

NEAR EAST UNIVERSITY



**GRADUATE SCHOOL OF APPLIED
AND SOCIAL SCIENCES**

OPTIMUM EDGE DETECTION

EMAD ADNAN AALIAN

Master Thesis

Department of Computer Engineering

Nicosia 2004

Emad Adnan Aalian : Optimum Edge Detection

**Approval of the Graduate School of Applied and
Social Sciences**

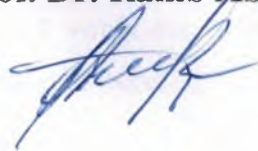
**Prof. Dr. Fakhraddin Mamedov
Director**

A handwritten signature in black ink, appearing to be 'F. Mamedov', written over a circular blue stamp of the Graduate School of Applied and Social Sciences.

**We certify this thesis is satisfactory for the award of the
Degree of Master of Science in Computer Engineering**

Examining Committee in charge:

Assoc. Prof. Dr. Rahib Abiyev, Committee Chairman, Chairman of
Computer Engineering Department, NEU

A handwritten signature in blue ink, appearing to be 'Rahib Abiyev', written over a circular blue stamp of the Graduate School of Applied and Social Sciences.

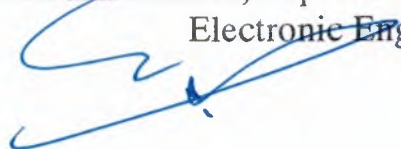
Assist. Prof. Dr. Firudin Muradov, Committee Member, Computer
Engineering Department, NEU

A handwritten signature in blue ink, appearing to be 'F. Muradov', written over a circular blue stamp of the Graduate School of Applied and Social Sciences.

Assoc. Prof. Dr. Sameer Ikhdair, Committee Member, Electrical &
Electronic Engineering Department, NEU

A handwritten signature in blue ink, appearing to be 'Sameer Ikhdair', written over a circular blue stamp of the Graduate School of Applied and Social Sciences.

Assoc. Prof. Dr. Adnan Khashman, Supervisor, Chairman of Electrical &
Electronic Engineering Department, NEU

A handwritten signature in blue ink, appearing to be 'Adnan Khashman', written over a circular blue stamp of the Graduate School of Applied and Social Sciences.

ACKNOWLEDGEMENTS

I would like to express my grateful thanks to my supervisor *Assoc. Prof. Dr. Adnan KHASHMAN* for his guidance and encourgment through my thesis preparation.

I would like to thank my family, specially my father and mother for their moral and financial support and patience, and also my brother Alaa who help me very much to finish my thesis.

I would like to thank my friend Marwan, Obid, Ahmad, Guven, and for all friends that i met in North Cyprus.

Also I would to thank my family-in-law in Cyprus specially my mother-in-law Shadye and to Selma for her moral support.

I would like to thank my work friends egemen ltd family Attilla, Ayla, Erkan, Ibrahim, Nasir, Sercan, Inci, Meltem, Serkan, Izzet, Vahbi, Nulifer, Nazaket, Hasan, Akin and to Ozcanlar in Famagusta

I would to thank Mr. Tayseer SHANABLAH for his support in solving the problems that I faced during my both undergraduate and postgraduate studies.

Finally, I would like to thank Near East University, its whole staff, students, and every one that I met there.

TABLE OF CONTENTS

ACKNOWLEDGMENTS.....	i
TABLE OF CONTENTS.....	ii
ABSTRACT.....	iv
INTRODUCTION.....	1
1. IMAGE PROCESSING TECHNIQUES.....	5
1.1. Overview.....	5
1.2. Human visual Perception.....	5
1.2.1. The Human Visual System.....	6
1.2.2. Image Representation.....	7
1.2.2.1. Binary Images.....	8
1.2.2.2. Gray-scale Images.....	8
1.2.2.3. Color Images.....	9
1.2.2.4. Multispectral images.....	10
1.3. Computer imaging.....	11
1.3.1. Computer Vision.....	12
1.3.2. Image processing.....	12
1.4. Computer imaging system	15
1.5. Summary.....	16
2. EDGE DETECTION OPERATORS.....	17
2.1. Overview	17
2.2. Edge Detectors.....	17
2.2.1. Laplacian of Gaussian.....	20
2.2.2. Roberts Cross Edge Detector.....	24
2.2.3. Sobel Edge Detector.....	27
2.2.4. Prewitt Edge Detector.....	29
2.3. Summary.....	32

3. THE DEVELOPED SOFTWARE APPLICATION.....	33
3.1. Overview.....	33
3.2. Software Specifications.....	33
3.3. Program Flowchart.....	35
3.4. The Program Interfaces.....	36
3.5. Summary.....	40
4. RESULTS AND COMPARISON.....	41
4.1. Overview.....	41
4.2. Image capturing enviroment.....	41
4.3. Camera details.....	42
4.4. Applying the developed software to the images.....	43
4.5. Optimum edge detector criteria.....	46
4.5.1. Continuity of the edges.....	46
4.5.2. Noise reduction.....	46
4.5.3. Lighting.....	47
4.5.4. Contrast.....	47
4.5.5. convolution speed.....	47
4.6 Comparison and Analysis.....	48
4.7. Additional Example for Edge Detection.....	51
4.7.1. Results Analysis.....	54
4.8 Summary.....	56
CONCLUSION.....	57
REFERENCES.....	58
APPENDIX A	59
APPENDEX B.....	68

ABSTRACT

Receiving and organizing the visual sensory data into patterns is an important part of human perception. Very often there is homogeneity between regions within these patterns with respect to the data. This homogeneity is coming from that these regions have constant or nearly constant gray level. Also it may result from textural properties of regions within the all image. In both cases it might be possible to segment the image according to a definable homogeneous characteristic. For example, in thresholding, an effort is made to separate the image into object and background by choosing the appropriate threshold value. Through this process we have to consider the degree of darkness, in a bi-level sense, as a measure of homogeneity. If an image happens to contain two objects both are darker the background, the operator can detect both of them, on the other hand, if one of them is lighter than background, it will be impossible for it to be detected.

After assigning the threshold the objects within images can simply be recognized by their crude outlines or edges. Thus, edge detection is considered as a low-level process in image recognition as it applied prior to other image processing techniques.

INTRODUCTION

A commonly held belief that edge detection is the first step in vision processing has fueled a long search for a good edge detection algorithm. Edge detection refers to the process of identifying and locating sharp discontinuities in an image. Edges are defined as discontinuities in the image intensity due to changes in scene structure. These discontinuities originate from different scene features and can describe the information that an image of the external world contains. Enhancement and smoothing attempt to make these discontinuities apparent to the detector, so that desirable edges can be extracted.

Edge detectors, where ground truth not available, are evaluated by their ability to produce edges that provide for the quick and accurate recognition, as judged by humans, of a three dimensional object from a grayscale image of the object in its natural setting. From a complete evaluation methodology was determined that a statistically significant difference exists in the relative performance of edge detection algorithms. The relative performance depends on the method used for selecting the input parameters, as significantly better performance was attained by the edge detectors when the parameters of each were optimized individually for each image than when a single set of parameters was optimized for the entire set of images.

Edge Detection Important For Feature Extraction and Subsequent Vision Tasks: Texture Analysis, Motion Detection/ Estimation, Stereopsis and Recognition in both Machine and Biological Vision Systems.

From 1960 to 1975 the discrete gradients (Roberts 1965, Prewitt 1970, Sobel 1970, Kirsch 1971) and Laplacians were invented and used as edge detectors. It is considered an edge is a jump in intensity. The cross section of an edge has the shape of a ramp. An ideal edge is a discontinuity (i.e., a ramp with an infinite slope). The first derivative assumes a local maximum at an edge. For a continuous image, where x and y are the row and column coordinates respectively, we typically consider the two directional derivatives and. Of particular interest in edge detection are two functions that can be expressed in terms of these directional derivatives: the gradient magnitude and the gradient orientation.

