

A Project Report
on

DIGITAL IMAGE PROCESSING TOOL USING JAVA

Submitted in partial fulfillment of the requirements for the Major Project II of

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Abhilekh Bahuguna	(R780209002)
Abhishek Kumar Singh	(R780209044)
Aditya Durgapal	(R780209047)
Ashish Sharma	(R780209009)

Under the guidance
Mr. Anil Kumar
Assistant Professor,
CIT, UPES
DEHRADUN



Department of Computer Science & Engineering
COLLEGE OF ENGINEERING STUDIES
UNIVERSITY OF PETROLEUM & ENERGY STUDIES
Dehradun- 248007

April 2013

CERTIFICATE

This is to certify that the Project entitled “**DIGITAL IMAGE PROCESSING TOOL USING JAVA**” submitted by

Abhilekh Bahuguna	(R780209002)
Abhishek Kumar Singh	(R780209044)
Aditya Durgapal	(R780209047)
Ashish Sharma	(R780209009)

for the partial fulfillment of the requirements of the course **Major Project II** of **Bachelor of Technology in Computer Science & Engineering** degree of **University of Petroleum & Energy Studies, Dehradun** embodies the confide work done by above students under my supervision.

Anil Kumar
30/04/13,
Signature of Mentor

Mr. Anil Kumar
Assistant Professor,
CIT, UPES
Dehradun

Approved by:

Dr. Ajay Shankar Singh
Project coordinator, Major Project-II
B.Tech CSE(VIII Sem) (2009-2013 batch)

Dr. Manish Prateek
Head of Department
CIT, UPES
Dehradun

External-I

A.H.S.

Ajay
MSI
30/04/13

External - II
MSI

DECLARATION

We, Abhilekh Bahuguna, Abhishek Kumar Singh, Aditya Durgapal and Ashish Sharma bearing the Roll No: R780209002, R780209044, R780209047 and R780209009 respectively hereby declare that this Project work entitled “**DIGITAL IMAGE PROCESSING TOOL USING JAVA**” was carried out by us under the guidance and supervision of Mr. Anil Kumar. This Project work is submitted to University of Petroleum & Energy Studies in partial fulfillment of the requirement for the award of Bachelor of Technology in Computer Science and Engineering during the Academic Semester January 2013 - April - 2013. We also declare that, we have not submitted this dissertation work to any other university for the award of either degree or diploma.

Place: Dehradun

Abhilekh Bahuguna

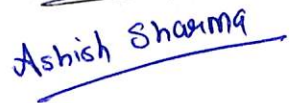
Abhishek Kumar Singh



Aditya Durgapal



Ashish Sharma



Date: 30/04/2013

ACKNOWLEDGEMENT

It is indeed a moment to thank all those many people who helped, supported and encouraged us during this project work.

Firstly we express our sincere gratitude to Mr. Anil Kumar, the guide of the project who carefully and patiently lent his valuable time and effort to give directions as well as to correct various documents with attention and care.

It is a great honor to do this project in this esteemed institution, and we would extend our thanks to the HOD, Dr. Manish Prateek and other faculty members who have shared their vast knowledge and experience during our stay.

We do also like to appreciate the consideration of the Project Coordinator Dr. Ajay Shankar Singh, our Faculties and colleagues, which enabled us to balance our work along with this project. It was their attitude that inspired us to do such an efficient and apposite work.

We are indebted to those people across the globe who shared their knowledge and perspectives in the form of online tutorials, forums and other resources which helped us to a great extent whenever we met with technical obstacles during this endeavour.

We wish to avail this opportunity to express a sense of gratitude and love to all our friends and our family for their unwavering support, strength, help and in short for everything they have done during the crucial times of the progress of our project.

Last but not the least we thank GOD ALMIGHTY for his blessings and guidance without which this dream project wouldn't have been reality.

Abhilekh Bahuguna
Abhishek Kumar Singh
Aditya Durgapal
Ashish Sharma

ABSTRACT

Image processing involves changing the nature of an image in order to either improve its pictorial information for human interpretation, render it more suitable for autonomous machine perception. Digital image processing involves using a computer to change the nature of a digital image.

The project aims to develop a tool for digital image processing. The tool performs various operations on a digital image.

For example the button labeled Image blurring is used to blurr the image. The button labeled compress will compress the size of the image without affecting its display quality. The color restoration button restores colors in a black and white image. Similarly, various operations like removing noise, morphing, deblurring, image rotation can be applied on the digital image using the proposed tool.

The use of JAVA at the front allows us to use its inbuilt functions for developing an image processing tool. The IDE used to develop the image processing tool is Eclipse.

TABLE OF CONTENTS

Item Description	Page No
Certificate	2
Declaration	3
Acknowledgement	4
Abstract	5
List of Figures	8
Chapter 1 Literature Review	10
Chapter 2 Introduction to Image Processing	14
2.1 Images and pictures	14
2.2 What is image processing?	14
2.3 Fundamental Steps Involved in Digital Image Processing	19
2.4 Images and digital images	21
2.5 Aspects of Image Processing	23
Chapter 3 Different techniques used in Image Processing	24
3.1 Haar Transform and Image Extraction	24
3.2 Image Histogram Analysis	30
3.3 Thresholding Technique	32
3.4 Filtering Technique using Digital Image Processing	32

Chapter 4	Image Processing using Java	36
4.1	RGB conversion into hue, saturation & intensity values	36
4.2	Enlarge image by pixel replication	37
4.3	Mapping grey levels into image	39
4.4	Algorithm for calculation of Image histogram	40
4.5	Histogram analysis	45
4.6	Convolution application with CLI	47
4.7	Image Rotation by Forward Mapping	49
4.8	Grey Level Thresholding	50
4.9	Connected Design Labeling	50
4.10	Gaussian filter	54
Conclusion		57
References		58

LIST OF FIGURES

Figure No.	Description	Page No.
2.1	Image Sharpening	15
2.2	Removing Noise from an image	16
2.3	Image Deblurring	17
2.4	Edge finding	17
2.5	Image Blurring	18
2.6	Steps in Image Processing	19
2.7	An Image as a function	21
2.8	Image plotted as a function of two variables	22
3.1	Flowchart of proposed text	24
3.2	Result of 2-D DWT decomposition	25
3.3	Traditional edge detection using mask operation	26
3.4	(a) Original grey image (b) DWT coefficients	27
3.5	(a) Original image (b) row operation of 2-D Haar DWT	27
3.6	Dilated image of three binary regions	28
3.7	Horizontal, diagonal and vertical edges dilation operators	29
3.8	Text extraction using logical And operator	29
3.9	(a) Candidate text regions (b) the extracted real text regions	30
4.1	Algorithm for grey level mapping	37

4.2	Algorithm for histogram equalization	45
4.3	Unequalized and equalized images and their histograms	46
4.4	Algorithm for colour image histogram	47
4.5	Convulation Application with GUI	48
4.6	Image Autorotation	49
4.7	Algorithm for filter method	51
4.8	Morphological filtering example I	53
4.9	Dealing with holes	53
4.10	Image Thresholding using Gaussian Filter	56

CHAPTER-1

Literature Review

1.1 Findings of Literature Survey:

Various methods have been proposed in the past for detection and localization of text in images and videos. These approaches take into consideration different properties related to text in an image such as color, intensity, connected-components, edges etc. These properties are used to distinguish text regions from their background and/or other regions within the image. The algorithm proposed by Wang and Kangas is based on color clustering. The input image is first pre-processed to remove any noise if present. Then the image is grouped into different color layers and a gray component. This approach utilizes the fact that usually the color data in text characters is different from the color data in the background. The potential text regions are localized using connected component based heuristics from these layers. Also an aligning and merging analysis (AMA) method is used in which each row and column value is analyzed. The experiments conducted show that the algorithm is robust in locating mostly Chinese and English characters in images; some false alarms occurred due to uneven lighting or reflection conditions in the test images. The text detection algorithm is also based on color continuity. In addition it also uses multi-resolution wavelet transforms and combines low as well as high level image features for text region extraction. The text finder algorithm proposed in is based on the frequency, orientation and spacing of text within an image. Texture based segmentation is used to distinguish text from its background. Further a bottom-up 'chip generation' process is carried out which uses the spatial cohesion property of text characters. The chips are collections of pixels in the image consisting of potential text strokes and edges. The results show that the algorithm is robust in most cases, except for very small text characters that are not properly detected. Also in the case of low contrast in the image, misclassifications occur in the texture segmentation.

A focus of attention based system for text region localization has been proposed by Liu and Samarabandu. The intensity profiles and spatial variance is used to detect text regions in images. A Gaussian pyramid is created with the original image at different resolutions or scales.

The text regions are detected in the highest resolution image and then in each successive lower resolution image in the pyramid.

The approach utilizes a support vector machine (SVM) classifier to segment text from non-text in an image or video frame. Initially text is detected in multiscale images using edge based techniques, morphological operations and projection profiles of the image. These detected text regions are then verified using wavelet features and SVM. The algorithm is robust with respect to variance in color and size of font as well as language. The goal of the research is to discover how the algorithms perform under variations of lighting, orientation, and scale transformations of the text. The algorithms are from Liu and Samarabandu and Gllavata, Ewerth and Freisleben. The comparison is based on the accuracy of the results obtained, and precision and recall rates. The technique is an edge-based text extraction approach, and the technique used is a connected-component based approach. In order to test robustness and performance the approaches used, each algorithm was first implemented in the original proposed format. The algorithms were tested on the image data set provided by as well as another data set which consists of a combination of indoor and outdoor images taken from a digital camera. The results obtained were recorded based on criteria such as invariance with respect to lighting conditions, color, rotation, and distance from the camera (scale) as well as horizontal and/or vertical alignment of text in an image. The experiments have also been conducted for images containing 8 different font styles and text characters belonging to language types other than English. Also, the precision and recall rates (Equations (1) and (2)), have been computed based on the number of correctly detected words in an image in order to further evaluate the efficiency and robustness of each algorithm. The Precision rate is defined as the ratio of correctly detected words to the sum of correctly detected words plus false positives. False positives are those regions in the image which are actually not characters of a text, but have been detected by the algorithm as text regions.

$$\text{Precisionrate} = \frac{\text{Correctlydetectedwords}}{\text{Correctlydetectedwords} + \text{FalsePositives}} \times 100 \%$$

The Recall rate is defined as the ratio of correctly detected words to the sum of correctly detected words plus false negatives. False Negatives are those regions in the image which are actually text characters, but have not been detected by the algorithm

$$\text{Recallrate} = \frac{\text{Correctlydetectedwords}}{\text{Correctlydetectedwords} + \text{FalseNegatives}} \times 100 \%$$

Various papers presents the status of research on text extraction

1.1.1 IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 5, NO. 8, AUGUST

1996: This correspondence discusses an extension of the well known phase correlation technique to cover translation, rotation, a scaling. Fourier scaling properties and Fourier rotational property are used to find scale and rotational movement. The phase correlation technique determines the translational movement. This method show excellent robustness against random noise.

1.1.2 IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 19, NO.4, APRIL 2010

Learning With Graph for Image Analysis: Bin Cheng, Jianchao Yang, Student Member, IEEE, Shuicheng Yan, Senior Member, IEEE, Yun Fu, Member, IEEE, and Thomas S. Huang, Life Fellow, IEEE: The graph construction procedure essentially determines the potential soft hose graph-oriented learning algorithm for image analysis. In this paper, we propose a process to build the so-called directed -graph, in which the vertices involve all the samples and the ingoing edge weights to each vertex describe it-norm driven reconstruction from the remaining samples and the noise. Then, a series of new algorithms for various mach in learning tasks, e.g., data clustering, sub space learning, and semi supervised learning, are derived upon the -graphs. Compared with the conventional -nearest-neighbor graph and -ball graph the -graph possesses the advantages.

