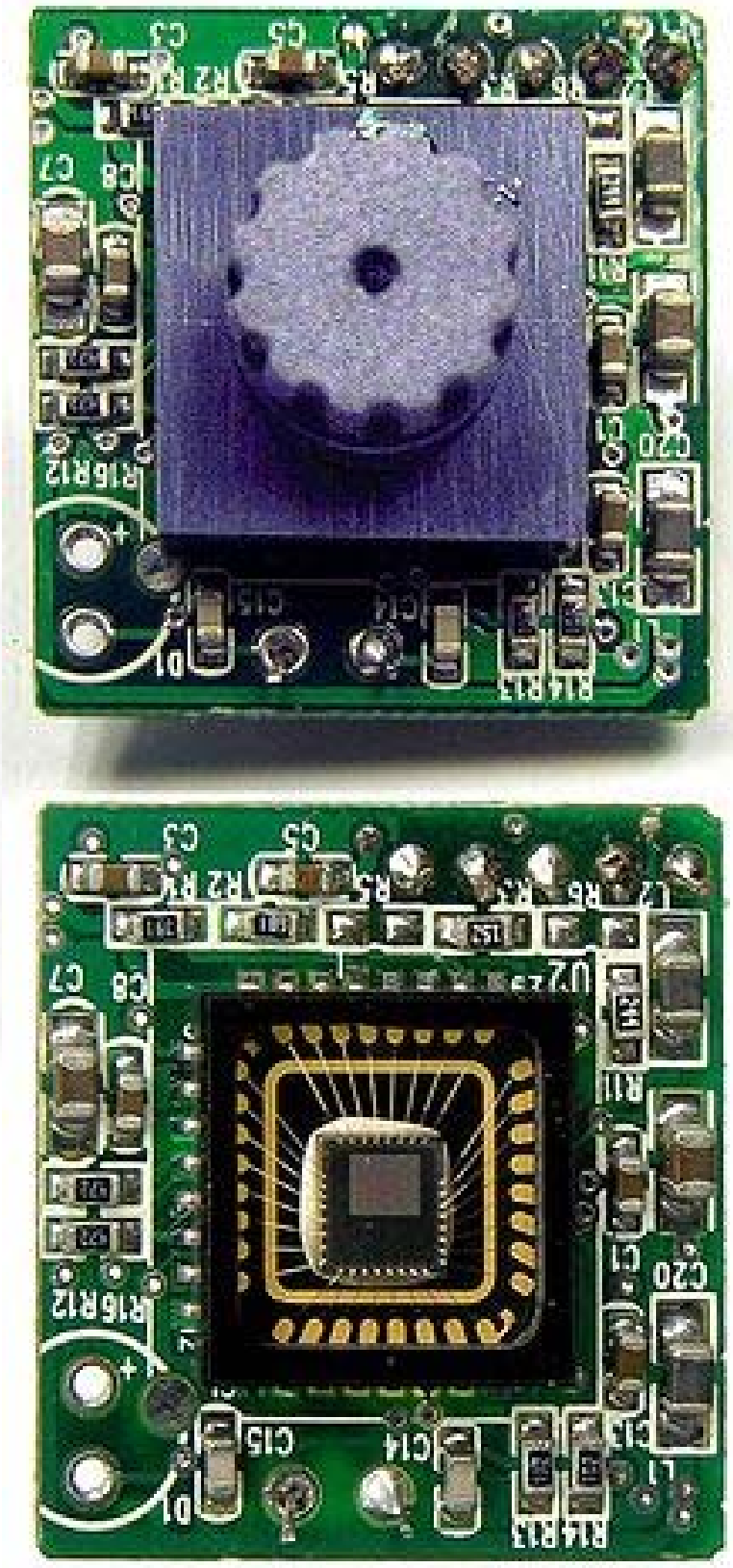


Figure 4 : Interior of a VGA Camera

# 7 SOFTWARE USED

## 7.1 MATLAB

MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing capabilities. An additional package, Simulink, adds graphical multi-domain simulation and Model-Based Design for dynamic and embedded systems.

In 2004, MATLAB had around one million users across industry and academia. MATLAB users come from various backgrounds of engineering, science, and economics. MATLAB is widely used in academic and research institutions as well as industrial enterprises.

### 7.1 SYNTAX

The MATLAB application is built around the MATLAB language, and most use of MATLAB involves typing MATLAB code into the Command Window (as an interactive mathematical shell), or executing text files containing MATLAB code and functions.

### VARIABLES

Variables are defined using the assignment operator, =. MATLAB is a dynamically typed programming language. It is a weakly typed language because types are implicitly converted. It is a dynamically typed language because variables can be assigned without declaring their type, except if they are to be treated as symbolic objects, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function.

For example:

```
>> x = 17
x = 17
>> x = 'hat'
x =hat
>> y = x + 0
y =    104        97       116
>> x = [3*4, pi/2]
x = 12.0000    1.5708
>> y = 3*sin(x)
y = -1.6097    3.0000
```

## VECTORS/MATRICES

As suggested by its name (a contraction of "Matrix Laboratory"), MATLAB can create and manipulate arrays of 1 (vectors), 2 (matrices), or more dimensions. In the MATLAB vernacular, a *vector* refers to a one dimensional (1×*N* or *N*×1) matrix, commonly referred to as an array in other programming languages. A *matrix* generally refers to a 2-dimensional array, i.e. an *m*×*n* array where *m* and *n* are greater than 1. Arrays with more than two dimensions are referred to as multidimensional arrays. Arrays are a fundamental type and many standard functions natively support array operations allowing work on arrays without explicit loops. Therefore the MATLAB language is also an example of array programming language.