In 1980 Marr & Hildreth invented their new edge detector. In the Marr-Hildreth operator a maximum of the first derivative will occur at a zero crossing of the second derivative. To get both horizontal and vertical edges we look at second derivatives in both the x and y directions. The Laplacian is linear and rotationally symmetric. Thus, we search for the zero crossings of the image that is first smoothed with a Gaussian mask and then the second derivative is calculated; or we can convolve the image with the Laplacian of the Gaussian, also known as the LoG operator.

The Marr-Hildreth operator became widely used for the following reasons:

- Marr had considerable reputation.
- Researchers found receptive fields in the eyes of animals (usually cats and macaque monkeys) that behaved just like this operator.
- The operator is symmetric. Edges are found in all orientations, unlike the first derivative operators which are directional.
- Zero crossings of the second derivative are simpler to determine than are maxima in the first derivative; all that needs to be done is to look for a sign change in the signal.

During 1983 John Canny did his Masters degree at MIT. He treated edge detection as a signal processing problem and aimed to design the 'optimal' edge detector. He formally specified an objective function to be optimized and used this to design the operator.

The objective function was designed to achieve the following optimization constraints:

- Maximize the signal to noise ratio to give good detection. This favors the marking of true positives.
- Achieve good localization to accurately mark edges.
- Minimize the number of responses to a single edge. This favors the identification of true negatives, that is, non-edges are not marked.

Note a difference-of-boxes operator will maximize the signal to noise ratio, but will give several responses to a single edge.

After some complicated analysis Canny came up with a function that was the sum of 4 exponential terms. However, this function looked very much like a first derivative of a Gaussian! So this is what ended up being used.

Between 1979 - 1987 Morphological Gradient & Laplacian were discovered. Beucher (1979) showed that with the use of mathematical morphology's dilation and erosion we can have a morphological gradient. Lee, Haralick & Sapiro (1987) used a linear smoothing operator and derived to better results.

Edge detection for gray-level image by morphological gradient method has better results than the conventional edge detection methods. The reason is that the morphological approaches eliminated the isolated points, which we often consider as noise. Vliet, Young and Beckers (1989), approached the second derivative (laplacian with a morphological laplacian).

Since 1985 till the present time the linear & non-linear scale-space is the newest method for edge detection. A vision system for handling objects of different sizes and at different distances needs a way to control the scale(s) at which the world is observed. Why should one represent a signal at multiple scales when all information is present in the original data anyway? The major reason for this is to explicitly represent the multi-scale aspect of real world images. Another aim is to simplify further processing by removing unnecessary and disturbing details.

In this project a software application were developed which integrate the common edges detectors. Five detectors were included in the program.

Chapter one has a general information and some basic definitions were discussed in order to supply the reader with enough knowledge that can help in understanding the rest of the chapters. The human vision perception and human vision system were explained briefly. After that it shows the image is represented. It make a list of image types that will be considered. The importance of computer imaging and its two types, computer vision and image processing, were discussed.

Chapter two explains the edge detection idea. It shows the types of edge detector in general. Such as Laplacian of the Gaussian, Roberts Cross, Sobel, and Prewitt. The

common names of detectors, Brief description, how it works, and guidelines for use were explained.

Chapter three shows the developed software application. It explains the software specifications with showing the way that detectors work and their masks. program interface which consist mainly from three buttons as open, save and reset were explained. the program flowchart made it easy for the reader to follow the mechanism of the program and how it works.

Chapter four contains the extract of the whole thesis. It starts with showing the box where our two images were captured. Then it lists the details of the camera which had been used to capture the images. Later on it shows the result of applying each of the five operators on the two captured images. A comparison and analysis is made between these results to determine the optimum edge detector after explaining the criteria of choosing the optimum edge detector.

The objectives of the work presented within this thesis are:

- Develop a software application that integrate the common edge detectors
- Provide a userfriendly GUI for further uses
- To compare & analyse the edge operators using own captured unique images.
- To create a criteria for optimum edge detector

CHAPTER 1: IMAGE PROCESSING TECHNIQUES

1.1. Overview

Computer imaging becomes one of the important things in the human life as it allowed us to be in touch with the whole world by sending and receiving data. It can be purely computerized or it can involve human being in its loop.

For the computer vision it will not involve any human and the decision will be taken just by the computer by analyzing and comparing. It involves many topics like image analysis, which involve other topics under it.

For the human vision or image processing, it will be different, as it will involve human being in its stages for examining. It involves many topics too like image restoration, image enhancement, image compression, and image segmentation. So in order to know how these processes apply we have to know first how the human visual system works and the image representation in the human eyes.

This chapter presents the basic information for the way the human visual system works in order in addition to an introduction to computer vision.

1.2. Human visual Perception

Human visual perception is something most of us take for granted. We do not think about how the makeup of the physiological systems affects what we see and how we see it. Although human visual perception encompasses both physiological and physiological components, we will focus primarily on the physiological aspects, which are more easily quantifiable, using the current models available for understanding the systems.

We have briefly discussed the need to understand how we perceive visual information in order to design compression algorithms that compact the data as much as possible but still retain all the necessary visual information. This is desired for both transmission and storage economy. Images are often transmitted over the airwaves and will be transmitted more frequently via the World Wide Web (internet), but people do

not want to wait minutes or hours for the images. Additionally, the storage requirements can become overwhelming without compression. For example, an 8-bit monochrome image, with a resolution of 512 pixels wide by 512 pixels high, requires one quarter of a megabyte of data. If we make this a color image, it requires three quarters of a megabyte of data (one quarter for each of three colors bands-red, green, and blue). With many applications requiring the capability to process thousands of images in relatively short periods of time, the need to reduce data is apparent. For the development of image enhancement algorithms, we also have the need to understand how the human visual system works. We need to know the types of operations that are likely to improve an image visually, and this can be achieved only by understanding how the information is perceived.

1.2.1. The Human Visual System

System has two primary components-the eye and the brain, which are connected by the optic nerve. The structure that we know the most about is the image receiving sensor the human eye. The brain can be thought of as being an information processing unit, analogous to the computer in our computer imaging system. These two are connected by the optic nerve, which is really a bundle of nerves that contains the pathways for the visual information to travel from the receiving sensor (the eye) to the processor (the brain). The way the human visual system works follows:

- 1) Light energy is focused by the lens of the eye onto the sensors on the retina,
- 2) These sensors respond to this light energy by an electrochemical reaction that sends an electrical signal down the optic nerve to the brain,
- 3) The brain uses these nerve signals to create neurological patterns that we perceive as images. The visible light energy corresponds to an electromagnetic wave that falls into the wavelength range of about 380 to 825 nanometers, although the response above 700 nanometers is minimal.

In imaging systems the spectrum is often divided into various *spectral bands*, where each band is defined by a range on the wavelengths (or frequency). For example, we can divide the visible spectrum into roughly three bands corresponding to blue (400 to 500 nm), and green (500 to 600 nm), and Red (600 to 700 nm). The eye has two

primary type of light energy receptors, or photoreceptive, which respond to the incoming light energy and convert it into electrical energy via a complex electrochemical process. These two types of sensor are called rods and cones. The sensors are distributed across the retina, the inner backside of the eye where the light energy falls after being focused by the lens. The *cones* are primarily used for daylight vision, are sensitive to color, are concentrated in the central region of the eye, and have a high resolution capability. The *rods* are used in night vision, see only brightness (not color), are distributed across the retina, and have medium to low level resolution.

1.2.2. Image Representation

We have seen that the human visual system receives an input image as a collection of spatially distributed light energy; this form is called an optical image. Optical images are the types we deal with everyday cameras capture them, monitors display them, and we see them. We know that these optical images are represented as video information in the form of analog electrical signals and have seen how these are sampled to generate the digital image. $I(r,c)$

The digital image is represented as a two-dimensional array of data, where each pixel value corresponds to the brightness of the image at the point (r, c) . In linear algebra terms, a two-dimensional array like our image model $I(r, c)$ is referred to as a *matrix*, and one row (or column) is called a *vector*. This image model is for monochrome (one-color, this is what we normally refer to as black and white) image data, but we have other types of image data that require extensions or modifications to this model. Typically, these are multiband images (color, multispectral), and they can be modeled by a different $I(r, c)$ function corresponding to each operate band of brightness information. The image types we will consider are:

- 1) Binary images.
- 2) Gray-scale images.
- 3) Color images.
- 4) Multispectral images.