- 1) Greater robustness data noise,
- 2) Automatic sparsity, and
- 3) Adaptive neighborhood

1.1.3 IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 19, NO .4, APRIL 2010:

Deform PF-MT: Particle Filter With Mode Tracker for Tracking Nonaffine Contour Deformations NamrataVaswani, YogeshRathi, Anthony Yezzi, and Allen Tannenbaum: In this paper proposed algorithms are for tracking the boundary contour of a deforming object from an image sequence, when the nonaffine (local) deformation over consecutive frames is large and

there is overlapping clutter, occlusions, low contrast, or outlier imagery. When the object is arbitrarily deforming, each, or at least most, contour points can move independently. Contour deformation then forms an infinite (in practice, very large), dimensional space. Direct application of particle filters (PF) for large dimensional problems is impractically expensive. However in most real problems, at any given time, most of the contour deformation occurs in a small number of dimensions (“effective basis space”) while the residual deformation in the rest of the state space (“residual space”) is small. This property enables us to apply the particle filtering with mode tracking (PF-MT) idea that was proposed for such large dimensional problems in recent work. Since most contour deformation is low spatial frequency we proposed to use the space of deformation at a sub sampled set of locations as the effective basis space. The resulting algorithm is called deform PF-MT. It requires significant modifications compared to the original PF-MT because the space of contours is a non-Euclidean infinite dimensional space.

CHAPTER-2

Introduction to Image Processing

2.1 Images and pictures

Human Beings are predominantly visual creatures. We rely heavily on our vision to make sense of the world around us. We not only look at things to identify and classify them, but we can scan for differences, and obtain an overall rough feeling for a scene with a quick glance.

Humans have evolved very precise visual skills, we can identify a face in an instant, we can differentiate colors, and we can process a large amount of visual information very quickly.

However, the world is in constant motion. Stare at something for long enough and it will change in some way. Even a large solid structure, like a building or a mountain, will change its appearance depending on the time of day (day or night); amount of sunlight (clear or cloudy), or various shadows falling upon it.

For our purposes, an image is a single picture which represents something. It may be a picture of a person, of people or animals, or of an outdoor scene, or a microphotograph of an electronic component, or the result of medical imaging. Even if the picture is not immediately recognizable, it will not be just a random blur.

2.2 What is image processing?

Image processing involves changing the nature of an image in order to either

1. improve its pictorial information for human interpretation,
2. render it more suitable for autonomous machine perception.

We shall be concerned with digital image processing, which involves using a computer to change the nature of a digital image (see below). It is necessary to realize that these two aspects represent two separate but equally important aspects of image processing. A

procedure which satisfies condition (1)- a procedure which makes an image “look better” – may be the very worst procedure for satisfying condition (2)- Humans like their images to be sharp, clear and detailed; machines prefer their images to be simple and uncluttered.

Examples of (1) may include:

- Enhancing the edges of an image to make it appear sharper; an example is shown in figure 2.1. Note how the second image appears “cleaner”; it is a more pleasant image. Sharpening edges is a vital component of printing: in order for an image to appear “at its best” on the printed page; some sharpening is usually performed.



(a) The original image



(b) Result after sharpening

Figure 2.1: Image sharpening

- Removing “noise” from an image; noise being random errors in the image. An example is given in figure 2.2. Noise is a very common problem in data transmission: all sorts of electronic components may affect data passing through them, and the results may be undesirable.

- Removing motion blur from an image. An example is given in figure 2.3. Note that in the deblurred image (b) it is easier to read the number plate, and to see the spikes on the fence behind the car, as well as other details not at all clear in the original image (a). Motion blur may occur when the shutter speed of the camera is too long for the speed of the object. In photographs of fast moving objects: athletes, vehicles for example, the problem of blur may be considerable.

Examples of (2) may include:

- Obtaining the edges of an image. This may be necessary for the measurement of objects in an image; an example is shown in figures 2.4. Once we have the edges we can measure their spread, and the area contained within them. We can also use edge detection algorithms as a first step in edge enhancement, as we saw above.

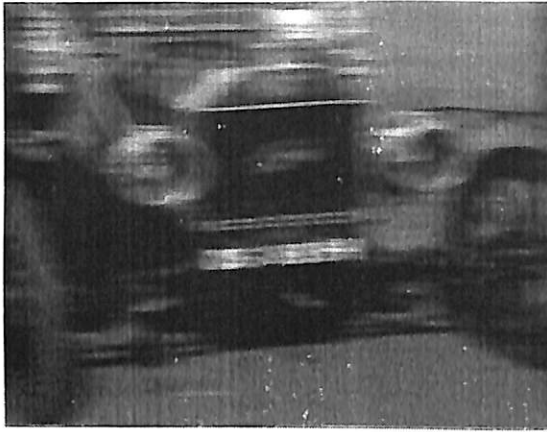


(a) The original image

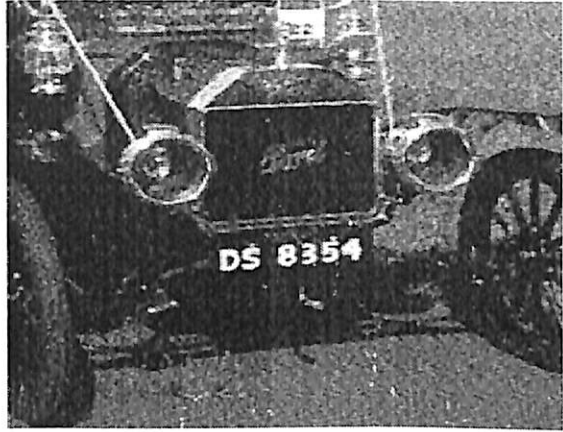


(b) After removing noise

Figure 2.2: Removing noise from an image



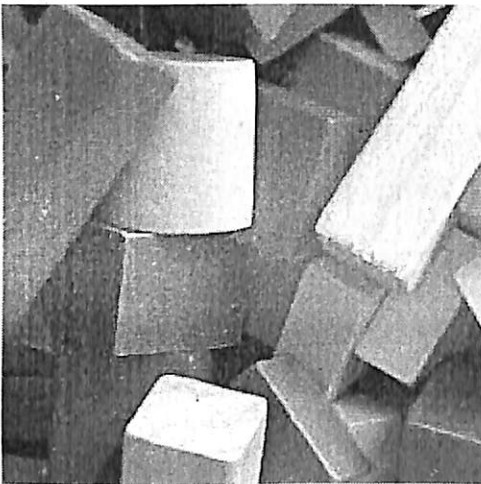
(a) The original image



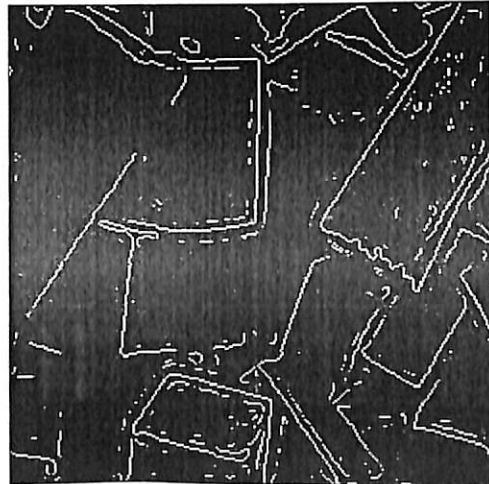
(b) After removing the blur

Figure 2.3: Image deblurring

From the edge result, we see that it may be necessary to enhance the original image slightly, to make the edges clearer.



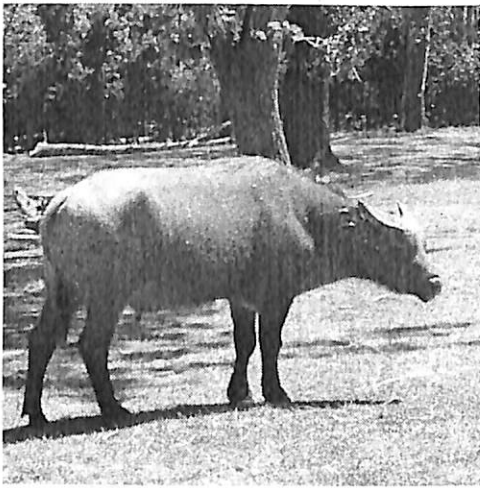
(a) The original image



(b) Its edge image

Figure 2.4: Finding edges in an image

Removing detail from an image. For measurement or counting purposes, we may not be interested in all the detail in an image. For example, a machine inspected items on an assembly line, the only matters of interest may be shape, size or color. For such cases, we might want to simplify the image. Figure 2.5 shows an example: in image (a) is a picture of an African buffalo, and image (b) shows a blurred version in which extraneous detail (like the logs of wood in the background) have been removed. Notice that in image (b) all the fine detail is gone; what remains is the coarse structure of the image. We could for example, measure the size and shape of the animal without being “distracted” by unnecessary detail.



(a) The original image



(b) Blurring to remove detail

Figure 2.5: Blurring an image

2.3 Fundamental Steps Involved in Digital Image Processing

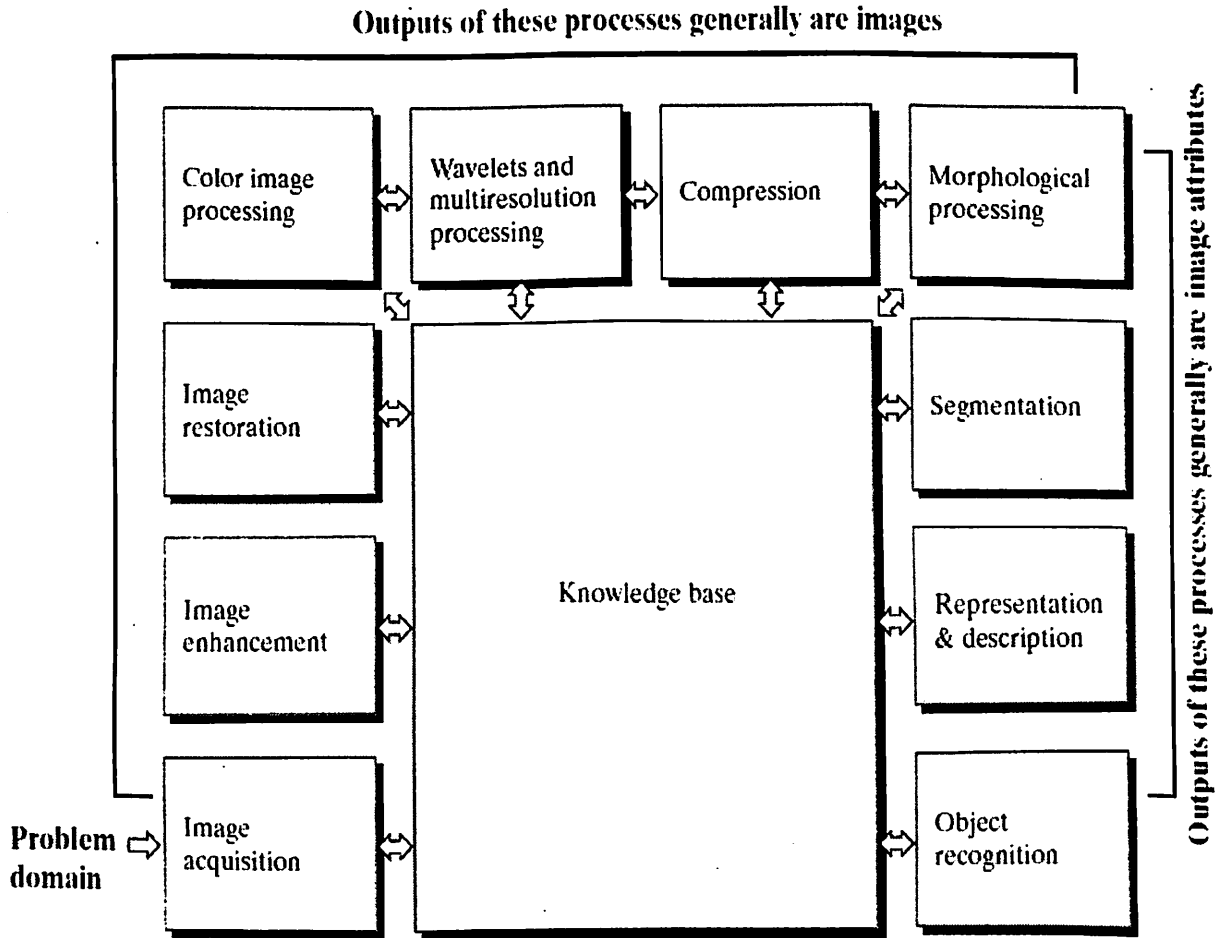


Fig 2.6 Steps in Image processing

(i) **Image Acquisition** : This is the first step or process of the fundamental steps of digital image processing. Image acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves preprocessing, such as scaling etc.