A simple array is defined using the syntax: *init*:*increment*:*terminator*. For instance:

```
>> array = 1:2:9
array =
 1 3 5 7 9
```

defines a variable named `array` (or assigns a new value to an existing variable with the name `array`) which is an array consisting of the values 1, 3, 5, 7, and 9. That is, the array starts at 1 (the *init* value), increments with each step from the previous value

by 2 (the *increment* value), and stops once it reaches (or to avoid exceeding) 9 (the *terminator* value).

```
>> array = 1:3:9
array = 1 4 7
```

the *increment* value can actually be left out of this syntax (along with one of the colons), to use a default value of 1.

```
>> ari = 1:5
ari = 1 2 3 4 5
```

assigns to the variable named `ari` an array with the values 1, 2, 3, 4, and 5, since the default value of 1 is used as the incrementer.

Indexing is one-based, which is the usual convention for matrices in mathematics, although not for some programming languages.

Matrices can be defined by separating the elements of a row with blank space or comma and using a semicolon to terminate each row. The list of elements should be surrounded by square brackets: []. Parentheses: () are used to access elements and subarrays (they are also used to denote a function argument list).

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =16  3  2 13
   5 10 11  8
   9  6  7 12
   4 15 14  1

>> A(2,3)
ans = 11
```

Sets of indices can be specified by expressions such as "2:4", which evaluates to [2, 3, 4]. For example, a submatrix taken from rows 2 through 4 and columns 3 through 4 can be written as:

```
>> A(2:4,3:4)
ans = 11 8
      7 12
      14 1
```

A square identity matrix of size *n* can be generated using the function *eye*, and matrices of any size with zeros or ones can be generated with the functions *zeros* and *ones*, respectively.

```
>> eye(3)

ans =
 1 0 0
 0 1 0
 0 0 1

>> zeros(2,3)

ans =
 0 0 0
 0 0 0

>> ones(2,3)

ans =
 1 1 1
 1 1 1
```

## SEMICOLONS

Unlike many other languages, where the semicolon is used to terminate commands, in MATLAB the semicolon serves to suppress the output of the line that it concludes (it serves a similar purpose in Mathematica). Commands that have return values would be: numbers/vectors/matrices and various mathematical functions executed with these (addition, multiplication etc.), strings and strings functions etc. Each number/vector/matrix/string/function with a return value, if it appears in a line not terminated by a semicolon, will have its value displayed on the screen once the line is interpreted. Some Matlab commands (such as the graphical "plot" command), however, will not have any return value associated with them, in which case the semicolon is redundant in that sense.

However, a semicolon, as it symbolizes the end of a command, can allow another command to be listed in the same line. That will hardly be used at all in a Matlab program file (i.e., a Matlab file with extension ".m") which typically groups several commands, each in a separate line - for better "readability"; but it could be handy when typing commands in the command line prompt window in the desktop, whereby each line is interpreted and executed immediately after the pressing of the *enter* key

(and therefore the semicolon allows to type a command and not execute it before typing another one).

## 7.2 STRUCTURES

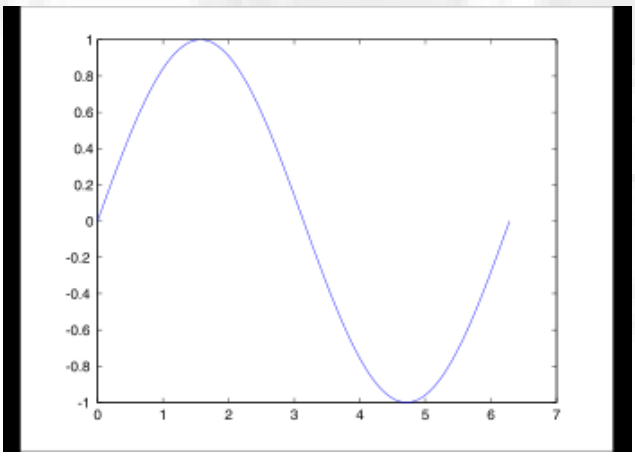
MATLAB supports structure data types. Since all variables in MATLAB are arrays, a more adequate name is "structure array", where each element of the array has the same field names. In addition, MATLAB supports dynamic field names (field look-ups by name, field manipulations etc). Unfortunately, MATLAB JIT does not support MATLAB structures, therefore just a simple bundling of various variables into a structure will come at a cost.

### Graphics and graphical user interface programming

MATLAB supports developing applications with graphical user interface features. It also has tightly-integrated graph-plotting features. For example the function *plot* can be used to produce a graph from two vectors *x* and *y*. The code:

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```

produces the following sine function




Three-dimensional graphics can be produced using the functions *surf*, *plot3* or *mesh*.