1.2.2.1. Binary Images

Binary images are the simplest type of images and can take on two values, typically black and white, or '0' and '1.' A binary image is referred to as a 1 bit/pixel image because it takes only 1 binary digit to represent each pixel. These types of images are most frequently used in computer vision applications where the only information required for the task is general shape, or outline, information. For example, to position a robotic gripper to grasp an object, to check a manufactured object for deformations, for facsimile (FAX) images, or in optical character recognition (OCR). Binary images are often created from Gray-scale images via a threshold operation, where every pixel above the threshold value is turned white ('1'), and those below it are turned black ('0').

1.2.2.2. Gray-scale Images

Gray-scale images are referred to as monochrome, or one-color, images. They contain brightness information only, no color information. The number of bits used for each pixel determines the number of different brightness levels available. The typical image contains 8 bits/pixel data, which allows us to have 256 (0-255) different brightness (gray) levels. This representation provides more than adequate brightness resolution, in terms of the human visual system's requirements, and provides a incise margins by allowing for approximately twice as many gray levels as required. This noise margin is useful in real-world applications because of the many different types of noise (false information in the signal) inherent in real systems. Additionally, the 8-bit representation is typical due to the fact that the *byte*, which corresponds to 8-bits of data, is the standard small unit in the world of digital computers.

In certain applications, such as medical imaging or astronomy, 12 or 16 bits/pixel representations are used. These extra brightness levels become useful only when the image is "blown-up," that is when a small section of the image is made much larger. In this case we may be able to discern details that would be missing without this additional brightness resolution. Of course, to be useful, this also requires a higher level of spatial resolution (number of pixels). If we go beyond these levels of brightness resolution, we typically divide the light energy into different bands, where each band refers to a specific subsection of the visible image spectrum.

1.2.2.3. Color Images

Color images can be modeled as three-band monochrome image data, where each band of data corresponds to a different color. The actual information stored in the digital image data is the brightness information in each spectral band. When the image is displayed, the corresponding brightness information is displayed on the screen by picture elements that emit light energy corresponding to that particular color. Typical color images are represented as red, green, and blue, or RGB images. Using the 8-bit monochrome standard as a model, the corresponding color image would have 24 bits/pixel-f-bits for each of the three color bands (red, green, and blue).

For many applications, RGB color information is transformed into a mathematical space that decouples the brightness information from the color information. After this is done, the image information consists of a one-dimensional brightness, or luminance, space and a two dimensional color space. Now the two-dimensional color space does not contain any brightness information, but it typically contains information regarding the relative amounts of the different colors. An additional benefit of modeling the color information in this manner is that it creates a more people-oriented way of describing the colors.

For example, the hue/saturation/lightness (HSL) color transform allows us to describe colors in terms that we can more readily understand (see Figure 1.1). The lightness is the brightness of the color, and the hue is what we normally think of as "color" (for example green, blue, or orange). The saturation is a measure of how much white is in the color (for example, pink is red with more white, so it is less saturated than a pure red). Most people can relate to this method of describing color. For example, "A deep, bright orange would have a large intensity ("bright"), a hue of orange and a high value of saturation (deep). We can picture this color in our minds but if we defined this color in terms of its RGB components, $R = 245$, $G = 110$, and $B = 20$, most people would have no idea how this color appears. Because the HSL color space was developed based on heuristics relating to human perception, various methods are available to transform RGB pixel values into the HSL color space. Most of these are algorithmic in nature and are geometric approximations to mapping the RGB color cube into some HSL-type color space.

HSL color Space

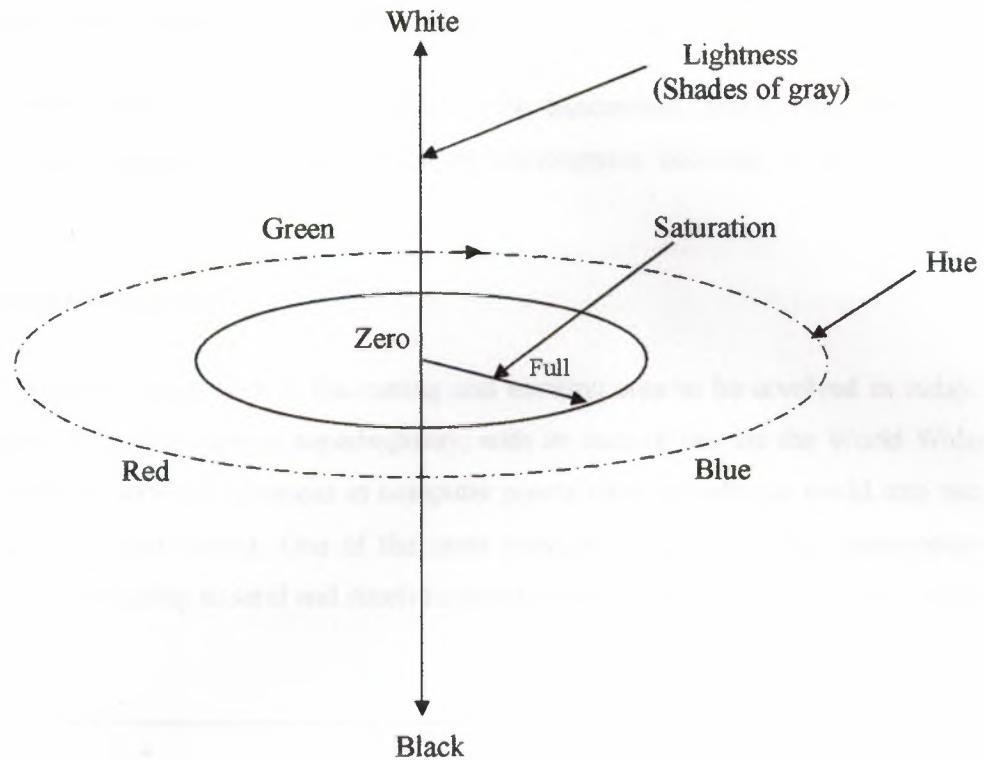


Figure1.1. HSL Color Space

1.2.2.4. Multispectral images

Multispectral images typically contain information outside the normal human perceptual range. This may include infrared, ultraviolet, X-ray, acoustic or radar data. These are not images in the usual sense because the information represented is not directly visible by the human system.

Source for these types of images include satellite systems, underwater sonar systems, various types of airborne radar, infrared imaging systems, and medical diagnostic imaging systems. The number of bands into which the data are divided is strictly a function of the sensitivity of the imaging sensors used to capture the images.

For example even the visible spectrum can be divided into many more than three bands; three are used because this mimics our visual system. Most of the satellites currently in orbit collect image information in tow to seven spectral bands, typically one to three are in the visible spectrum, one or more are in the infrared region, and some have sensors the operate in the radar range. The newest satellites have sensors that collect image information in 30 or more bands.

As the amount of data that needs to be transmitted, stored, and processed increases, the importance of topics such as compression becomes more and more apparent.

1.3.Computer imaging

Computer imaging is a fascinating and exciting area to be involved in today. The advent of the information superhighway, with its ease of use via the World Wide Web, combined with the advances in computer power have brought the world into our offices and into our homes. One of the most interesting aspects of this information revolution is the ability to send and receive complex data that transcend ordinary written text.