(ii) **Image Enhancement** : Image enhancement is the process of manipulating an image so that result is more suitable than original for specific application. This technique is meant to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc.

(iii) **Image Restoration** : Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is

objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

(iv) **Color Image Processing** : Color image processing is an area that has been gaining its importance because of the significant increase in the use of digital images over the Internet. This may include color modeling and processing in a digital domain etc.

(v) **Wavelets and Multiresolution Processing** : Wavelets are the foundation for representing images in various degrees of resolution. Images subdivision successively into smaller regions for data compression and for pyramidal representation.

(vi) **Compression** : Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very much necessary to compress data.

(iv) **Color Image Processing** : Color image processing is an area that has been gaining its importance because of the significant increase in the use of digital images over the Internet. This may include color modeling and processing in a digital domain etc.

(v) **Wavelets and Multiresolution Processing** : Wavelets are the foundation for representing images in various degrees of resolution. Images subdivision successively into smaller regions for data compression and for pyramidal representation.

(vi) **Compression** : Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very much necessary to compress data.

vii) **Morphological Processing** : Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape.

(viii) **Segmentation** : Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.

(ix) **Representation and Description** : Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. Description deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another

(x) **Object recognition** : Recognition is the process that assigns a label, such as, “vehicle” to an object based on its descriptors

(xi) **Knowledge Base** : Knowledge may be as simple as detailing regions of an image where the information of interest is known to be located, thus limiting the search that has to be conducted in seeking that information.

The knowledge base also can be quite complex, such as an interrelated list of all major possible defects in a materials inspection problem or an image database containing high-resolution satellite images of a region in connection with change-detection applications.

2.4 Images and digital images

Suppose we take an image, a photo, say. For the moment, let’s make things easy and suppose the photo is monochromatic (that is, shades of grey only), so no color. We may consider this image as being a two dimensional function, where the function values give the brightness of the image at any given point, as shown in figure 2.6. We may assume that in such an image brightness values can be any real numbers in the range 0.0 (black) to 1.0 (white). The ranges of x and y will clearly depend on the image, but they can take all real values between their minima and maxima.

Such a function can of course be plotted, as shown in figure 2.7. However, such a plot is of limited use to us in terms of image analysis. The concept of an image as a function, however, will be vital for the development and implementation of image processing techniques.

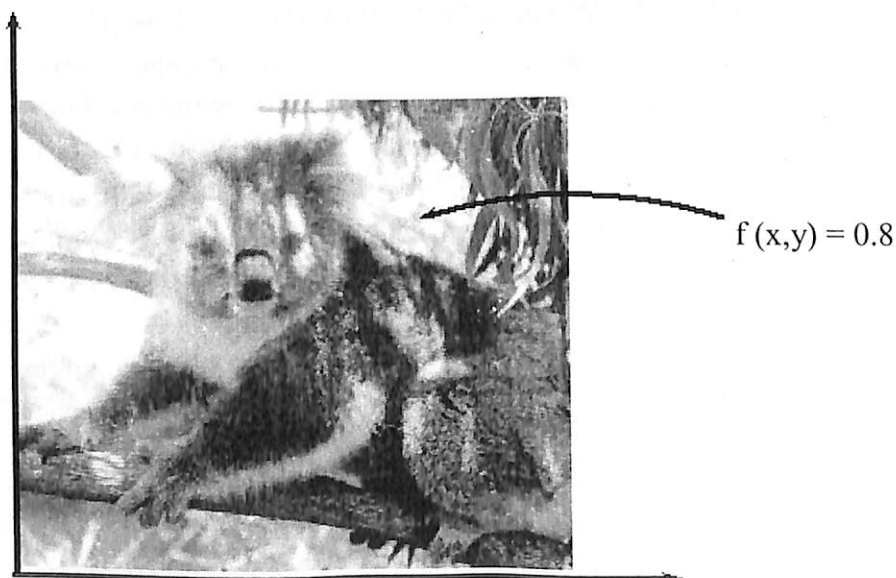


Figure 2.7: An image as a function

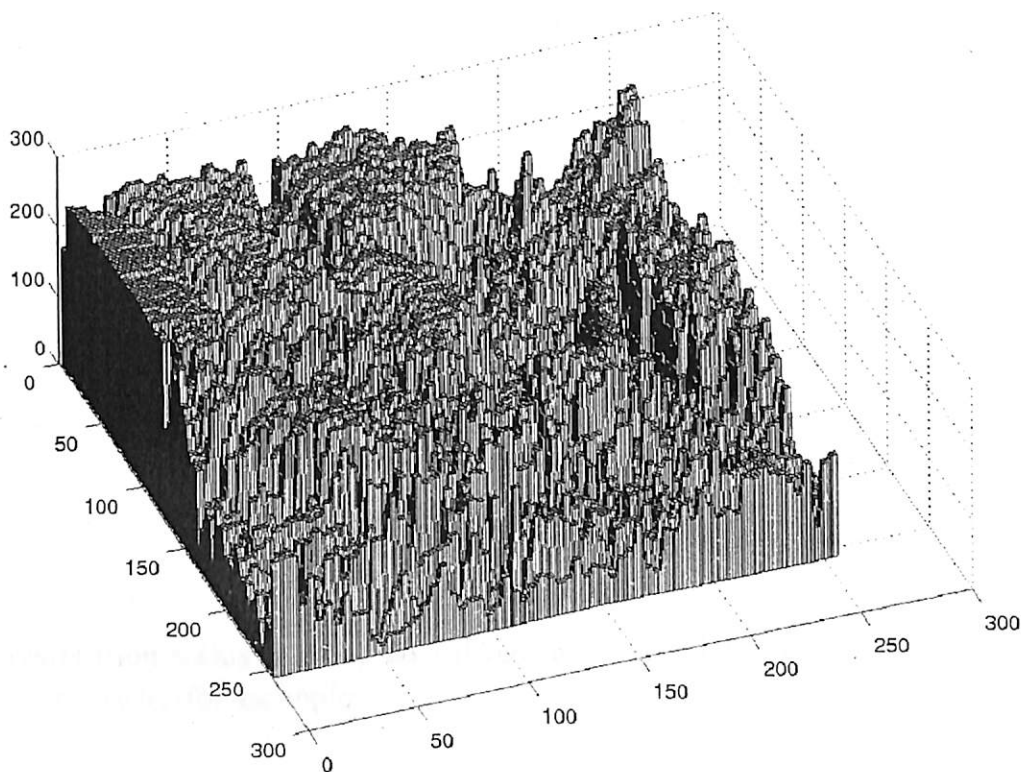


Figure 2.8: The image plotted as a function of two variables

A digital image differs from a photo in that the x, y and $f(x, y)$ values are all discrete. Usually they take on only integer values, so the image shown will have x and y ranging from 1 to 256 each, and the brightness values also ranging from 0 (black) to 255 (white). A digital image, as we have seen above, can be considered as a large array of sampled points from the continuous image, each of which has a particular quantized brightness; these points are the pixels which constitute the digital image. The pixels surrounding a given pixel constitute its neighborhood. A neighborhood can be characterized by its shape in the same way as a matrix: we can speak, for example, of a 3×3 neighborhood, or of a 5×7 neighborhood. Except in very special circumstances, neighborhoods have odd numbers of rows and columns; this ensures that the current pixel is in the center of the neighborhood. If a neighborhood has an even number of rows or columns (or both), it may be necessary to specify which pixel in the neighborhood is the "current pixel".

2.5 Aspects of image processing

It is convenient to subdivide different image processing algorithms into broad subclasses. There are different algorithms for different tasks and problems, and often we would like to distinguish the nature of the task at hand.

Image enhancement : This refers to processing an image so that the result is more suitable for a particular application. Example include:

- sharpening or de-blurring an out of focus image,
- highlighting edges,
- improving image contrast, or brightening an image,
- removing noise.

Image restoration : This may be considered as reversing the damage done to an image by a known cause, for example:

- removing of blur caused by linear motion,
- removal of optical distortions,
- removing periodic interference.

Image segmentation : This involves subdividing an image into constituent parts, or isolating certain aspects of an image:

- finding lines, circles, or particular shapes in an image,
- in an aerial photograph, identifying cars, trees, buildings, or roads.

These classes are not disjoint; a given algorithm may be used for both image enhancement or for image restoration. However, we should be able to decide what it is that we are trying to do with our image: simply make it look better (enhancement), or removing damage (restoration).

CHAPTER-3

Different techniques used in Image Processing

3.1 Haar Transform and Image Extraction

3.1.1 Introduction to Haar Transform

In this chapter, we present a method to extract texts in images using Haar discrete wavelet transform (Haar DWT). The edges detection is accomplished by using 2-D Haar DWT and some of the non-text edges are removed using thresholding. After-ward, we use different morphological dilation operators to connect the isolated candidate text edges in each detail component sub-band of the binary image. Although the color component may differ in a text region, the information about colors does not help extracting texts from images. If the input image is a gray-level image, the image is processed directly starting at discrete wavelet transform.

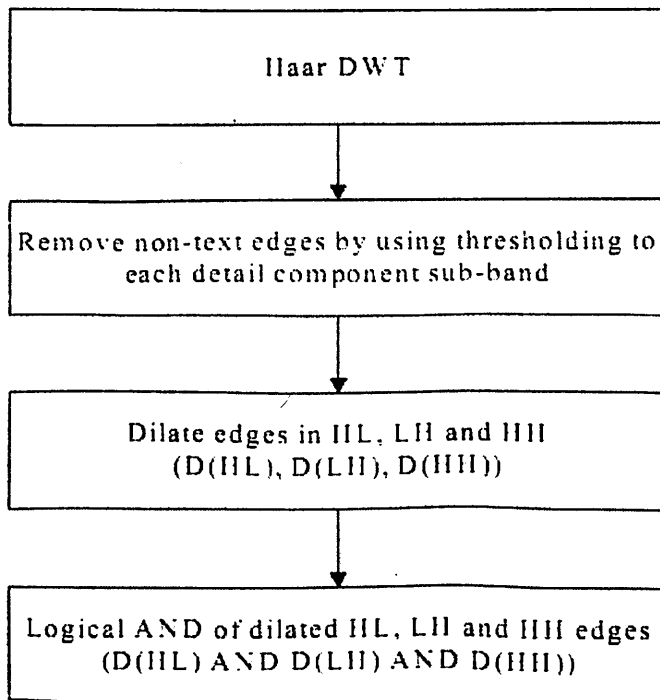


Fig 3.1: Flow chart of proposed text

If the input image is colored, its RGB components are combined to give an intensity image Y as follows

$$Y = 0.299R + 0.587 G + 0.114 B \dots\dots(1)$$

Image Y is then processed with discrete wavelet transform and the whole extraction algorithm afterward. If the input image itself is stored in the DWT compressed form, DWT operation can be omitted in the proposed algorithm. The flow chart of the proposed algorithm is shown in Figure 3.1.