```
[X,Y] = meshgrid(-10:0.25:10,-10:0.25:10);
```

```
f = sinc(sqrt((X/pi).^2+(Y/pi).^2));
mesh(X,Y,f);
axis([-10 10 -10 10 -0.3 1])
xlabel('{\bfx}')
ylabel('{\bfy}')
zlabel('{\bfsinc} ({\bfR})')
hidden off
```

This code produces a wireframe 3D plot of the two-dimensional unnormalized sinc function

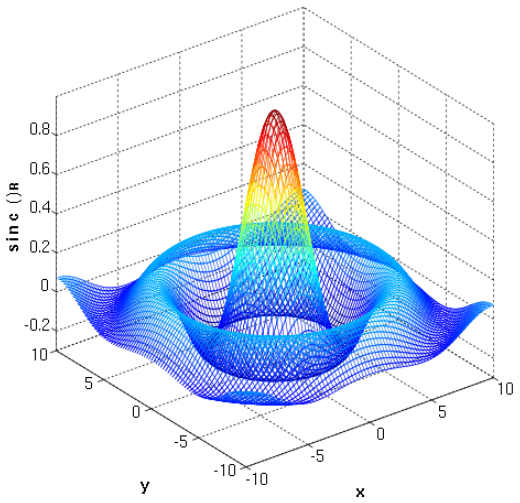



Figure 5 : 3D plot of the two-dimensional unnormalized sinc function