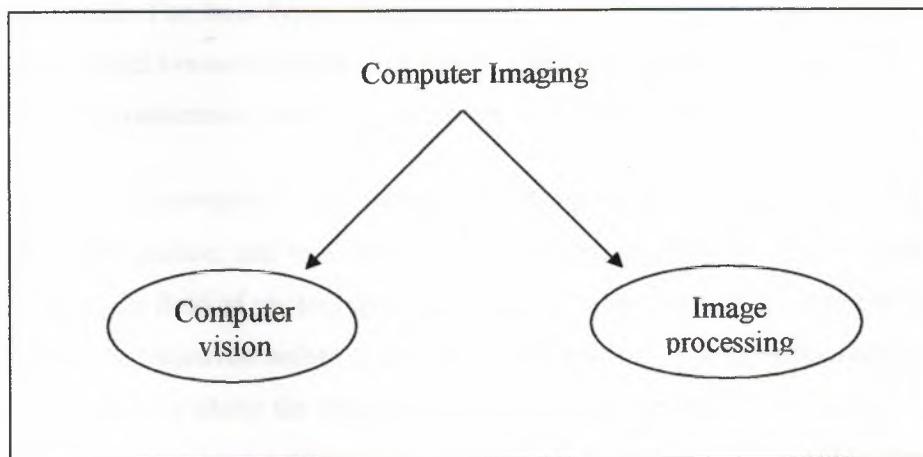


Figure 1.2 Computer Imaging

1.3.1. Computer Vision

Computer vision is the computer imaging where the application does not involve human being in the visual loop. In other words, the image are examined and acted upon by a computer. One of the major topics within the field of computer vision is image analysis.

Image analysis involves the examination of the image data to facilitate solving a vision problem. The image analysis process involves two other topics feature extraction and pattern classification. Feature extraction is the process of acquiring higher level image information, such as shape or color information, and pattern classification is the act of taking this higher-level information and identifying objects within the image.

Computer vision systems are used in many and various types of environments from manufacturing plants to hospital surgical suites to the surface of mars.

1.3.2. Image processing

Image processing is the computer imaging where the application involves a human being in the visual loop. In other word, the image is to be examined and acted upon by people. For these types of applications, we require some understanding of how the human visual system operates. The major topics within the files of image processing include image restoration, image enhancement, and image compression.

Image restoration is the process of taking an image with some known, or estimated, degradation, and restoring it to its original appearance. Image restoration is often used in the field of photography or publishing where an image somehow degraded but needs to be improved before it can be printed. For this type of application, we need to know something about the degradation process in order to develop a model for the distortion. When we have a model for the degradation process, we can apply the inverse process to the image to restore it to its original form.

Image enhancement involves taking an image and improving it visually, typically by tanking advantage of the human visual system's response. One of the simplest and often most dramatic enhancement techniques is to simple stretch the

contrast of an image. Enhancement methods tend to be problem specific. For example a method that is used to enhance satellite image may not be suitable for enhancing medical images. Although enhancement and restoration are similar in aim, to make an image look better, they differ in how they approach the problem. Restoration methods attempt to model the distortion to the image and reverse this degradation, whereas enhancement method uses knowledge of the human visual system's response to improve an image visually.

Image compression involves reducing the typically massive amount of data needed to represent an image. This is done by eliminating data that are visually unnecessary and by taking advantages of the redundancy that is inherent in most images. Although image compression is used in computer vision systems.

Image segmentation is important in many computer vision and image processing applications. The goal of image segmentation is to find region that represent objects or meaningful parts of objects. Division of the image into region corresponding to objects of interest is necessary before any processing can be done at a level higher than that of the pixel. Identifying real objects, pseudo-objects, and shadows or actually finding any thing of interest within the image requires some form of segmentation.

Image segmentation methods look for objects that either have some measure of homogeneity within them selves or have some measure of contrast with the objects on their border. Most image segmentations algorithms are modifications, extensions, or combinations of these two basic concepts. The homogeneity and contrast measures can include features such as gray level, color, and texture. After we have performed some preliminary segmentation, we may incorporate higher-level object properties, such as perimeter and shape into the segmentation process.

Also we have to consider some problems associated with image segmentation. The major problems are result of noise in the image and digitization of the continuous image. Noise is typically caused by the camera, the lenses, the lighting, or the signals path and can be reduced by the use of the preprocessing methods previously discussed.

Spatial digitization can cause problems regarding connectivity of objects. These problems can be resolved with careful connectivity definition and heuristics applicable to the specific domain.

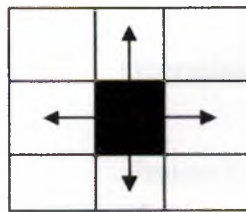
Connectivity refers to the way in which we define an object. After we have segmented an image. We must define which of the surrounding pixels are considered to be neighboring pixels. A pixel has eight possible neighbors: Two horizontal neighbors.

Two vertical neighbors, And four diagonal neighbors.

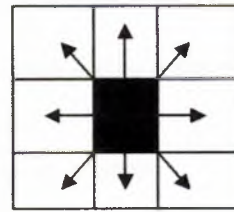
We can define connectivity in three different ways:

1. Four-connectivity.
2. Eight-connectivity.
3. Six-connectivity.

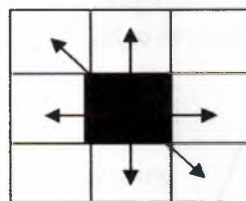
Figure 1.3 illustrates these three definitions.



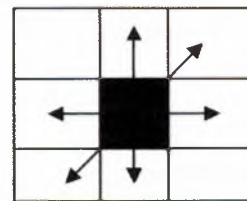
a. Four-connectivity



b. Eight-connectivity



c. Six-connectivity NW/SE



d. Six-connectivity NE/SW.

Figure 1.3. Connectivity

We can divide image segmentation techniques into three main categories:

1. Region growing and shrinking.
2. Clustering methods.
3. Boundary detection.

1.4. Computer imaging system

Computer imaging systems are comprised of two primary component types, hardware and software. The hardware components can be divided into the image acquisition subsystem, the computer itself, and the display devices.

The software allows us to manipulate the image and perform any desired processing on the image data. Additionally, we may also use software to control the image acquisition and storage process.

(Image Acquisition like: Camera, Scanner, Video player...).

(Image Display like: Monitor, Printer, Film, Video Recorder...).

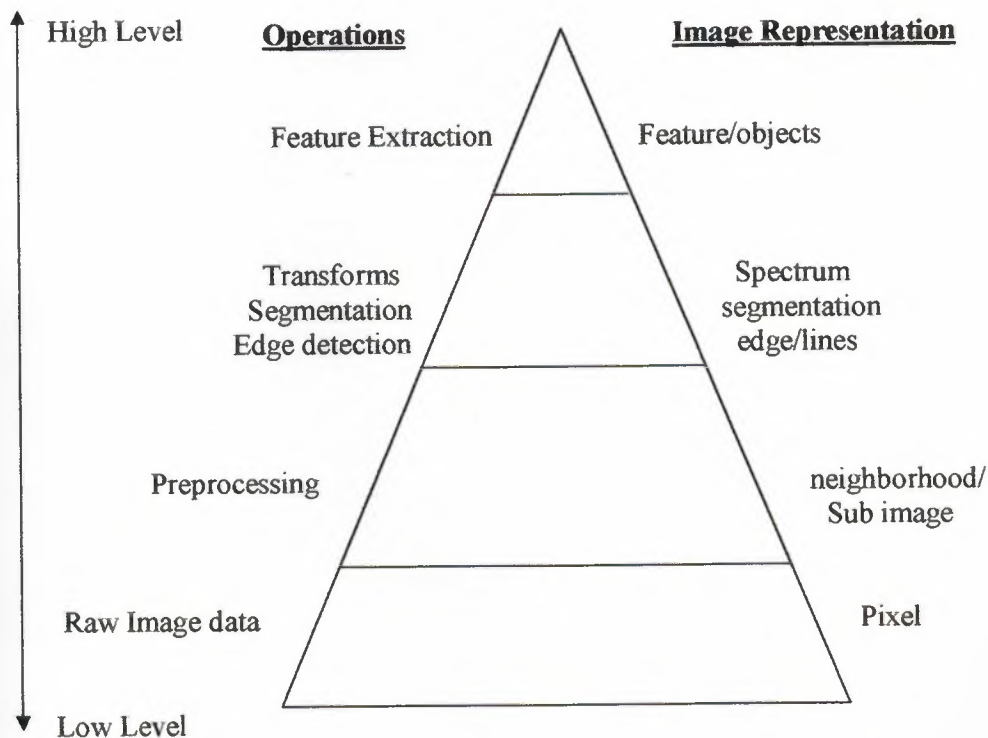


Figure 1.4. The Hierarchical Image Pyramid

1.5. Summary

This chapter tries to give a small view about the image processing and its importance in our life beginning with the computer imaging which involve computer vision and image processing. It explained briefly the both parts of computer imaging, Computer vision and Image processing and there differences.