3.1.2 Haar discrete wavelet transform

The discrete wavelet transform is a very useful tool for signal analysis and image processing, especially in multi-resolution representation. It can decompose signal into different components in the frequency domain. One-dimensional discrete wavelet transform (1-D DWT) decomposes an input sequence into two components (the average component and the detail component) by calculations with a low-pass filter and a high-pass filter. Two-dimensional discrete wavelet transform (2-D DWT) decomposes an input image into four sub-bands, one average component (LL) and three detail components (LH, HL, HH) as shown in Figure 3.2. In image processing, the multi-resolution of 2-D DWT has been employed to detect edges of an original image. The traditional edge detection filters can provide the similar result as well. However, 2-D DWT can detect three kinds of edges at a time while traditional edge detection filters cannot. As shown in Figure 3.3, the traditional edge detection filters detect three kinds of edges by using four kinds of mask operators. Therefore, processing times of the traditional edge detection filters is slower than 2-D DWT.

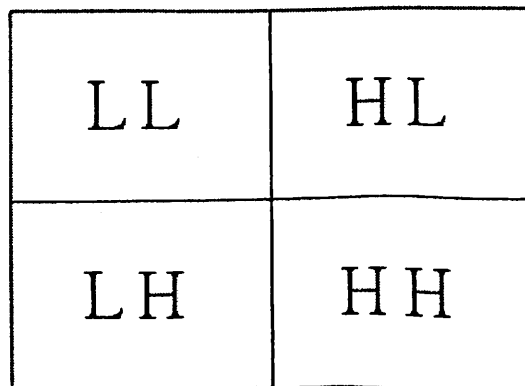


Figure 3.2 The result of 2-D DWT decomposition

Figure 3.4 (a) shows a gray level image. The 9-7 taps DWT filters decompose this gray image into four sub-bands as shown in Figure 3.4 (b). As we can see, three kinds of edges present in the detail component sub-bands but look unobvious (very small coefficients). If we replace the 9-7 taps DWT filters with Haar DWT, the detected edges become more obvious and the processing time decreases. The operation for Haar DWT is simpler than that of any other wavelets. It has been applied to image processing especially in multi-resolution representation. Harr DWT has the following important features.

- Haar wavelets are real, orthogonal, and symmetric.
- Its boundary conditions are the simplest among all wavelet-based methods.
- The minimum support property allows arbitrary spatial grid intervals.
- It can be used to analyze texture and detect edges of characters.
- The high-pass filter and the low-pass filter coefficient is simple (either 1 or -1).

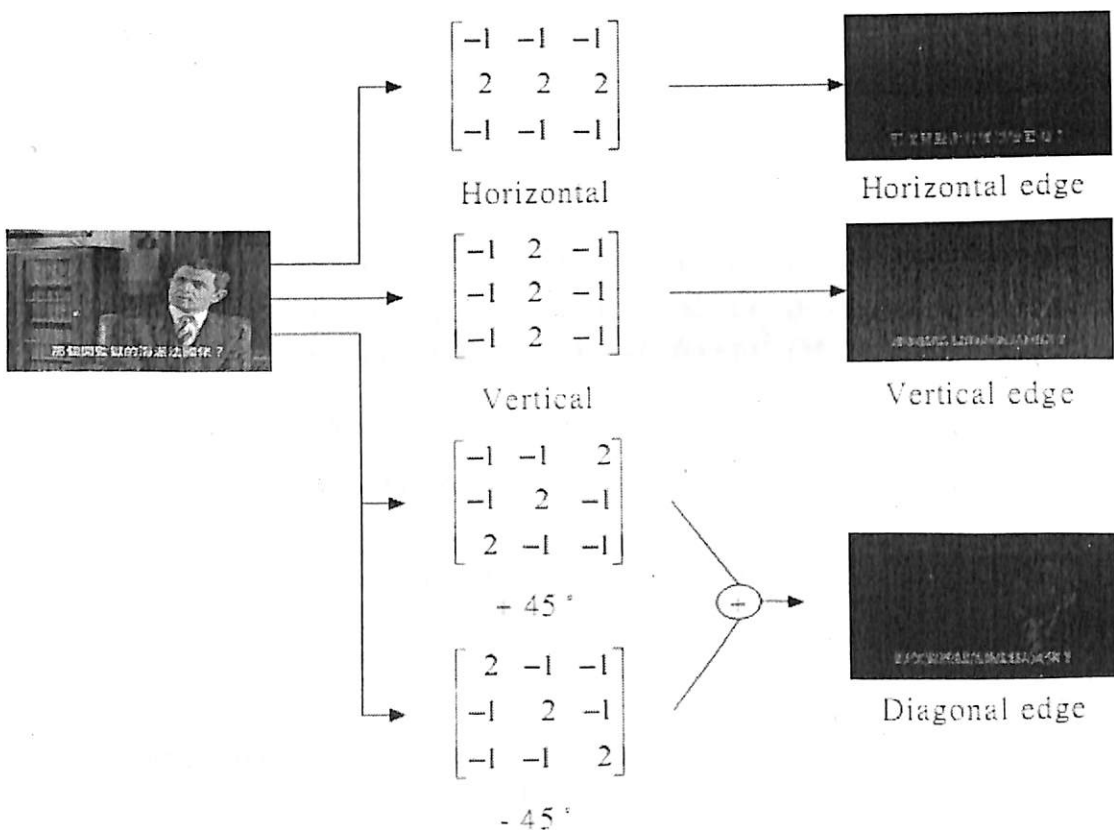


Figure 3.3 Traditional edge detection using mask operation



Figure 3.4 (a) Original gray image (b) DWT coefficients

Figure 3.5 (a) shows the example of a 4×4 gray-level image. The wavelet coefficients can be obtained in gray-level image using addition and subtraction. 2-D DWT is achieved by two ordered 1-D DWT operations (row and column). First of all, we perform the row operation to obtain the result shown in Figure 3.5 (b). Then it is transformed by the column operation and the final resulted 2-D Haar DWT is shown in Figure 3.5 (c). 2-D Haar DWT decomposes a gray-level image into one average component sub-band and three detail component sub-bands.

$$\begin{array}{ccc}
 \begin{bmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ M & N & O & P \end{bmatrix} & & \begin{bmatrix} (A+B) & (C+D) & (A-B) & (C-D) \\ (E+F) & (G+H) & (E-F) & (G-H) \\ (I+J) & (K+L) & (I-J) & (K-L) \\ (M+N) & (O+P) & (M-N) & (O-P) \end{bmatrix} \\
 \text{(a)} & & \text{(b)} \\
 & & \begin{bmatrix} (A+B)+(E+F) & (C+D)+(G+H) & (A-B)+(E-F) & (C-D)+(G-H) \\ (I+J)+(M+N) & (K+L)+(O+P) & (I-J)+(M-N) & (K-L)+(O-P) \\ (A+B)-(E+F) & (C+D)-(G+H) & (A-B)-(E-F) & (C-D)-(G-H) \\ (I+J)-(M+N) & (K+L)-(O+P) & (I-J)-(M-N) & (K-L)-(O-P) \end{bmatrix} \\
 & & \text{(c)}
 \end{array}$$

Figure 3.5 (a) The original image (b) the row operation of 2-D Haar DWT (c) the column

3.1.3 Text region extraction

In this subsection, we use morphological operators and the logical AND operator to further removes the non-text regions. In text regions, vertical edges, horizontal edges and diagonal edges are mingled together while they are distributed separately in non-text regions. Since text regions are composed of vertical edges, horizontal edges and diagonal edges, we can determine the text regions to be the regions where those three kinds of edges are intermixed. Text edges are generally short and connected with each other in different orientation. In Figure 3.7, we use different morphological dilation operators to connect isolated candidate text edges in each detail component sub-band of the binary image. In this research, 3×5 for horizontal operators, 3×3 for diagonal operators and 7×3 for vertical operators as in shown Figure 3.8 are applied. The dilation operators for the three detail sub-bands are designed differently so as to fit the text characteristics. The logical AND is then carried on three kinds (vertical, horizontal and diagonal) of edges after morphological dilation. This process is indicated in Figure 3.9 Since three kinds of edge regions are intermixed in the text regions, overlapping appears a lot after the morphological dilation due to the expansion of each single edge. On the contrary, only one kind of edge region or two kinds of edge regions exist separately in the non-text regions and hence there is no overlapping even after the dilation. Therefore, the AND operator helps us to obtain the candidate text regions as shown in Figure 3.10 (a). Some-times the text candidate regions may contain some non-text component regions which are too large or too small. By limiting the block size, we obtain the final text regions. Each text region has a moderate size $w \times h$ (pixels) in a candidate text region image.