```
[X,Y] = meshgrid(-10:0.25:10,-10:0.25:10);
f = sinc(sqrt((X/pi).^2+(Y/pi).^2));
surf(X,Y,f);
axis([-10 10 -10 10 -0.3 1])
xlabel('{\bfx}')
ylabel('{\bfy}')
zlabel('{\bfsinc} ({\bfR})')
```

This code produces a surface 3D plot of the two-dimensional unnormalized sinc function.

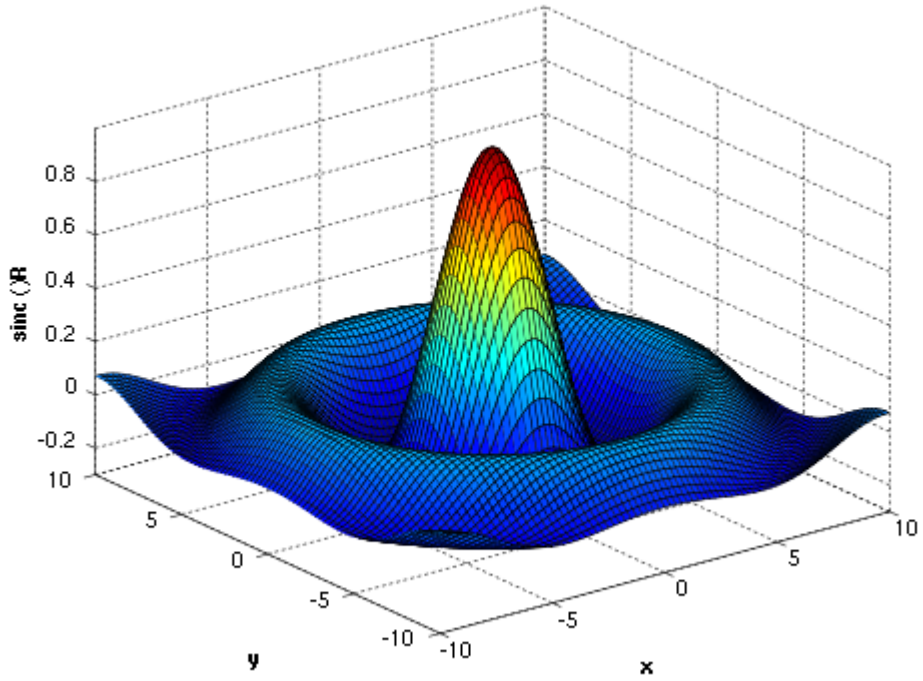



Figure 6 : Surface 3D plot of the two dimenasional unnormalizedsinc function

## INTERFACING WITH OTHER LANGUAGES

MATLAB can call functions and subroutines written in the C programming language or Fortran. A wrapper function is created allowing MATLAB data types to be passed and returned. The dynamically loadable object files created by compiling such functions are termed "MEX-files" (for MATLAB executable).Libraries written in Java, ActiveX or .NET can be directly called from MATLAB and many MATLAB libraries (for example XML or SQL support) are implemented as wrappers around Java or ActiveX libraries. Calling MATLAB from Java is more complicated, but can be done with MATLAB extension,which is sold separately by MathWorks, or using an undocumented mechanism called JMI (Java-to-Matlab Interface), which should not be confused with the unrelated Java Metadata Interface that is also called JMI.As alternatives to the MuPAD based Symbolic Math Toolbox available from MathWorks, MATLAB can be connected to Maple or Mathematica. Libraries also exist to import and export MathML.

## ALTERNATIVES

MATLAB has a number of competitors.[19] Commercial competitors include Mathematica, Maple, IDL by ITT Visual Information Solutions and Metlynx.
There are also free open source alternatives to MATLAB, in particular GNU Octave, FreeMat, and Scilab which are intended to be mostly compatible with the MATLAB language (but not the MATLAB desktop environment).

Among other languages that treat arrays as basic entities (array programming languages) are APL and J, Fortran 95 and 2003, as well as the statistical language S (the main implementations of S are S-PLUS and the popular open source language R).

There are also several libraries to add similar functionality to existing languages, such as IT++ for C++, Perl Data Language for Perl, ScalaLab for Scala and SciPy together with NumPy and Matplotlib for Python.

## 7.3 SVM - Support Vector Machines

### Introduction to Support Vector Machine (SVM) Models

A Support Vector Machine (SVM) performs classification by constructing an *N*-dimensional plane that optimally separates the data into two categories. SVM models are closely related to neural networks. In fact, a SVM model using a sigmoid kernel function is equivalent to a two-layer, perceptron neural network.

Support Vector Machine (SVM) models are a close cousin to classical multilayer perceptron neural networks. Using a kernel function, SVM's are an alternative training method for polynomial, radial basis function and multi-layer perceptron classifiers in which the weights of the network are found by solving a quadratic programming problem with linear constraints, rather than by solving a non-convex, unconstrained minimization problem as in standard neural network training.

In the parlance of SVM literature, a predictor variable is called an *attribute*, and a transformed attribute that is used to define the plane is called a *feature*. The task of choosing the most suitable representation is known as *feature selection*. A set of features that describes one case (i.e., a row of predictor values) is called a *vector*. So the goal of SVM modeling is to find the optimal plane that separates clusters of vector in such a way that cases with one category of the target variable are on one side of the plane and cases with the other category are on the other size of the plane. The vectors near the plane are the *support vectors*. The figure below presents an overview of the SVM process.