This chapter also tries to show the mechanism of the human visual system, its main components, and its image types. Next chapter will focus in the techniques used to detect the objects' edges.

CHAPTER 2: EDGE DETECTION OPERATORS

2.1. Overview

Objects within images can simply be recognized by their crude outlines or edges. Thus, edge detection is considered as a low-level process in image recognition as it is applied prior to other image processing techniques.

Edge detection operators are based on the idea that edge information in any image is found by looking at the relationship a pixel has with its neighbors. If a pixel's gray-level value is similar to those around it, there is probably not an edge at that point. However, if a pixel has neighbors with widely varying gray levels, it may present an edge point. In other words, an edge is defined by a discontinuity in gray-level values. Ideally, an edge separates two distinct objects. In practice, apparent edges are caused by changes in color or texture or by the specific lighting conditions present during the image acquisition process.

This chapter presents some of the most commonly used operators in edge detection, such as Roberts operator, Prewitt operator, Sobel operator, and Laplacian of the Gaussian operator. It gives a brief description about each operator, how they work, and guidelines for use.

2.2. Edge Detectors

Edges are places in the image with strong intensity contrast. Since edges often occur at image locations representing object boundaries, edge detection is extensively used in image segmentation when we want to divide the image into areas corresponding to different objects. Representing an image by its edges has the further advantage that the amount of data is reduced significantly while retaining most of the image information.

Since edges consist of mainly high frequencies, we can, in theory, detect edges by applying a high pass frequency filter in the Fourier domain or by convolving the image with an appropriate kernel in the spatial domain. In practice, edge detection is

performed in the spatial domain, because it is computationally less expensive and often yields better results.

Since edges correspond to strong illumination gradients, we can highlight them by calculating the derivatives of the image. This is illustrated for the one-dimensional case in Figure 2.1.

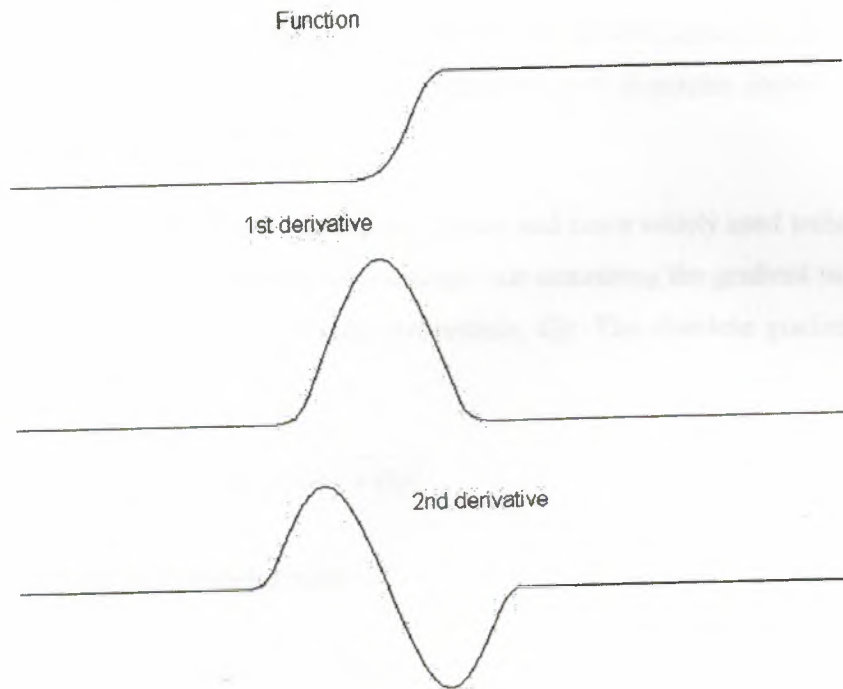


Figure 2.1 1st and 2nd derivative of an edge illustrated in one dimension.

We can see that the position of the edge can be estimated with the maximum of the 1st derivative or with the zero-crossing of the 2nd derivative. Therefore we want to find a technique to calculate the derivative of a two-dimensional image. For a discrete one-dimensional function $f(i)$, the first derivative can be approximated by

$$\frac{df(i)}{d(i)} = f(i+1) - f(i) \quad (2.1)$$

Calculating this formula is equivalent to convolving the function with $[-1 \ 1]$. Similarly the 2nd derivative can be estimated by convolving $f(i)$ with $[1 \ -2 \ 1]$.

Different edge detection kernels which are based on the above formula enable us to calculate either the 1st or the 2nd derivative of a two-dimensional image. There are two common approaches to estimate the 1st derivative in a two-dimensional image, Prewitt compass edge detection and *gradient edge detection*.

Prewitt compass edge detection involves convolving the image with set of (usually 8) kernels, each of which is sensitive to a different edge orientation. The kernel producing the maximum response at a pixel location determines the edge magnitude and orientation. Different sets of kernels might be used: examples include Prewitt, Sobel, Kirsch and Robinson kernels.

Gradient edge detection is the second and more widely used technique. Here, the image is convolved with only two kernels, one estimating the gradient in the x -direction, G_x , the other the gradient in the y -direction, G_y . The absolute gradient magnitude is then given by

$$|G| = \sqrt{G_x^2 + G_y^2} \quad (2.2)$$

and is often approximated with

$$|G| = |G_x| + |G_y| \quad (2.3)$$

In many implementations, the gradient magnitude is the only output of a gradient edge detector, however the edge orientation might be calculated with the most common kernels used for the gradient edge detector are the Sobel, Roberts Cross and Prewitt operators.

$$\theta = \arctan(G_y / G_x) - 3\pi / 4 \quad (2.4)$$

After having calculated the magnitude of the 1st derivative, we now have to identify those pixels corresponding to an edge. The easiest way is to threshold the gradient image, assuming that all pixels having a local gradient above the threshold must represent an edge. An alternative technique is to look for local maxima in the gradient image, thus producing one pixel wide edges. A more sophisticated technique is

used by the Canny edge detector. It first applies a gradient edge detector to the image and then finds the edge pixels using *non-maximal suppression* and *hysteretic tracking*.

An operator based on the 2nd derivative of an image is the Marr edge detector, also known as *zero crossing detector*. Here, the 2nd derivative is calculated using a Laplacian of Gaussian (LoG) filter. The Laplacian has the advantage that it is an isotropic measure of the 2nd derivative of an image, *i.e.* the edge magnitude is obtained independently from the edge orientation by convolving the image with only one kernel. The edge positions are then given by the zero-crossings in the LoG image. The scale of the edges which are to be detected can be controlled by changing the variance of the Gaussian.

A general problem for edge detection is its sensitivity to noise, the reason being that calculating the derivative in the spatial domain corresponds to accentuating high frequencies and hence magnifying noise.

2.2.1. Laplacian of Gaussian

- **Common Names**

Laplacian of Gaussian, LoG, and Marr Filter

- **Brief Description**

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection (see zero crossing edge detectors).

The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single gray level image as input and produces another gray level image as output.

- **How It Works**

The Laplacian $L(x,y)$ of an image with pixel intensity values $I(x,y)$ is given by:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \quad (2.5)$$

This can be calculated using a convolution filter. Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian.

Three commonly used small kernels are shown in Figure 2.2.

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

-1	2	-1
2	-4	2
-1	2	-1

Figure 2.2 Three commonly used discrete approximations to the Laplacian filter. (Note, we have defined the Laplacian using a negative peak because this is more common; however, it is equally valid to use the opposite sign convention.)