Figure 3.6 The dilated image of three binary regions

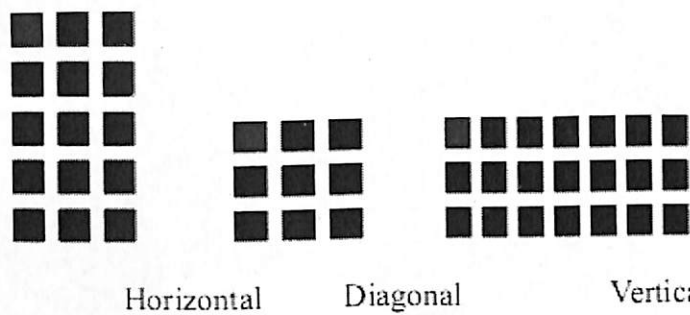


Figure 3.7 Horizontal, Diagonal and Vertical edges dilation operators

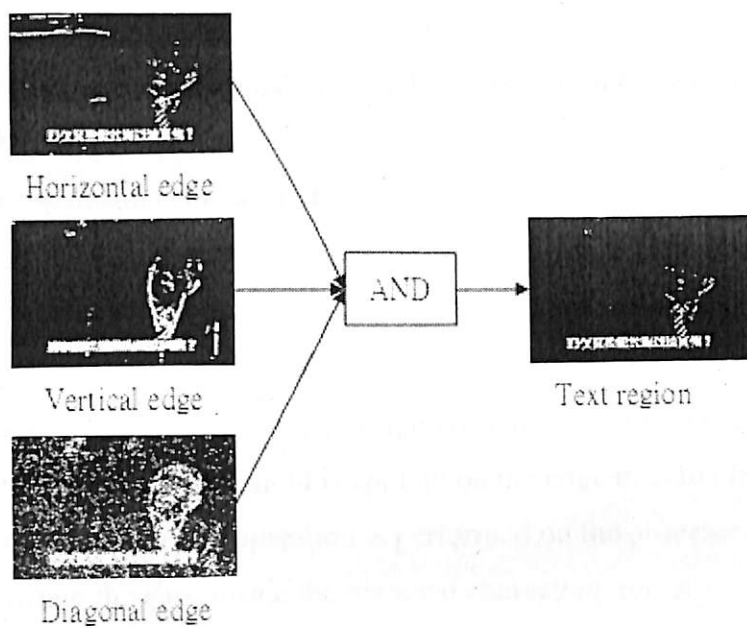
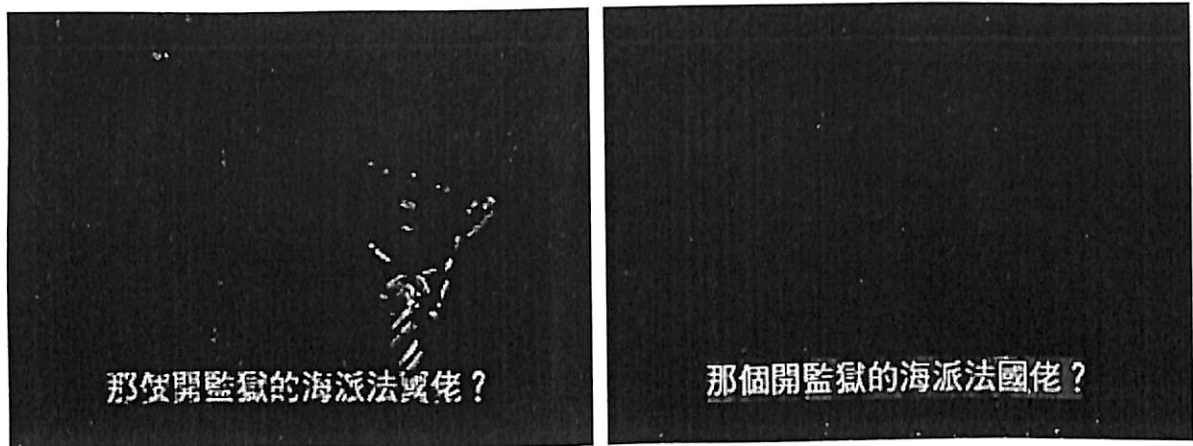


Figure 3.8 Text extraction by using the logical AND operator



(a)

(b)

Figure 3.9 (a) The candidate text region (b) the extracted real text region

Removing the candidate text regions smaller than this limit, the final text region is shown in the Figure 3.10 (b).

The dense edges are the distinct characteristics of the text blocks which are used to detect the possible text regions. The candidate text regions are found by finding the edges in the mentioned sub-bands and fusing the edges contained in each sub-band. The Sobel edge detector is efficient to extract the strong edges. The Sobel edge detector is applied on each sub-band to get the candidate text edges. In the next step, using a weighted 'OR' operator, these candidate text edges are used to form the edge map. A threshold is applied on the edge map to obtain the binary edge map. Then, a morphological dilation operation is performed on the processed edge map. This operation results in filling the gaps inside the obtained characters' regions.

3.2 Image Histogram Analysis

3.2.1 Histogram

In statistics, a histogram is a graphical representation of the distribution of data. It is an estimate of the probability distribution of a continuous variable and was first introduced by Karl Pearson. A histogram is a representation of tabulated frequencies, shown as adjacent rectangles, erected over discrete intervals (bins), with an area equal to the frequency of the observations in the interval. The height of a rectangle is also equal to the frequency density of the interval, i.e., the frequency divided by the width of the interval. The total area of the histogram is equal to the number of data.

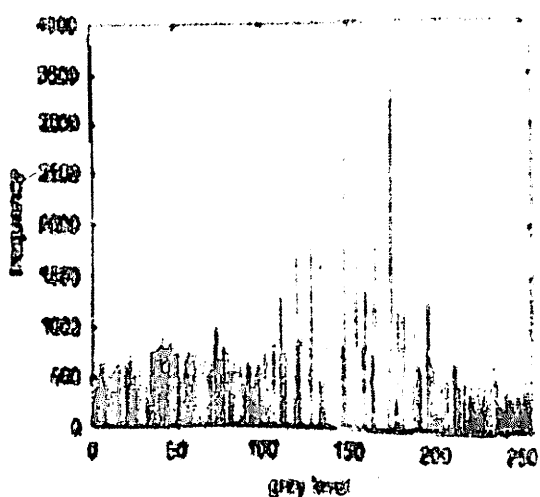
In an image processing context, the histogram of an image normally refers to a histogram of the pixel intensity values. This histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. For an 8-bit grayscale image there are 256 different possible intensities, and so the histogram will graphically display 256 numbers showing the distribution of pixels amongst those grayscale values. Histograms can also be taken of color images --- either individual histograms of red, green and blue channels can be taken, or a 3-D histogram can be produced, with the three axes representing the red, blue and green channels, and brightness at each point representing the pixel count. The exact output from the operation depends upon the implementation --- it may simply be a picture of the required histogram in a suitable image format, or it may be a data file of some sort representing the histogram statistics.

3.2.2 Algorithm for image histogram

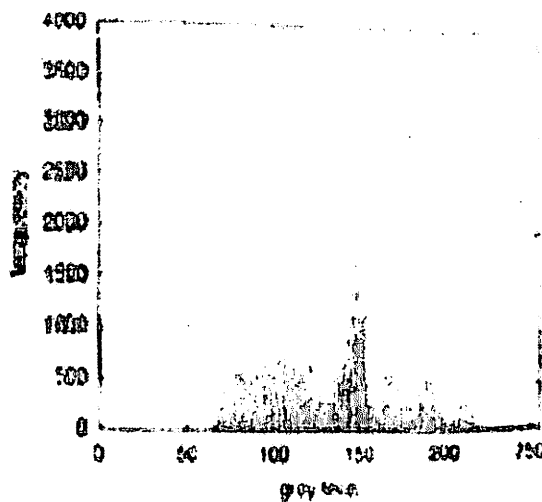
```

Create an array histogram with  $2^b$  elements
for all grey levels, i , do
  histogram[i ] = 0;
end for
for all pixel coordinates, x and y, do
  Increment histogram[f (x,y)] by 1
end for
  
```

Histogram of an equalized Image



Histogram of equalized Image



3.3 Thresholding Technique

3.3.1 Brief Description

In many vision applications, it is useful to be able to separate out the regions of the image corresponding to objects in which we are interested, from the regions of the image that correspond to background. Thresholding often provides an easy and convenient way to perform this segmentation on the basis of the different intensities or colors in the foreground and background regions of an image.

In addition, it is often useful to be able to see what areas of an image consist of pixels whose values lie within a specified range, or *band* of intensities (or colors). Thresholding can be used for this as well.

3.3.2 How It Works

The input to a thresholding operation is typically a grayscale or color image. In the simplest implementation, the output is a binary image representing the segmentation. Black pixels correspond to background and white pixels correspond to foreground (or *vice versa*). In simple implementations, the segmentation is determined by a single parameter known as the *intensity threshold*. In a single pass, each pixel in the image is compared with this threshold. If the pixel's intensity is higher than the threshold, the pixel is set to, say, white in the output. If it is less than the threshold, it is set to black.

In more sophisticated implementations, multiple thresholds can be specified, so that a *band* of intensity values can be set to white while everything else is set to black. For color or multi-spectral images, it may be possible to set different thresholds for each color channel, and so select just those pixels within a specified cuboid in RGB space. Another common variant is to set to black all those pixels corresponding to background, but leave foreground pixels at their original color/intensity (as opposed to forcing them to white), so that that information is not lost.

3.4 Filtering Techniques used in Digital Image Processing

There are two type of filtering techniques used in digital image processing

- 1) Spatial filtering technique
- 2) Frequency filtering technique

3.4.1 Spatial filtering technique

Spatial filtering technique is one of the principal tools used in this field of broad spectrum of applications. The name *filter* is obtained from frequency domain processing, where filtering refers to accepting or rejecting certain frequency components.

3.4.1.1 Brief Description

A spatial filter consists of (1) a neighborhood and (2) a predefined operation that is performed on the image pixels encompassed by the neighborhood. Filtering creates a new pixel with coordinates equal to the coordinates of the center of the neighborhood and whose value is the result of the filtering operations.

If the operation performed on the image pixel is *linear* it is called linear spatial filter. Otherwise the filter is *nonlinear*.

3.4.1.2 The Mechanism of Spatial Filtering

At any point (x,y) in the image, the response, $g(x,y)$, of the filter is the sum of products of the filter coefficients and the image pixels encompassed by the filter:

$$g(x,y) = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \dots + w(0,0)f(x,y) + \dots + w(1,1)f(x+1,y+1)$$

The central coefficient of the filter, $w(0,0)$, aligns with the pixel at location (x,y) . For a mask of size $m \times n$, we assume that $m = 2a+1$ and $n = 2b+1$ where a and b are positive integers.

3.4.1.3 Gaussian Filters

In electronics and signal processing, a Gaussian filter is a filter whose impulse response is a Gaussian function. Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time. This behavior is closely connected to the fact that the Gaussian filter has the minimum possible group delay. Mathematically, a Gaussian filter modifies the input signal by convolution with a Gaussian function; this transformation is also known as the Weierstrass transform.

The one-dimensional Gaussian filter has an impulse response given by

$$g(x) = \sqrt{\frac{a}{\pi}} \cdot e^{-a \cdot x^2}$$

and the frequency response is given by Fourier transform

$$\hat{g}(f) = e^{-\frac{\pi^2 f^2}{a}}$$

with f the ordinary frequency. These equations can also be expressed with the standard deviation as parameter

$$g(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{x^2}{2\sigma^2}}$$

3.4.1.3.1 Algorithm for Gaussian Blur

```

Tmp1 = Input[j][i];           // Fetch the next input pixel
Tmp2 = SR0 + Tmp1;           // Form the intermediate value
SR0 = Tmp1;                   // Update 1st row state buffer
Tmp1 = SR1 + Tmp2;           // Form the row machine output
SR1 = Tmp2;                   // Update 2nd row state buffer
// Column machine
Tmp2 = SC0[i] + Tmp1;        // Form the intermediate value
SC0[i] = Tmp1;               // Update 1st column state buffer
Output[j-1][i-1] = (8 + SC1[i] + Tmp2)/16; // Form the output
SC1[i] = Tmp2;               // Update 2nd column state buffer

```

3.4.2 Frequency Filters

3.4.2.1 Brief Description

Frequency filters process an image in the frequency domain. The image is Fourier transformed, multiplied with the filter function and then re-transformed into the spatial domain. Attenuating high frequencies results in a smoother image in the spatial domain, attenuating low frequencies enhances the edges.

All frequency filters can also be implemented in the spatial domain and, if there exists a simple kernel for the desired filter effect, it is computationally less expensive to perform the filtering in the spatial domain. Frequency filtering is more appropriate if no straightforward kernel can be found in the spatial domain, and may also be more efficient.

3.4.2.2 How It Works

Frequency filtering is based on the Fourier Transform. (For the following discussion we assume some knowledge about the Fourier Transform, therefore it is advantageous if you have already read the corresponding worksheet.) The operator usually takes an image and a filter function in the Fourier domain. This image is then multiplied with the filter function in a pixel-by-pixel fashion:

$$G(k, l) = F(k, l)H(k, l)$$

where, $F(k,l)$ is the input image in the Fourier domain, $H(k,l)$ the filter function and $G(k,l)$ is the filtered image. To obtain the resulting image in the spatial domain, $G(k,l)$ has to be re-transformed using the inverse Fourier Transform.

Since the multiplication in the Fourier space is identical to convolution in the spatial domain, all frequency filters can in theory be implemented as a spatial filter. However, in practice, the Fourier domain filter function can only be approximated by the filtering kernel in spatial domain.