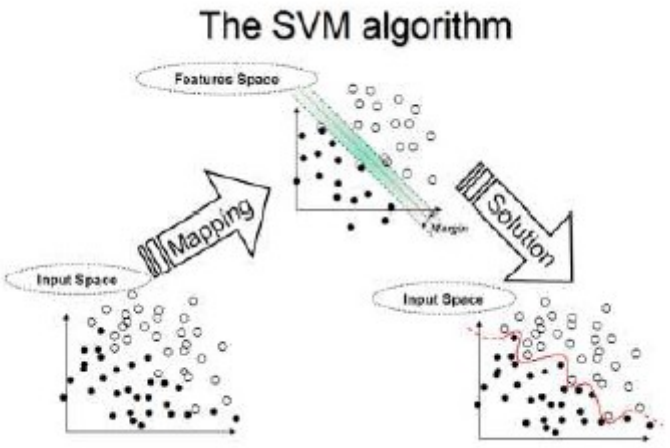



Figure 7: A Two-Dimensional Example of SVM

Before considering *N*-dimensional planes, let's look at a simple 2-dimensional example. Assume we wish to perform a classification, and our data has a categorical target variable with two categories. Also assume that there are two predictor variables with continuous values. If we plot the data points using the value of one predictor on the X axis and the other on the Y axis we might end up with an image such as shown below. One category of the target variable is represented by rectangles while the other category is represented by ovals.

In this idealized example, the cases with one category are in the lower left corner and the cases with the other category are in the upper right corner; the cases are completely separated. The SVM analysis attempts to find a 1-dimensional plane (i.e. a line) that separates the cases based on their target categories. There are an infinite number of possible lines; two candidate lines are shown above. The question is which line is better, and how do we define the optimal line.

The dashed lines drawn parallel to the separating line mark the distance between the dividing line and the closest vectors to the line. The distance between the dashed lines is called the *margin*. The vectors (points) that constrain the width of the margin are the *support vectors*. The following figure illustrates this.

An SVM analysis finds the line (or, in general, plane) that is oriented so that the margin between the support vectors is maximized. In the figure above, the line in the right panel is superior to the line in the left panel.

If all analyses consisted of two-category target variables with two predictor variables, and the cluster of points could be divided by a straight line, life would be easy. Unfortunately, this is not generally the case, so SVM must deal with (a) more than two predictor variables, (b) separating the points with non-linear curves, (c) handling the cases where clusters cannot be completely separated, and (d) handling classifications with more than two categories.

## FLYING HIGH ON PLANES

In the previous example, we had only two predictor variables, and we were able to plot the points on a 2-dimensional plane. If we add a third predictor variable, then we can use its value for a third dimension and plot the points in a 3-dimensional cube. Points on a 2-dimensional plane can be separated by a 1-dimensional line. Similarly, points in a 3-dimensional cube can be separated by a 2-dimensional plane.

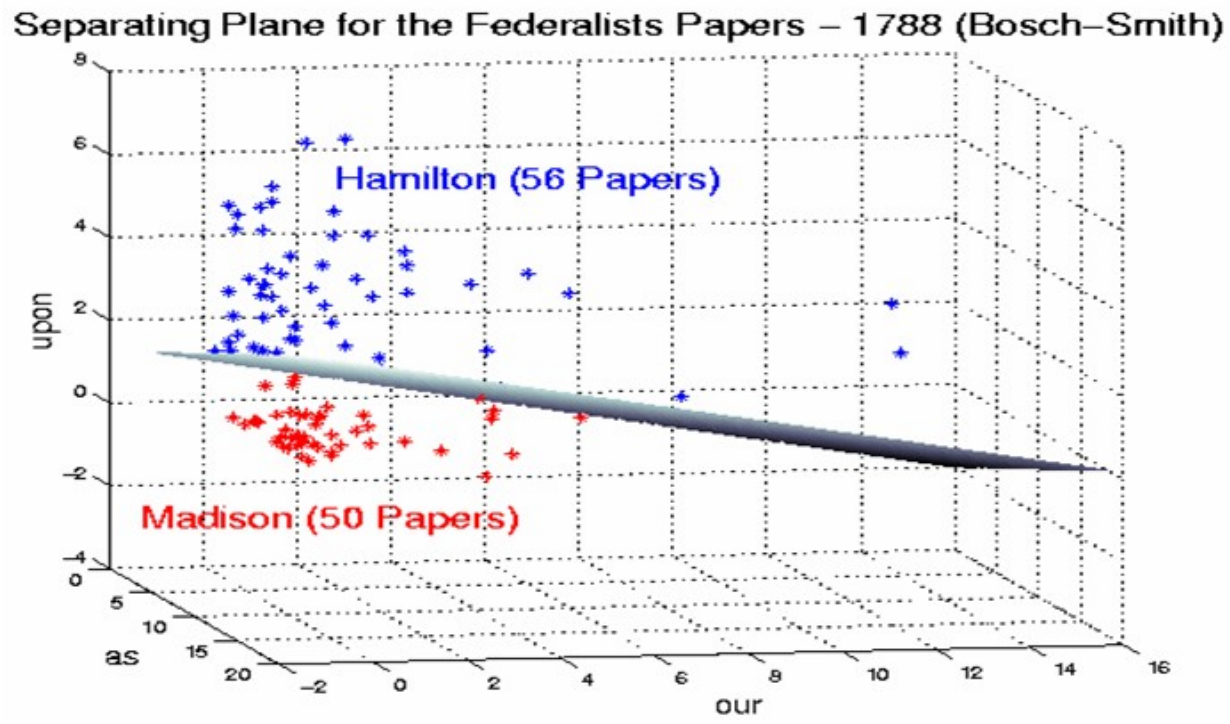



Figure 8 : An example to show classification of data using SVM

As we add additional predictor variables (attributes), the data points can be represented in *N*-dimensional space, and a (N-1)-dimensional plane can separate them.

## WHEN STRAIGHT LINES GO CROOKED

The simplest way to divide two groups is with a straight line, flat plane or an N-dimensional plane. But what if the points are separated by a nonlinear region such as shown below

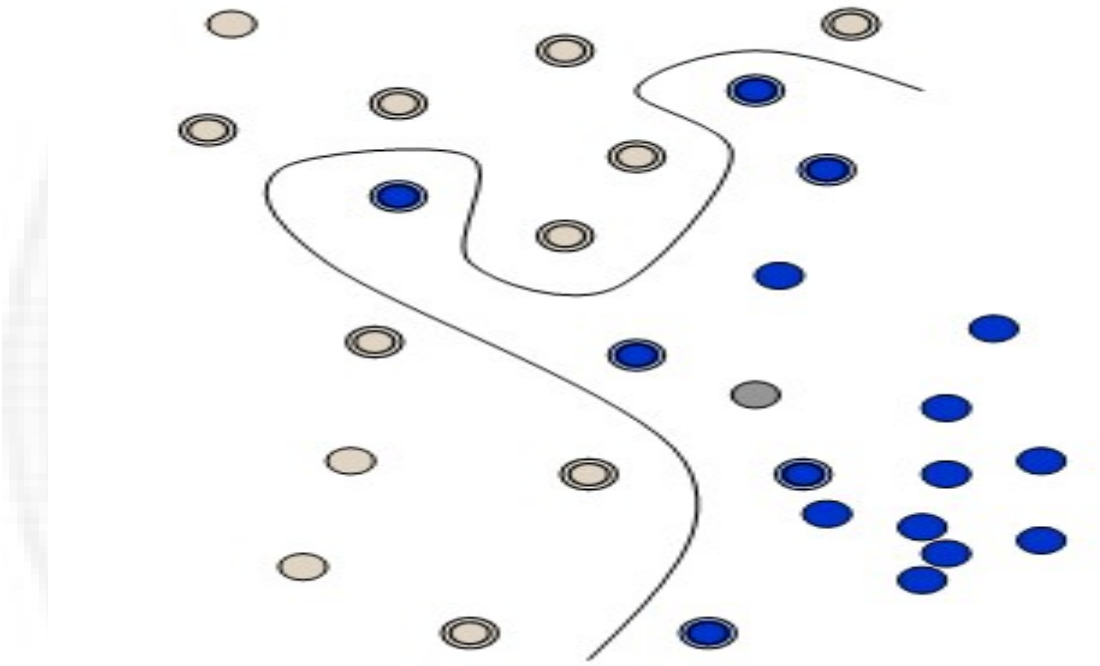


Figure 9 : SVM using non linear dividing line.

In this case we need a nonlinear dividing line.

Rather than fitting nonlinear curves to the data, SVM handles this by using a *kernel function* to map the data into a different space where a plane can be used to do the separation.

# 8 PROGRAMING