Using one of these kernels, the Laplacian can be calculated using standard convolution methods.

Because these kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often Gaussian smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

In fact, since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter first of all, and then convolve this hybrid filter with the image to achieve the required result. Doing things this way has two advantages:

- Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations.
- The LoG ('Laplacian of Gaussian') kernel can be precalculated in advance so only one convolution needs to be performed at run-time on the image.

The 2-D LoG function centered on zero and with Gaussian standard deviation σ has the form:

$$LoG(x, y) = -\frac{1}{\pi\sigma^4} \left[1 - \frac{x^2 + y^2}{2\sigma^2} \right] e^{-\frac{x^2 + y^2}{2\sigma^2}} \quad (2.6)$$

and is shown in Figure 2.3.

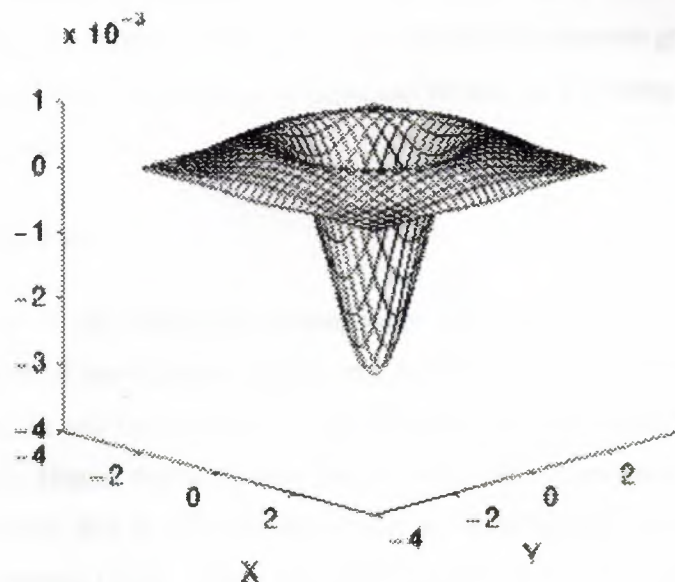


Figure 2.3 The 2-D Laplacian of Gaussian (LoG) function. The x and y axes are marked in standard deviations (σ).

A discrete kernel that approximates this function (for a Gaussian $\sigma = 1.4$) is shown in Figure 2.4.

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Figure 2.4 Discrete approximation to LoG function with Gaussian $\sigma = 1.4$

Note that as the Gaussian is made increasingly narrow, the LoG kernel becomes the same as the simple Laplacian kernels shown in Figure 2.1. This is because smoothing with a very narrow Gaussian ($\sigma < 0.5$ pixels) on a discrete grid has no effect. Hence on a discrete grid, the simple Laplacian can be seen as a limiting case of the LoG for narrow Gaussians.

- **Guidelines for Use**

The behavior of the LoG zero crossing edge detector is largely governed by the standard deviation of the Gaussian used in the LoG filter. The higher this value is set, the smaller features will be smoothed out of existence, and hence fewer zero crossings will be produced. Hence, this parameter can be set to remove unwanted detail or noise as desired. The idea that at different smoothing levels different sized features become prominent is referred to as 'scale'. The LoG operator calculates the second spatial derivative of an image. This means that in areas where the image has a constant intensity (*i.e.* where the intensity gradient is zero), the LoG response will be zero. In the vicinity of a change in intensity, however, the LoG response will be positive on the darker side, and negative on the lighter side. This means that at a reasonably sharp edge between two regions of uniform but different intensities, the LoG response will be:

- Zero at a long distance from the edge,
- Positive just to one side of the edge,
- Negative just to the other side of the edge,
- Zero at some point in between, on the edge itself.

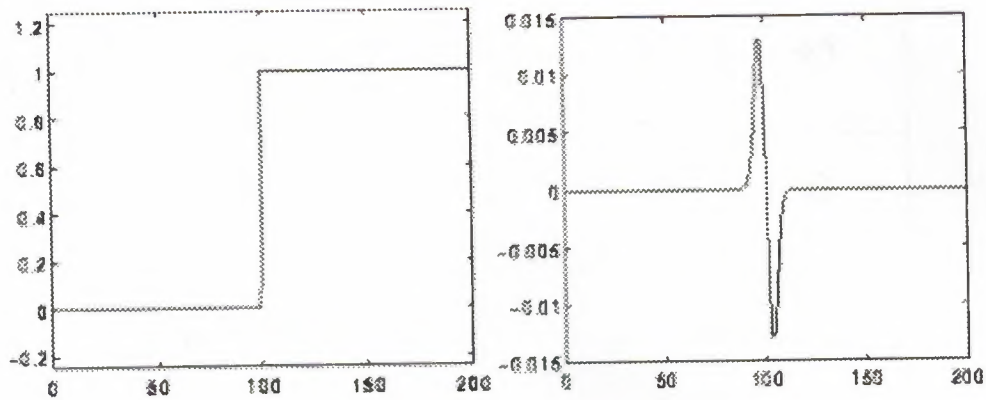


Figure 2.5 Response of 1-D LoG filter to a step edge. The left hand graph shows a 1-D image, 200 pixels long, containing a step edge. The right hand graph shows the response of a 1-D LoG filter with Gaussian $\sigma = 3$ pixels.

By itself, the effect of the filter is to highlight edges in an image.

2.2.2. Roberts Cross Edge Detector

- **Common Names:** Roberts Cross
- **Brief Description**

The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. It thus highlights regions of high spatial frequency, which often correspond to edges. In its most common usage, the input to the operator is a grayscale image, as is the output. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point.

- **How It Works**

In theory, the operator consists of a pair of 2×2 convolution kernels as shown in Figure 2.6. One kernel is simply the other rotated by 90° . This is very similar to the Sobel operator.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

Figure 2.6 Roberts Cross convolution kernels

These kernels are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Gx and Gy). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2} \quad (2.7)$$

Although typically, an approximate magnitude is computed using:

$$|G| = |Gx| + |Gy| \quad (2.8)$$

This is much faster to compute.

The angle of orientation of the edge giving rise to the spatial gradient (relative to the pixel grid orientation) is given by:

$$\theta = \arctan(Gy / Gx) - 3\pi / 4 \quad (2.9)$$

In this case, orientation 0 is taken to mean that the direction of maximum contrast from black to white runs from left to right on the image, and other angles are measured clockwise from this.

Often, the absolute magnitude is the only output the user sees --- the two components of the gradient are conveniently computed and added in a single pass over the input image using the pseudo-convolution operator shown in the figure below

P₁	P₂
P₃	P₄

Figure 2.7 Pseudo-convolution kernels used to quickly compute approximate gradient magnitude

Using this kernel the approximate magnitude is given by:

$$|G| = |P_1 - P_4| + |P_2 - P_3| \quad (2.10)$$

- **Guidelines for Use**

The main reason for using the Roberts Cross operator is that it is very quick to compute. Only four input pixels need to be examined to determine the value of each output pixel, and only subtractions and additions are used in the calculation. In addition there are no parameters to set. Its main disadvantages are that since it uses such a small kernel, it is very sensitive to noise. It also produces very weak responses to genuine edges unless they are very sharp. The Sobel operator performs much better in this respect.

2.2.3. Sobel Edge Detector

- **Common Names:** Sobel, also related is Prewitt Gradient Edge Detector.
- **Brief Description**

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

- **How It Works**

In theory at least, the operator consists of a pair of 3×3 convolution kernels as shown in the following figure. One kernel is simply the other rotated by 90°. This is very similar to the Roberts Cross operator.

- 1	0	+ 1
- 2	0	+ 2
- 1	0	+ 1

+ 1	+ 2	+ 1
0	0	0
- 1	- 2	- 1

Figure 2.8 Sobel convolution kernels

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these G_x and G_y). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{G_x^2 + G_y^2} \quad (2.11)$$

Typically, an approximate magnitude is computed using:

$$|G| = |Gx| + |Gy| \quad (2.12)$$

Which is much faster to compute the angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by:

$$\theta = \arctan(Gy / Gx) \quad (2.13)$$

In this case, orientation 0 is taken to mean that the direction of maximum contrast from black to white runs from left to right on the image, and other angles are measured anti-clockwise from this.