Chapter 4

Image processing using Java

4.1 A program to convert RGB colors into values of hue, saturation and intensity.

A conversion method from Java's Color class is used :

```
Import java.awt.Color;
Import java.text.DecimalFormat;
public class HSICalc
{
    public static void main(String[] argv) {
        if (argv.length > 2) {
            int[] rgb = new int[3];
            for (int i = 0; i < 3; ++i)
                rgb[i] = Integer.parseInt(argv[i]);
            float[] values = Color.RGBtoHSB(rgb[0], rgb[1], rgb[2], null);
            String[] labels = { "H=", "S=", "I=" };
            DecimalFormat floatValue = new DecimalFormat("0.000");
            for (int i = 0; i < 3; ++i)
                System.out.println(labels[i] + floatValue.format(values[i]));
        }
        else
        {
            System.err.println("usage: java HSICalc <r> <g> <b>");
            System.exit(1);
        }
    }
}
```

4.2 Java code to enlarge an image by pixel replication:

```
public static BufferedImage enlarge (BufferedImage image, int n )
{
    int w = n*image.getWidth();
    int h = n*image.getHeight();
    BufferedImage enlargedImage = new BufferedImage(w, h, image.getType());
    for (int y = 0; y < h; ++y)
    for (int x=0; x < w; ++x)
    enlargedImage.setRGB(x, y, image.getRGB(x/n, y/n));
    return enlargedImage;
}
```

ALGORITHM 6.1 An inefficient method for grey level mapping

```
Define a scaling factor  $a = \sqrt{2^b - 1}$ 
for all pixel coordinates,  $x$  and  $y$ , do
     $g(x, y) = a\sqrt{f(x, y)}$ 
end for
```

Fig 4.1 Algorithm for Grey Level Mapping

```
IntervalTimer timer = new IntervalTimer();
Timer.start();
//some code to be timed
System.out.println(timer.elapsed()); //doesn't stop the clock
//more code to be timed
    System.out.println(timer.stop());
import java.util.Random;
public class MapTest1 {
    public static void randomFill(short[] array) {
        Random random = new Random();
        for (int i = 0; i < array.length; ++i)
            array[i] = (short) random.nextInt(256);
    }
    public static void main(String[] argv) {
        int n = 512;
        if (argv.length > 0)
```

```

n = Integer.parseInt(argv[0]);
// Create image and fill it with random values
int numPixels = n*n;
short[] image = new short[numPixels];
randomFill(image);
// Perform the mapping directly
IntervalTimer timer = new IntervalTimer();
timer.start();
for (int i = 0; i < numPixels; ++i)
image[i] = (short) Math.round(Math.sqrt(image[i]));
System.out.println("direct calculation: " + timer.stop() + " sec");
// Perform the mapping with a lookup table
randomFill(image);
timer.start();
short[] table = new short[256];
for (int i = 0; i < 256; ++i)
table[i] = (short) Math.round(Math.sqrt(i));
for (int i = 0; i < numPixels; ++i)
image[i] = table[image[i]];
System.out.println("lookup table: " + timer.stop() + " sec");
System.exit(0);
}
}

```

4.3 A Java class to perform mapping of grey levels in an image:

```
import java.awt.image.*;
public abstract class GreyMapOp implements BufferedImageOp {
protected byte[] table = new byte[256];
public int getTableEntry(int i) {
if (table[i] < 0)
return 256 + (int) table[i];
else
return (int) table[i];
}
protected void setTableEntry(int i, int value) {
if (value < 0)
table[i] = (byte) 0;
else if (value > 255)
table[i] = (byte) 255;
else
table[i] = (byte) value;
}
public void computeMapping() {
computeMapping(0, 255);
}
public abstract void computeMapping(int low, int high);
public BufferedImage filter(BufferedImage src, BufferedImage dest) {
checkImage(src);
if (dest == null)
dest = createCompatibleDestImage(src, null);
LookupOp operation = new LookupOp(new ByteLookupTable(0, table), null);
operation.filter(src, dest);
return dest;
}
}
```

4.4 Algorithm for Calculation of an image histogram:

```
Create an array histogram with  $2^b$  elements
for all grey levels, i , do
  histogram[i ] = 0;
end for
for all pixel coordinates, x and y, do
  Increment histogram[f (x,y)] by 1
end for
```

4.4.1 Program to perform image histogram:

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.swing.*;
import javax.swing.border.*;
public class HistogramTool extends JFrame implements ActionListener {
  private Histogram histogram; // histogram data to be displayed
  private HistogramView[] view; // plot of the histogram
  private HistogramInfoPane infoPane; // displays value and frequency
  private JPanel mainPane; // contains histogram and info panel
  private JMenu menu; // input/output menu
  private JFileChooser fileChooser = // handles selection of files
  new JFileChooser(System.getProperty("user.dir"));
  public HistogramTool(Histogram theHistogram, String description) {
    super(description); // labels the frame
    // Create components to display histogram and information
    histogram = theHistogram;
    infoPane = new HistogramInfoPane(histogram);
    mainPane = new JPanel(new BorderLayout());
    if (histogram.getNumBands() == 3)
      createMultipleViews(); // three views (R, G, B) in a tabbed pane
    else
      createSingleView();
    mainPane.add(infoPane, BorderLayout.SOUTH);
    setContentPane(mainPane);
    // Add a menu bar to support image input and histogram output
    JMenuBar menuBar = new JMenuBar();
    menuBar.setBorder(new BevelBorder(BevelBorder.RAISED));
    createFileMenu();
    menuBar.add(menu);
    setJMenuBar(menuBar);
  }
}
```



```

addWindowListener(new WindowMonitor());
}
// Creates a single HistogramView object to display a
// greyscale histogram and adds it to the GUI
public void createSingleView() {
view = new HistogramView[1];
view[0] = new HistogramView(histogram, infoPane);
mainPane.add(view[0], BorderLayout.CENTER);
}
// Creates three HistogramView objects for the red, green
// and blue bands of a colour histogram, places these in a
// tabbed pane and adds the tabbed pane to the GUI
public void createMultipleViews() {
view = new HistogramView[3];
Color[] bandColor = { Color.red, Color.green, Color.blue };
String[] tabLabel = { "Red", "Green", "Blue" };
JTabbedPane views = new JTabbedPane(JTabbedPane.BOTTOM);
for (int i = 0; i < 3; ++i) {
view[i] = new HistogramView(histogram, i, infoPane);
view[i].setColor(bandColor[i]);
views.add(tabLabel[i], view[i]);
}
mainPane.add(views, BorderLayout.CENTER);
}
// Creates a menu to support image input, histogram output
// and termination of the application
public void createFileMenu() {
menu = new JMenu("File");
menu.setMnemonic('F');
String[] itemName = { "Load image", "Save histogram", "Exit" };
char[] shortcut = { 'L', 'S', 'X' };
for (int i = 0; i < 3; ++i) {
JMenuItem item = new JMenuItem(itemName[i], shortcut[i]);
item.addActionListener(this);
menu.add(item);
}
}
// Handles Action events triggered by menu selections
public void actionPerformed(ActionEvent event) {
String command = event.getActionCommand();
if (command.startsWith("Load")) {
loadImage();
repaint();
}
else if (command.startsWith("Save")) {
saveHistogram();
repaint();
}
}

```

```

else if (command.equals("Exit")) {
setVisible(false);
dispose();
System.exit(0);
}
}
// Loads a new image, computes its histogram and updates the GUI
public void loadImage() {
fileChooser.setDialogTitle("Load image");
if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
// Load image and compute its histogram
try {
File file = fileChooser.getSelectedFile();
ImageDecoder input =
    ImageFile.createImageDecoder(file.getAbsolutePath());
BufferedImage image = input.decodeAsBufferedImage();
    histogram.computeHistogram(image); setTitle(file.getName());
}
catch (FileNotFoundException e) {
error("File not found.");
return;
}
catch (ImageDecoderException e) {
error("Cannot read this image format.");
return;
}
catch (IOException e) {
error("Failed to read image data.");
return;
}
catch (HistogramException e) {
error("Cannot compute histogram for this image type.");
return;
}
// Rebuild GUI
mainPane.removeAll();
if (histogram.getNumBands() == 3)
createMultipleViews();
else
createSingleView();
mainPane.add(infoPane, BorderLayout.SOUTH);
mainPane.invalidate();
validate();
pack();
}
}
// Saves current histogram to a file selected by user
public void saveHistogram() {

```

```

    if (histogram.getNumBands() == 0) {
error("No histogram data to save!");
return;
}
else {
fileChooser.setDialogTitle("Save histogram");
if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
try {
File file = fileChooser.getSelectedFile();
if (file.exists()) {
int response = JOptionPane.showConfirmDialog(this,
"File will be overwritten! Are you sure?", "File exists",
JOptionPane.OK_CANCEL_OPTION);
if (response != JOptionPane.OK_OPTION)
return;
}
}
}
histogram.write(new FileWriter(file));
fileChooser.rescanCurrentDirectory();
}
catch (IOException e) {
error("Cannot open output file.");
}
}
}
}
// Displays an error message in a dialog box
public void error(String message) {
JOptionPane.showMessageDialog(this, message, "Error",
JOptionPane.ERROR_MESSAGE);
}
public static void main(String[] argv) {
if (argv.length > 0) {
try {
ImageDecoder input = ImageFile.createImageDecoder(argv[0]);
BufferedImage image = input.decodeAsBufferedImage();
Histogram hist = new Histogram(image);
HistogramTool histTool = new HistogramTool(hist, argv[0]);
histTool.pack();
histTool.setVisible(true);
}
catch (Exception e) {
System.err.println(e);
System.exit(1);
}
}
else {
Histogram hist = new Histogram();
HistogramTool histTool = new HistogramTool(hist, "HistogramTool");
}
}
}
}

```

```
histTool.pack();  
histTool.setVisible(true);  
}  
}  
}
```

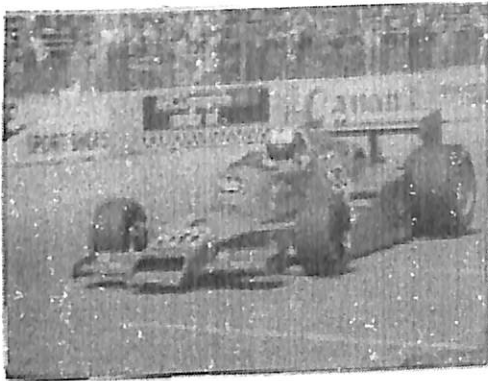
4.5 Program for Histogram Analysis:

```
import java.awt.image.BufferedImage;
import java.io.FileWriter;
public class CalcHist {
public static void main(String[] argv) {
if (argv.length > 1) {
try {
ImageDecoder input = ImageFile.createImageDecoder(argv[0]);
BufferedImage image = input.decodeAsBufferedImage();
Histogram histogram = new Histogram(image);
FileWriter histFile = new FileWriter(argv[1]);
histogram.write(histFile);
if (argv.length > 2) {
FileWriter cumHistFile = new FileWriter(argv[2]);
histogram.writeCumulative(cumHistFile);
}
System.exit(0);
}
catch (Exception e) {
System.err.println(e);
System.exit(1);
}
}
else {
System.err.println(
"usage: java CalcHist <imageFile> <histFile> [<cumHistFile>]");
System.exit(1);
}
}
}
```

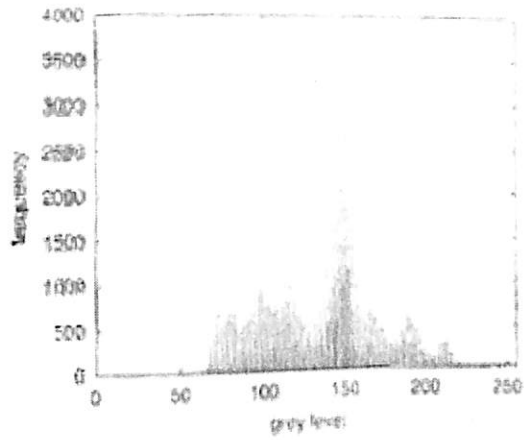
ALGORITHM Histogram equalisation