```
function varargout = take_snaps(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @take_snaps_OpeningFcn, ...
                   'gui_OutputFcn',  @take_snaps_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before take_snaps is made visible.
function take_snaps_OpeningFcn(hObject, eventdata, handles, varargin)

% Choose default command line output for take_snaps
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = take_snaps_OutputFcn(hObject, eventdata,
handles)
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in push_takesnap.
function push_takesnap_Callback(hObject, eventdata, handles)
% hObject    handle to push_takesnap (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global reset vid
global I
% ============ Image 1=========================
axes(handles.axes1);
% trigger(vid);
I=getdata(vid,1); % Get the frame in im
imshow(I);
imwrite(I,'test.bmp');
% ================================================
reset = 0;
```

```
stop(vid)
clear vid

% Update gui structure
guidata(hObject,handles)

% --- Executes on button press in push_done.
function push_done_Callback(hObject, eventdata, handles)
% hObject    handle to push_done (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(handles.figure1);

% --- Executes on button press in push_startcamera.
function push_startcamera_Callback(hObject, eventdata, handles)
% hObject    handle to push_startcamera (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global vid reset
if strcmp(get(hObject,'string'),'Stop Camera')
     reset=0;
    stop(vid)
    set(hObject,'string','Start Camera')
elseif strcmp(get(hObject,'string'),'Start Camera')
    reset=1;
    set(hObject,'string','Stop Camera')
    delete(imaqfind)
    vid=videoinput('winvideo',1,'YUY2_640x480');
    %  View the default color space used for the data – The value of
the ReturnedColorSpace property indicates the color space of the
image data.
    color_spec=vid.ReturnedColorSpace;

    % Modify the color space used for the data – To change the color
space of the returned image data, set the value of the
ReturnedColorSpace property.
    if  ~strcmp(color_spec,'rgb')
        set(vid,'ReturnedColorSpace','rgb');
    end

    axes(handles.axes1);
    triggerconfig(vid,'manual');
    set(vid,'FramesPerTrigger',1 );
    set(vid,'TriggerRepeat', Inf);
    start(vid);
    while reset
        trigger(vid);
        im=getdata(vid,1); % Get the frame in im
        axes(handles.axes1)
        imshow(im);


    end
end
```

```
% --- Executes on button press in push_train.
function push_train_Callback(hObject, eventdata, handles)
% hObject    handle to push_train (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%%
% Toolbox website
% OSU SVM Classifier Matlab Toolbox
%  http://www.ece.osu.edu/~maj/osu_svm/
% This MATLAB toolbox is based on LIBSVM.
%
h1 = waitbar(0,'Please wait while training the classifier');
addpath('SVM');
Str =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
'R','S'...

,'T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9'};
Samples = [];
Labels = [];
for ii = 1:length(Str);
    waitbar(ii/length(Str));
    fpath = ['Database/' Str{ii} '/'];
    D = dir(fpath);
    for nn = 3:length(D);
        impath = [fpath '/' D(nn).name];
        if strcmp(impath(end-3:end),'.bmp')
            im = imread(impath);
            im = ~im;
            % Calculate feature vector
            loci_vector = calculate_characterstic_loci(im);

            % Append feature vector
            Samples = [Samples loci_vector];

            % Append respective labels
            Labels  = [Labels ii];
        end

    end
end

[AlphaY, SVs, Bias, Parameters, nSV, nLabel] = LinearSVC(Samples,
Labels);
SVM.AlphaY=AlphaY;
SVM.SVs=SVs;
SVM.Bias=Bias;
SVM.Parameters=Parameters;
SVM.nSV=nSV;
SVM.nLabel=nLabel;
close(h1)
save SVM SVM
msgbox('Training is done')

% --- Executes on button press in push_ocr.
function push_ocr_Callback(hObject, eventdata, handles)
% hObject    handle to push_ocr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
addpath('SVM');
load SVM
Str =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
'R','S'...

,'T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9'};


im = imread('test.bmp');

% I = imread('roi.bmp');
% imshow(I);
I = rgb2gray(im);
bw = multiedge(I,2);
axes(handles.axes1)
imshow(im)
% remove borders
bw(1:2,:)=[];
bw(end-1:end,:)=[];
bw(:,1:2) = [];
bw(:,end-1:end) = [];
% bw = bwmorph(bw,'close');
bw = bwmorph(bw,'bridge');
bw = imfill(bw,'holes');
bw = bwareaopen(bw,50);

bw = clear_borders(bw);

% LAbel
[L count] = bwlabel(bw);

% SEgment lines
S = sum(bw,2);
S(S<50) = 0;

% For each group find

% For every group find peak
[l1 c1] = bwlabel(S);
peaks = [];
for ii = 1:c1
    vals = S.*(l1==ii);
    [~,ix] = max(vals);
    peaks =[peaks ix];

end
% figure;
fid = fopen('out.txt','wt');
% For every line find one by one word and recognise it
for ii = 1:length(peaks)
    % extract the row w.r.t peak
    R = L(peaks(ii),:);

    % find the non zero terms in the row
    nonz = R(R~=0);
    % take unique values
    Uni = unique(nonz);
```

```
    last_col = 0;
    str = [];
    % for every label extract the digit and recognize it

    for nn = 1:length(Uni)

        bw1 = L==Uni(nn);
        % extract letter
        [row col] = find(bw1);

        if min(col)-last_col>20 && last_col~=0;
            str = [str ' '];
        end
        last_col = max(col);
        W = I(min(row):max(row),min(col):max(col));
        th = graythresh(W);
        W = ~im2bw(W,th);

        loci_vector = calculate_characterstic_loci(W);

        AlphaY=SVM.AlphaY;
        SVs=SVM.SVs;
        Bias=SVM.Bias;
        Parameters=SVM.Parameters;
        nSV=SVM.nSV;
        nLabel=SVM.nLabel;
        S1=loci_vector;
        [L1, DecisionValue]= SVMClass(S1, AlphaY, SVs, Bias,
Parameters, nSV, nLabel);
        str =[ str Str{L1}];


    end
    fprintf(fid,'%s\n',str);
    disp(str)


end
fclose(fid)
winopen('out.txt')


% --- Executes on button press in push_addDB.
function push_addDB_Callback(hObject, eventdata, handles)
% hObject    handle to push_addDB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



Str =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
'R','S'...