Often, this absolute magnitude is the only output the user sees --- the two components of the gradient are conveniently computed and added in a single pass over the input image using the pseudo-convolution operator shown in the figure below.

P₁	P₂	P₃
P₄	P₅	P₆
P₇	P₈	P₉

Figure 2.9 Pseudo-convolution kernels used to quickly compute approximate gradient magnitude

Using this kernel the approximate magnitude is given by:

$$|G| = |(P_1 + 2 \times P_2 + P_3) - (P_7 + 2 \times P_8 + P_9)| + |(P_3 + 2 \times P_6 + P_9) - (P_1 + 2 \times P_4 + P_7)| \quad (2.14)$$

- ***Guidelines for Use***

The Sobel operator is slower to compute than the Roberts Cross operator, but its larger convolution kernel smooths the input image to a greater extent and so makes the operator less sensitive to noise. The operator also generally produces considerably higher output values for similar edges, compared with the Roberts Cross.

As with the Roberts Cross operator, output values from the operator can easily overflow the maximum allowed pixel value for image types that only support smallish integer pixel values (e.g. 8-bit integer images). When this happens the standard practice is to simply set overflowing output pixels to the maximum allowed value. The problem can be avoided by using an image type that supports pixel values with a larger range.

Natural edges in images often lead to lines in the output image that are several pixels wide due to the smoothing effect of the Sobel operator. Some thinning may be desirable to counter this. Failing that, some sort of hysteretic ridge tracking could be used as in the canny operator.

2.2.4. Prewitt Edge Detector

- **Common Names:**

Prewitt, Compass edge detector.

- ***Brief Description***

The Prewitt is similar to the Sobel, but with different mask coefficients.

Prewitt Edge Detection is an alternative approach to the differential gradient edge detection. The operation usually outputs two images, one estimating the local edge gradient magnitude and one estimating the edge orientation of the input image.

- ***How It Works***

When using Prewitt edge detection the image is convolved with a set of (in general 8) convolution kernels, each of which is sensitive to edges in a different

orientation. For each pixel the local edge gradient *magnitude* is estimated with the maximum response of all 8 kernels at this pixel location:

$$|G| = \max(|G_i| : i = 1 \text{ to } n) \quad (2.15)$$

Where G_i is the response of the kernel i at the particular pixel position and n is the number of convolution kernels. The local edge *orientation* is estimated with the orientation of the kernel that yields the maximum response.

Various kernels can be used for this operation; for the following discussion we will use the Prewitt kernel. Two templates out of the set of 8 are shown in following Figure:

- 1	+ 1	+ 1
- 1	- 2	+ 1
- 1	+ 1	+ 1

+ 1	+ 1	+ 1
- 1	- 2	+ 1
- 1	- 1	+ 1

Figure 2.10 Prewitt edge detecting templates sensitive to edges at 0° and 45° .

The whole set of 8 kernels is produced by taking one of the kernels and rotating its coefficients circularly. Each of the resulting kernels is sensitive to an edge orientation ranging from 0° to 315° in steps of 45° , where 0° corresponds to a vertical edge.

The maximum response $|G|$ for each pixel is the value of the corresponding pixel in the output magnitude image. The values for the output orientation image lie between 1 and 8, depending on which of the 8 kernels produced the maximum response.

This edge detection method is also called *edge template matching*, because a set of edge templates is matched to the image, each representing an edge in a certain orientation. The edge magnitude and orientation of a pixel is then determined by the template that matches the local area of the pixel the best.

On the other hand, the Privet operator needs (here) 8 convolutions for each pixel, whereas the gradient operator needs only 2, one kernel being sensitive to edges in the vertical direction and one to the horizontal direction.

As already mentioned earlier, there are various kernels that can be used for Prewitt Edge Detection. The most common ones are shown in the following figure.

0

45

Sobel

-1	0	1
-2	0	2
-1	0	1

0	1	2
-1	0	1
-2	-1	0

Kirsch

-3	-3	5
-3	0	5
-3	-3	5

-3	5	5
-3	0	5
-3	-3	-3

Robinson

-1	0	1
-1	0	1
-1	0	1

0	1	1
-1	0	1
-1	-1	0

Figure 2.11 Some examples for the most common Prewitt edge detecting kernels, each example showing two kernels out of the set of eight.

For every template, the set of all eight kernels is obtained by shifting the coefficients of the kernel circularly.

The result for using different templates is similar; the main difference is the different scale in the magnitude image. The advantage of Sobel and Robinson kernels is that only 4 out of the 8 magnitude values must be calculated. Since each pair of kernels rotated about 180° opposite is symmetric, each of the remaining four values can be generated by inverting the result of the opposite kernel.

2.3. Summary

We can see that every operator has its own parameters or mask coefficients which use in edge detection. It also differs in the speed of computing which can be one of the main reasons for using the operator such as Roberts Cross operator. With many of these operators; noise in the image can create problems. That is why it is best to preprocess the image to eliminate, or at least minimize, the noise effects. To deal with noise effects, we must make tradeoffs between the sensitivity and the accuracy of an edge detector.

This chapter represented a comparison among the different types of edge detector. It mentioned its common names, brief description about it. How it works and guidelines for use. At the end it showed the common variants among them.

CHAPTER THREE: THE DEVELOPED SOFTWARE APPLICATION

3.1. Overview

This chapter presents the original work done by the author where common edge detector are integrated into a software program with user friendly GUI.

The following section has the list of the main program, which has been developed. It explains the program interface in details such as the operations of the buttons in the interface, also it contains the results that we achieved by running it using all the five applied edge detection operators.

The language used in developing this program was Delphi language. The main reason behind using it is that the Delphi is an object oriented, component based, visual, rapid development environment for event driven Windows applications, based on the Pascal language.

3.2. Software Specifications

After pressing the program icon the program interface will appear as in figure 3.2. The first button on the left is responsible about opening new images which can be used to apply one of the five operators the program have. After applying one of these operators, either you can save the modified image or you can easily use the rest button to bring back the original image then again apply new operator. Program has been modified to be able to deal with the images both with .jpg and .bmp extension.

The small square at the left bottom is showing the mask algorithm of the operator you choosed to apply on the image. The program contains five operators Sobel, Priwett, Laplace, Robert, and LoG.

Sobel operator is a square mask with size 3×3 which represented by the matrix $[-1 \ 0 \ 1; -2 \ 0 \ 2; -1 \ 0 \ 1]$. The advantages of Sobel operator that it has less sensitivity to noise and it has odd size which gives better edge position. The main disadvantage of it that it is less precise and slower than the other operators like Robert

Prewitt operator is another square mask with size 3×3 and has the values $[-1 \ 0 \ 1; -2 \ 0 \ 2; -1 \ 0 \ 1]$. It has the same advantages and disadvantages of Sobel operator. However, Prewitt operator can introduce a larger mask of size 5×5 and due to this it is slower but more robust and strong with noise.

Laplace operator is another square mask with size 3×3 and has the values $[-1 \ -1 \ -1; -1 \ 8 \ -1; -1 \ -1 \ -1]$.

Robert uses a 2×2 mask which is convoluted with the image. It has a good advantages as it has a very fast speed and more precise because of its small size. On the other hand, it has some problems because it is sensitive to noise and position of edge maybe lost because of even number.

Laplacian of the Gaussian LOG is a more popular operator that applied first to filter noise. The amount of noise reduction can be adjusted by changing σ standard deviation of Gaussian. Then a Laplacian edge detector is applied.

Following section explains the program interface in more details.