```
Compute a scaling factor,  $\alpha = 255 / \text{number of pixels}$ 
Calculate histogram using Algorithm 6.3
 $c[0] = \alpha * \text{histogram}[0]$ 
for all remaining grey levels,  $i$ , do
 $c[i] = c[i - 1] + \alpha * \text{histogram}[i]$ 
end for
for all pixel coordinates,  $x$  and  $y$ , do
 $g(x, y) = c[f(x, y)]$ 
end for
```

Fig 4.2

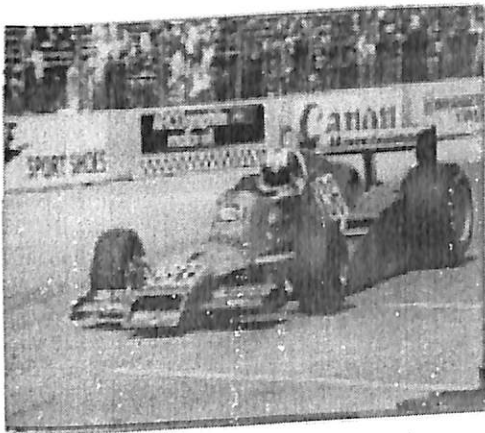


(a)

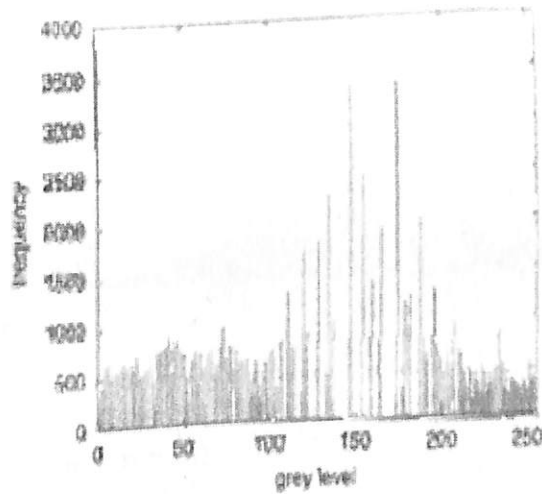


(b)

(a) An unequalised image. (b) Its histogram.



(a)



(b)

(a) An equalised image. (b) Its histogram.

Fig 4.3 Unequalized and equalized images and their histograms

ALGORITHM Calculation of a colour image histogram.

```
Create a 3D array histogram of dimensions  $2^b \times 2^b \times 2^b$ 
for all red values, r, do
  for all green values, g, do
    for all blue values, b, do
      histogram[r][g][b] = 0
    end for
  end for
end for
for all pixel coordinates, x and y, do
  Find r, the red component of f(x, y)
  Find g, the green component of f(x, y)
  Find b, the blue component of f(x, y)
  Increment histogram[r][g][b] by 1
end for
```

Fig 4.4 Algorithm for color image histogram

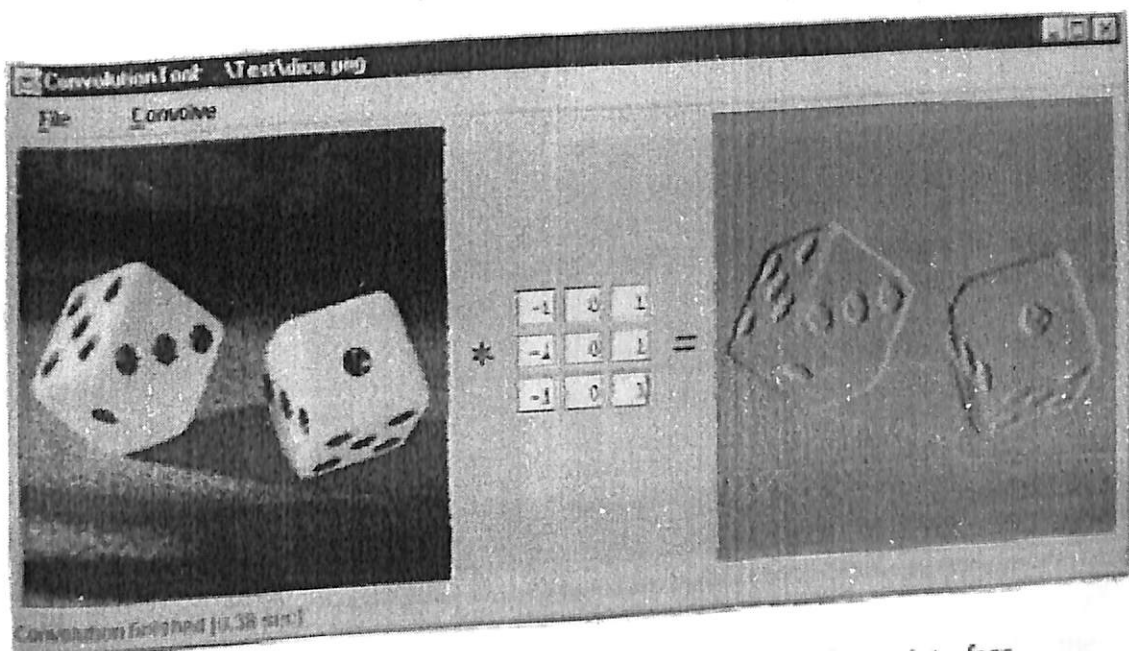
4.6 A convolution application with a command line interface:

```
import java.awt.image.*;
import java.io.*;
import com.pearsoneduc.ip.io.*;
import com.pearsoneduc.ip.op.*;
import com.pearsoneduc.ip.util.IntervalTimer;
public class Convolve {
public static void main(String[] argv) {
if (argv.length > 5) {
try {
// Parse command line arguments
ImageDecoder input = ImageFile.createImageDecoder(argv[0]);
ImageEncoder output = ImageFile.createImageEncoder(argv[1]);
Reader kernelInput = new FileReader(argv[2]);
boolean normaliseKernel = (Integer.parseInt(argv[3]) != 0);
int borderStrategy = Math.max(1, Math.min(4, Integer.parseInt(argv[4]]));
int rescaleStrategy =
Math.max(1, Math.min(3, Integer.parseInt(argv[5]]));
// Load image and kernel
BufferedImage inputImage = input.decodeAsBufferedImage();
Kernel kernel =
StandardKernel.createKernel(kernelInput, normaliseKernel);
// Create convolution operator and convolve image
ConvolutionOp convOp = new ConvolutionOp(kernel,
borderStrategy, ConvolutionOp.SINGLE_PASS, rescaleStrategy);
```

```

IntervalTimer timer = new IntervalTimer();
timer.start();
BufferedImage outputImage = convOp.filter(inputImage, null);
System.out.println("Convolution finished [" + timer.stop() + " sec]");
// Write results to output file
output.encode(outputImage);
System.exit(0);
}
catch (Exception e) {
System.err.println(e);
System.exit(1);
}
}
else {
System.err.println("usage: java Convolve " +
"<infile> <outfile> <kernel> <norm> <border> <rescale>");
System.exit(1);
}
}
}

```



A convolution application with a graphical user interface.

Fig 4.5

4.7 Image Rotation by forward mapping:

```
public static BufferedImage rotate(BufferedImage input, double angle){
int width = input.getWidth();
int height = input.getHeight();
BufferedImage output = new BufferedImage(width, height, input.getType());
double a0 = Math.cos(angle*Math.PI/180.0);
double b0 = Math.sin(angle*Math.PI/180.0);
double a1 = -b0, b1 = a0;
int rx, ry;
for (int y = 0; y < height; ++y)
for (int x = 0; x < width; ++x){
rx = (int) Math.round(a0*x + a1*y);
ry = (int) Math.round(b0*x + b1*y);
if (rx >= 0 && rx < width && ry >= 0 && ry < height)
output.setRGB(rx, ry, input.getRGB(x,y));
}
return output;
}
```



(a)



(b)



(c)

Fig 4.6 Image autorotation: (a) Sample image (b) rotated image with pixels outside the coordinate plane (c) rotated image with pixels completely visible.

4.8 A Java class to perform grey level thresholding:

```
public class ThresholdOp extends GreyMapOp {
public ThresholdOp(int threshold) {
computeMapping(threshold, 255);
}
public ThresholdOp(int low, int high) {
computeMapping(low, high);
}
public void setThreshold(int threshold) {
computeMapping(threshold, 255);
}
public void setThresholds(int low, int high) {
computeMapping(low, high);
}
public void computeMapping(int low, int high) {
if (low < 0 || high > 255 || low >= high)
throw new java.awt.image.ImagingOpException("invalid thresholds");
int i;
for (i = 0; i < low; ++i)
table[i] = (byte) 0;
for (; i <= high; ++i)
table[i] = (byte) 255;
for (; i < 256; ++i)
table[i] = (byte) 0;
}
}
```

4.9 Java code to perform connected region labeling, taken from the Region Label class.

```
public BufferedImage filter(BufferedImage src, BufferedImage dest) {
checkImage(src);
if (dest == null)
dest = createCompatibleDestImage(src, null);
width = src.getWidth();
height = src.getHeight();
WritableRaster in = src.copyData(null);
WritableRaster out = dest.getRaster();
int n = 1;
for (int y = 0; y < height; ++y)
for (int x = 0; x < width; ++x)
if (in.getSample(x, y, 0) > 0) {
label(in, out, x, y, n);
++n;
if (n > MAX_REGIONS)
}
```

```

return dest;
}
return dest;
}
private void label(WritableRaster in, WritableRaster out, int x, int y, int n) {
in.setSample(x, y, 0, 0);
out.setSample(x, y, 0, n);
int j, k;
for (int i = 0; i < connectivity; ++i) {
j = x + delta[i].x;
k = y + delta[i].y;
if (inImage(j, k) && in.getSample(j, k, 0) > 0)
label(in, out, j, k, n);
}
}
private final boolean inImage(int x, int y) {
return x >= 0 && x < width && y >= 0 && y < height;
}
}

```

ALGORITHM

Region growing.

Let f be an image for which regions are to be grown
Define a set of regions, R_1, R_2, \dots, R_n , each consisting of a single seed pixel
repeat
 for $i = 1$ to n **do**
 for each pixel, p , at the border of R_i **do**
 for all neighbours of p **do**
 Let x, y be the neighbour's coordinates
 Let μ_i be the mean grey level of pixels in R_i
 if the neighbour is unassigned **and** $|f(x, y) - \mu_i| \leq \Delta$ **then**
 Add neighbour to R_i
 Update μ_i
 end if
 end for
 end for
 end for
until no more pixels are being assigned to regions

Fig 4.7: Binary Erode Op's filter method.

```

public BufferedImage filter(BufferedImage src, BufferedImage dest) {
    checkImage(src);
    if (dest == null)
        dest = createCompatibleDestImage(src, null);
    int w = src.getWidth();
    int h = src.getHeight();
    Raster srcRaster = src.getRaster();
    WritableRaster destRaster = dest.getRaster();
    // Determine range of pixels for which operation can be performed
    Point origin = structElement.getOrigin(null);
    int xmin = Math.max(origin.x, 0);
    int ymin = Math.max(origin.y, 0);
    int xmax = origin.x + w - structElement.getWidth();
    int ymax = origin.y + h - structElement.getHeight();
    xmax = Math.min(w-1, xmax);
    ymax = Math.min(h-1, ymax);
    // Fit structuring element into source image
    for (int y = ymin; y <= ymax; ++y)
        for (int x = xmin; x <= xmax; ++x)
            if (structElement.fits(srcRaster, x, y))
                destRaster.setSample(x, y, 0, nonZeroValue);
    return dest;
}

```

filter() method of BinaryOpenOp.