,'T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9'};
% % Prompt user to select the image
% [fn fp] = uigetfile('*.jpg;*.bmp;*.png','Test image');
```

```
%
% % Concatinate filename and filepath to create one image path
% impath = [fp fn];

% Read image
im = imread('test.bmp');

% % Crop the roi
% I = imcrop(im);
I = im;


% Convert image to gray scale
I = rgb2gray(I);

% --------------option 1 -----------------------------
% Detect edges of the image
bw = multiedge(I,2);
% ---------------option 2 --------------------------
% th = graythresh(I);
% bw = im2bw(I,th);
% ------------------------------------------------

% remove borders
% first two row delete
bw(1:2,:)=[];
% Last two row delete
bw(end-1:end,:)=[];
% first two column delete
bw(:,1:2) = [];
% Last two column delete
bw(:,end-1:end) = [];

% Perfrom morphological operations
bw = bwmorph(bw,'bridge');
% bw = bwmorph(bw,'dilate');
bw = imfill(bw,'holes');

% Remove smaller areas (if more noise is present then increase the
% threhold)
bw = bwareaopen(bw,50);

bw = clear_borders(bw);
%%
% LAbel
[L count] = bwlabel(bw);

% SEgment lines
S = sum(bw,2);

% Apply threhold ( change the value as per the noise);
S(S<50) = 0;

% For each line group find the peak

% For every group find peak
[l1 c1] = bwlabel(S);
peaks = [];
for ii = 1:c1
```

```
    vals = S.*(l1==ii);
    % Find maximum value in respective line
    [~,ix] = max(vals);
    peaks =[peaks ix];
end

load idx
% For every line find one by one word and recognise it
for ii = 1:length(peaks)
    % extract the row w.r.t peak
    R = L(peaks(ii),:);

    % find the non zero terms in the row
    nonz = R(R~=0);

    % take unique values
    Uni = unique(nonz);

     % for every label extract the digit and recognize it
    for nn = 1:length(Uni)
        bw1 = L==Uni(nn);

        % find non zero terms in binary
        [row col] = find(bw1);

        W = I(min(row):max(row),min(col):max(col));
        th = graythresh(W);
        W = im2bw(W,th);

        imwrite(W,['Database/' num2str(idx) '.bmp'])
        idx = idx+1;
    end

end
save idx idx
msgbox('The images are copied in Database folder')

% --- Executes on button press in push_segmentation.
function push_segmentation_Callback(hObject, eventdata, handles)
% hObject    handle to push_segmentation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Str =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
'R','S'...

,'T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9'};
% % Prompt user to select the image
% [fn fp] = uigetfile('*.jpg;*.bmp;*.png','Test image');
%
% % Concatinate filename and filepath to create one image path
% impath = [fp fn];

% Read image
im = imread('test.bmp');

% % Crop the roi
% I = imcrop(im);
I = im;
```

```
% Show the image
imshow(I);

% Convert image to gray scale
I = rgb2gray(I);

% --------------option 1 ------------------------------
% Detect edges of the image
bw = multiedge(I,2);
% ---------------option 2 -------------------------
% th = graythresh(I);
% bw = im2bw(I,th);
% ------------------------------------------------


figure;
imshow(bw)
title('Edge detection');

% remove borders
% first two row delete
bw(1:2,:)=[];
% Last two row delete
bw(end-1:end,:)=[];
% first two column delete
bw(:,1:2) = [];
% Last two column delete
bw(:,end-1:end) = [];

% Perfrom morphological operations
bw = bwmorph(bw,'bridge');
% bw = bwmorph(bw,'dilate');
bw = imfill(bw,'holes');

% Remove smaller areas (if more noise is present then increase the
% threhold)
bw = bwareaopen(bw,50);
figure;
imshow(bw)
title('Morphological operation')


bw = clear_borders(bw);
%%
% LAbel
[L count] = bwlabel(bw);
figure;
imshow(L,[])
title('LAbeling area');

% SEgment lines
S = sum(bw,2);
figure;
plot(S);
title('Row wise summation')

% Apply threhold ( change the value as per the noise);
S(S<50) = 0;
```

```
% For each line group find the peak
figure;
imshow(bw)
hold on
% For every group find peak
[l1 c1] = bwlabel(S);
peaks = [];
for ii = 1:c1
    vals = S.*(l1==ii);
    % Find maximum value in respective line
    [~,ix] = max(vals);
    peaks =[peaks ix];
    plot([1 size(bw,2)],[ix ix],'r-');
    pause(0.5)
end

% For every line find one by one word and recognise it
for ii = 1:length(peaks)
    % extract the row w.r.t peak
    R = L(peaks(ii),:);

    % find the non zero terms in the row
    nonz = R(R~=0);

    % take unique values
    Uni = unique(nonz);

     % for every label extract the digit and recognize it
    for nn = 1:length(Uni)
        bw1 = L==Uni(nn);

        % find non zero terms in binary
        [row col] = find(bw1);

        W = I(min(row):max(row),min(col):max(col));
        th = graythresh(W);
        W = im2bw(W,th);
        rectangle('position',[min(col) min(row) max(col)-min(col)
max(row)-min(row)],'Edgecolor','g')


    end

end


% --- Executes on button press in push_browse.
function push_browse_Callback(hObject, eventdata, handles)
% hObject    handle to push_browse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
[fn fp] = uigetfile('*.jpg;*.bmp;*.png','Test image');
impath = [fp fn];
im = imread(impath);
axes(handles.axes1)
imshow(im)

imwrite(im,'test.bmp')
```

# 9 SIMULATION RESULTS

Test sample


Edge detection

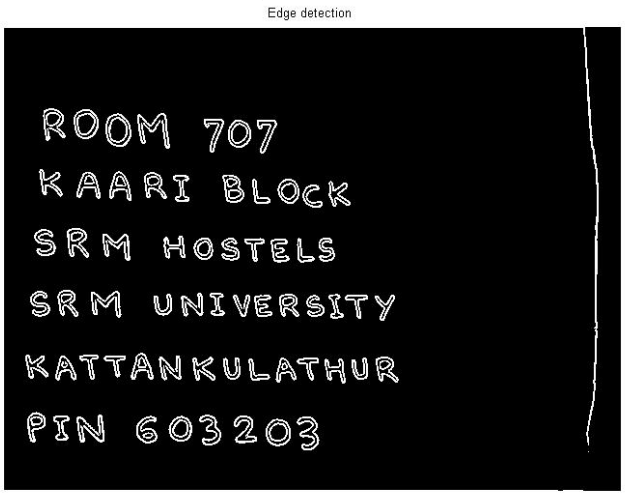
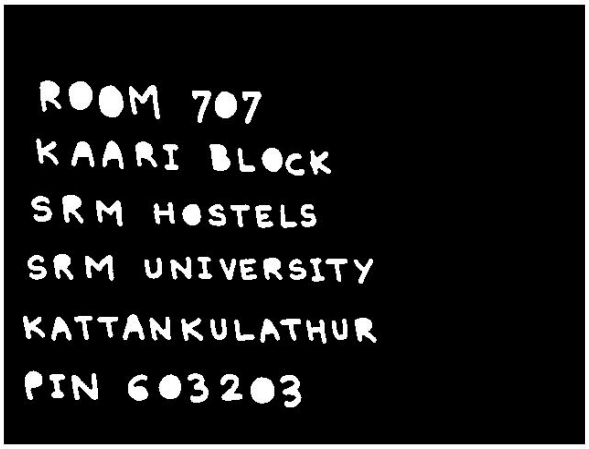


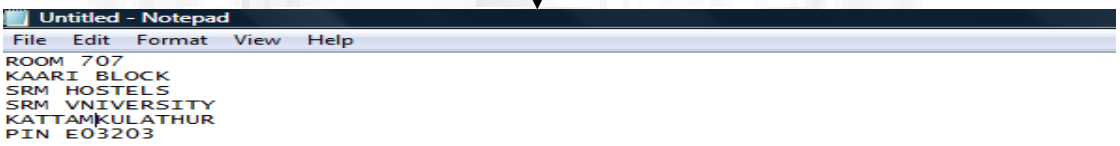

Image morphing



Labelling


Bounding box


**FINAL RESULT**



# 10 CONCLUSION AND FUTURE EXPANSION OF OCR TECHNOLOGY

By using the methods of characterization loci and SVM toolbox we were able to achieve an accuracy in the range of 60-70 % on untrained data for online character recognition which dramatically increased over 85% when the same data was trained into the database.

The coding used for noise reduction was also successful and capable of removing stray marks on the sheet of paper being used as well as the other noises that came while taking the picture were also removed automatically.

Today's uses of OCR are still somewhat limited to the scanning of the written word into useable computer text. The uses include word processing, mail delivery system scanning, ticket reading, and other such tasks.

In the future, the uses of OCR have been speculated to be far more advanced. Some of these advancements are already in the workings. Most believe that the use for reading the written text will diminish as electronic data interchange (EDI) is used more and more, while paper documents are phased out.

Many actually debate whether or not paper will even exist in the next several decades. However, a recent AIIM study found that imaging increased paper consumption because of increased printing. While the debate carries on, it is clear that advancements will lead OCR to greater heights than before.

OCR is already used to detect viruses, or unfortunately create them, and to stop spammers. Anti- spamming applications continue to be improved, as the need for increased technology is a constant. OCR is used to prevent viruses by detecting codes hidden in images.

While the difficulty of transferring information from legacy systems to modern operating systems can be cumbersome, OCR is able to read screenshots. This can facilitate the transferring of information between incompatible technologies.

One of the current hopes for OCR is the chance of developing OCR software that can read compressed files. Text that is compressed into an image and saved as ASCII or Hexadecimal data could be read by OCR and transferred back into readable text.

Finally, it is anticipated that OCR will be used in the development of more advanced robotics. The eyes of a robot are essentially a camera meant to input information. If OCR is used to help the robot comprehend text, the uses could be almost endless. All of these advancements and more are expected to keep the technology of OCR in use and continue to expand its current capabilities.

# 11 REFERENCES


WWW.GOOGLE.COM

WWW.MATHWORKS.COM

MATLAB HELP

MATHWORKS FILE SHARE

WIKIPEDIA

IEEE Paper on “Neural Network based Handwritten Character Recognition system without feature extraction” by J.Pradeep, E.Srinivasan, S.Himavathi

IEEE Paper on “Recognition of Persian handwritten digits using Characterization Loci and Mixture of Experts” by Mohammad R. Moradian ,Alireza Esmkhani ,Farzad M. Jafarlou

IEEE Paper on “Character Recognition Using Neural Networks” by Rokus Arnold, Poth Miklos