```

public BufferedImage filter(BufferedImage src, BufferedImage dest){
    BinaryErodeOp erodeOp = new BinaryErodeOp(structElement);
    BinaryDilateOp dilateOp = new BinaryDilateOp(structElement);
    if (dest == null)
        dest = createCompatibleDestImage(src, null);
    return dilateOp.filter(erodeOp.filter(src, null), dest);
}

```

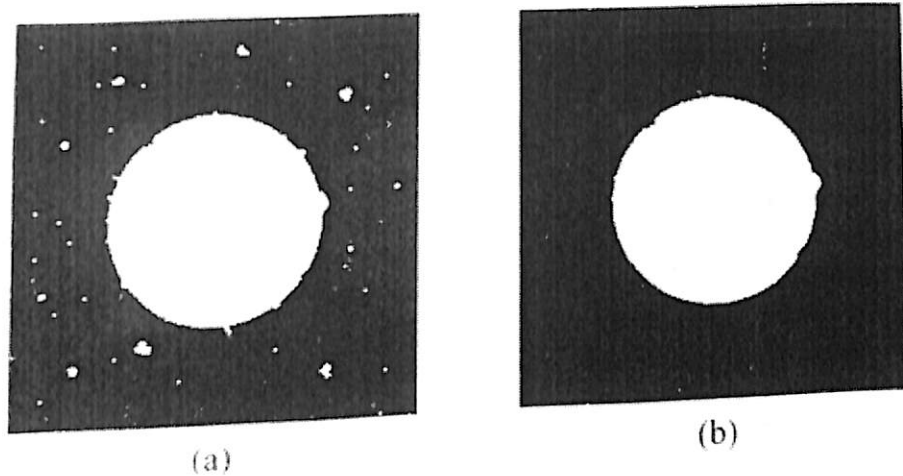


Figure 4.8 Example of morphological filtering. (a) Input image, a cookie surrounded by crumbs. (b) Result of opening the cookie image.

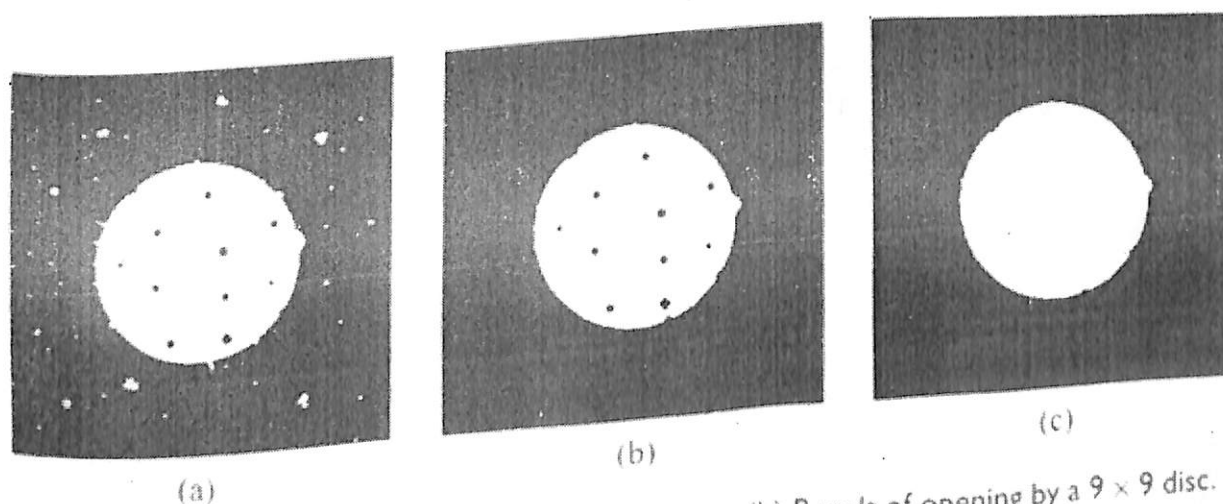


Figure 4.9 Dealing with holes. (a) Input image. (b) Result of opening by a 9×9 disc. (c) Result of closing the opened image, again using a 9×9 disc.

4.10 Gaussian Filter:

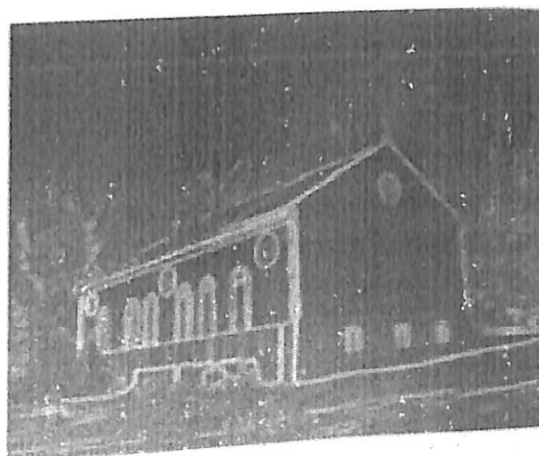
A **Gaussian filter** is a filter whose impulse response is a Gaussian function. Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time. This behavior is closely connected to the fact that the Gaussian filter has the minimum possible group delay. Mathematically, a Gaussian filter modifies the input signal by convolution with a Gaussian function; this transformation is also known as the Weierstrass transform.

Class to perform Gaussian low pass filtering:

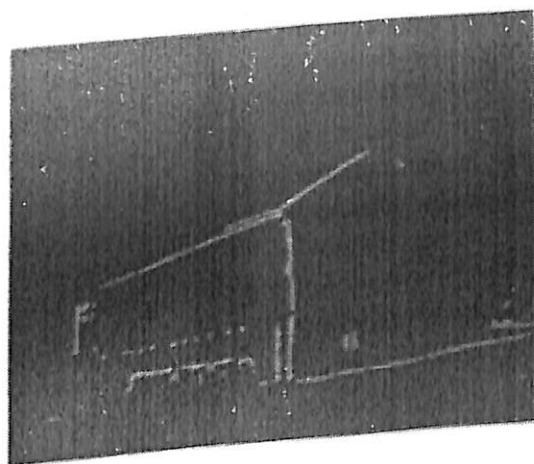
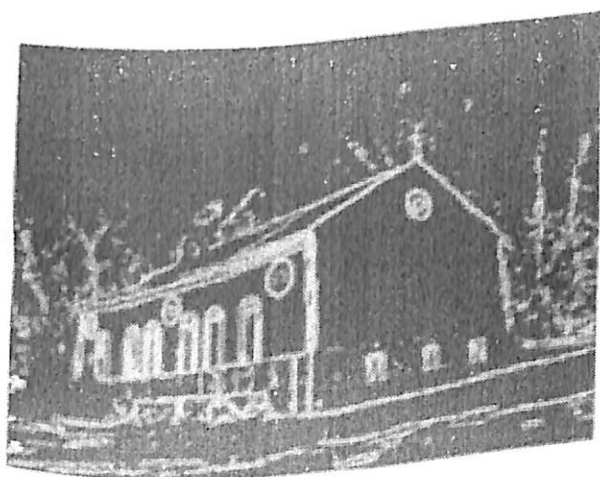
```
public class GaussianKernel extends StandardKernel {
public GaussianKernel() {
this(1.0f);
}
public GaussianKernel(float sigma) {
super(getSize(sigma), getSize(sigma), createKernelData(sigma));
}
public static int getSize(float sigma) {
int radius = (int) Math.ceil(4.0f*sigma);
return 2*radius+1;
}
public static float[] createKernelData(float sigma) {
int n = (int) Math.ceil(4.0f*sigma);
int size = 2*n+1;
float[] data = new float[size*size];
double r, s = 2.0*sigma*sigma;
float norm = 0.0f;
int i = 0;
for (int y = -n; y <= n; ++y)
for (int x = -n; x <= n; ++x, ++i) {
r = Math.sqrt(x*x + y*y);
data[i] = (float) Math.exp(-r*r/s);
norm += data[i];
}
for (i = 0; i < size*size; ++i)
data[i] /= norm;
return data;
}
public static void main(String[] argv) {
float sigma = 1.0f;
if (argv.length > 0)
sigma = Float.valueOf(argv[0]).floatValue();
StandardKernel kernel = new GaussianKernel(sigma);
kernel.write(new java.io.OutputStreamWriter(System.out));
} }
}
```

A program for mean filter application:

```
import java.awt.image.*;
import com.pearsoneduc.ip.io.*;
import com.pearsoneduc.ip.op.MeanKernel;
import com.pearsoneduc.ip.util.IntervalTimer;
public class MeanFilter {
public static void main(String[] argv) {
if (argv.length > 3) {
try {
ImageDecoder input = ImageFile.createImageDecoder(argv[0]);
ImageEncoder output = ImageFile.createImageEncoder(argv[1]);
int w = Integer.parseInt(argv[2]);
int h = Integer.parseInt(argv[3]);
BufferedImage inputImage = input.decodeAsBufferedImage();
Kernel kernel = new MeanKernel(w, h);
ConvolveOp blurOp = new ConvolveOp(kernel);
IntervalTimer timer = new IntervalTimer();
timer.start();
BufferedImage outputImage = blurOp.filter(inputImage, null);
System.out.println("Mean filtering finished [" +
timer.stop() + " sec]");
output.encode(outputImage);
System.exit(0);
}
catch (Exception e) {
System.err.println(e);
System.exit(1);
}
}
else {
System.err.println("usage: java MeanFilter <infile> <outfile> <w> <h>");
System.exit(1);
}
}
```



(a) The barn image (b) Gradient magnitudes for the barn image, scaled to 0-255



Edge maps created by thresholding the gradient magnitudes
(a) Threshold of 50 (b) Threshold of 150

Fig 4.10 Image Thresholding using Gaussian Filter

CONCLUSION

The proposed tool is able to perform filtering operations on an image by using Gaussian filtering technique. It is efficient to change the contrast of the image using histogram analysis and to rotate the image by using image rotation method. The tool can also be used to compress the image size. The tool is also able to perform blurring and deblurring operation on an image.

The Graphical user interface developed for this tool is very user friendly and easy to operate. The use of buttons to select specified operations and frame to display an image makes the tool attractive and at the same time easy to work upon.

Some problems faced with the tool are that it is able to perform limited tasks only. Also the format of the images supported is .jpg, .png, .piff. Images other than the specified formats may not work with the tool.

REFERENCES

- [1] *SEGMENTATION OF TEXT FROM COMPOUND IMAGES* by Dr.N.Krishnan, C. Nelson Kennedy Babu , S.Ravi and JosphineThavamani presented in International Conference on Computational Intelligence and Multimedia Applications 2007.
- [2] Fujii, Masafumi., Wolfgang, J. R., and Hofer. 2001. Filed-Singularity correction in 2-D time-domain Haar-wavelet modeling of waveguide components. *IEEE Transactions on Microwave Theory and Techniques*, 49, 4.
- [3] Chen, P. Y. and Liao, E. C. 2002. A new algorithm for Haar discrete wavelet transform. *IEEE International Symposium on Intelligent Signal Processing and Communication Systems*, 21, 24: 453-457.
- [4] Gonzalez, Rafael C. and Woods, Richard E [2011]. *Digital Image Processing*, Pearson Education, Third Edition.
- [5] Andrews, H.C [1970]. *Computer Technique in Image Processing*, Academic Press, New York.