3.3. Program Flowchart

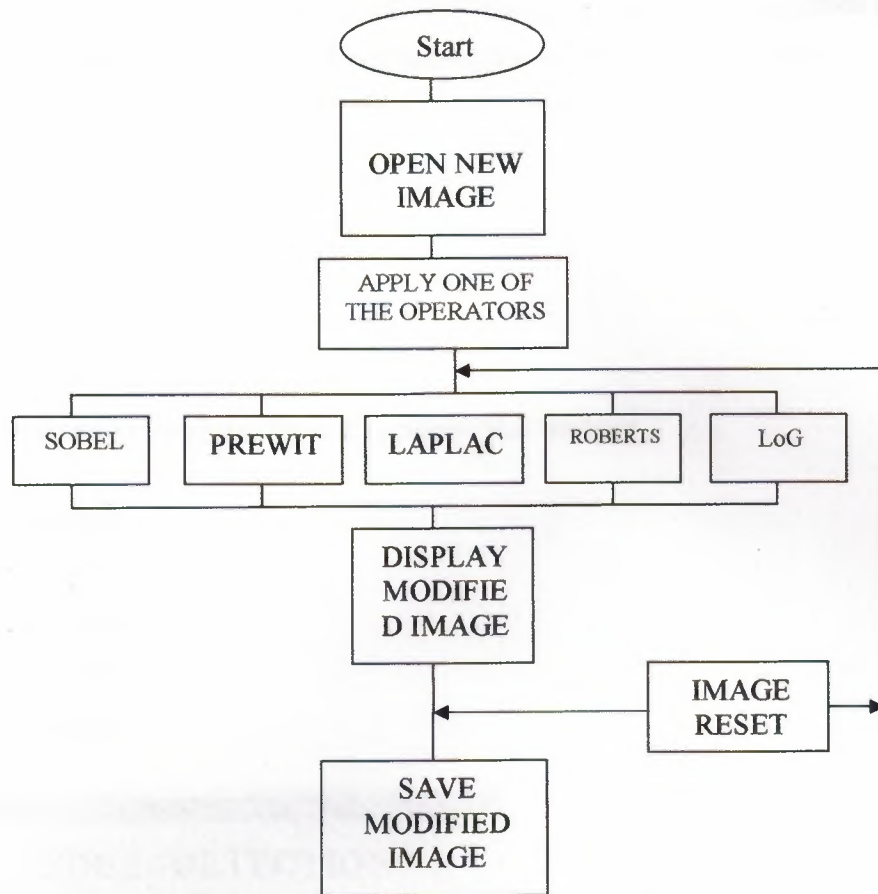


Figure 3.1 the program flowchart

After starting the program new image can be chosen then one of the five operators can be applied on the image the modified image then will be displayed in the place of the image itself. Then we will have two options, either save modified image or make reset to obtain the same image again and apply another operator on it to see the difference between the results.

3.4. The Program Interfaces

The following Figures Shows the interfaces of the application presented in this thesis. As we see in this figure we have buttons that divided into two groups :

The first one :

- Open image
- Save image
- Image reset

The second one for applying the five operators that we have:

1. Sobel operator
2. Prewitt operator
3. Laplace operator
4. Robert operator
5. LoG operator

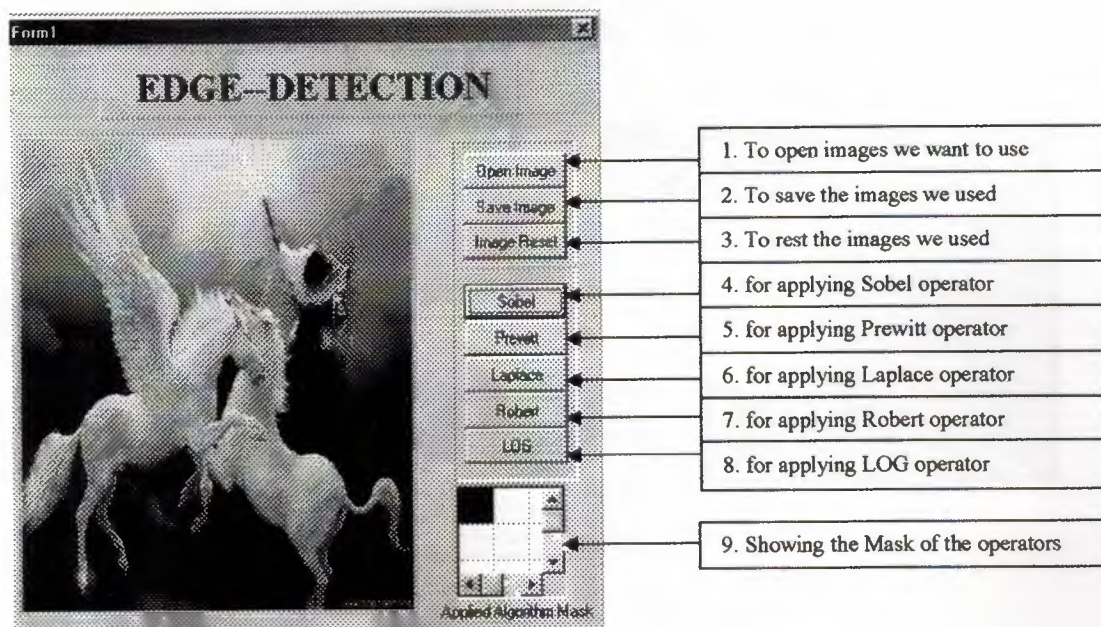


Figure 3.2. The main interface of the application

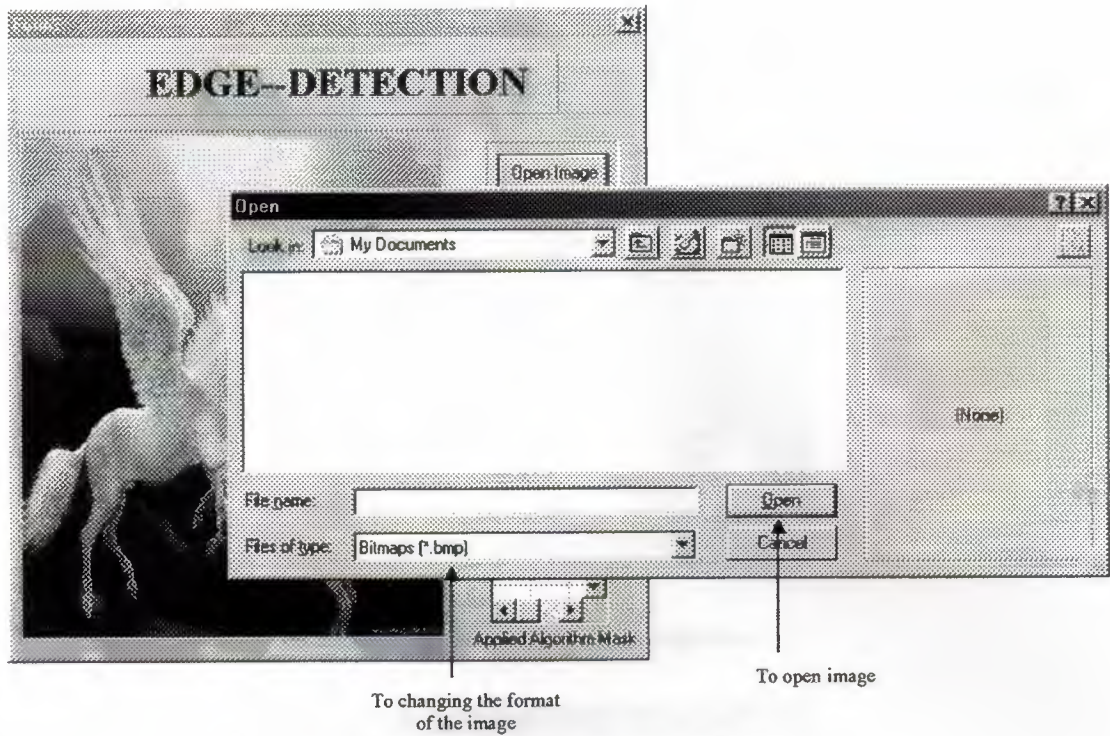


Figure 3.3. The function of OPEN IMAGE button

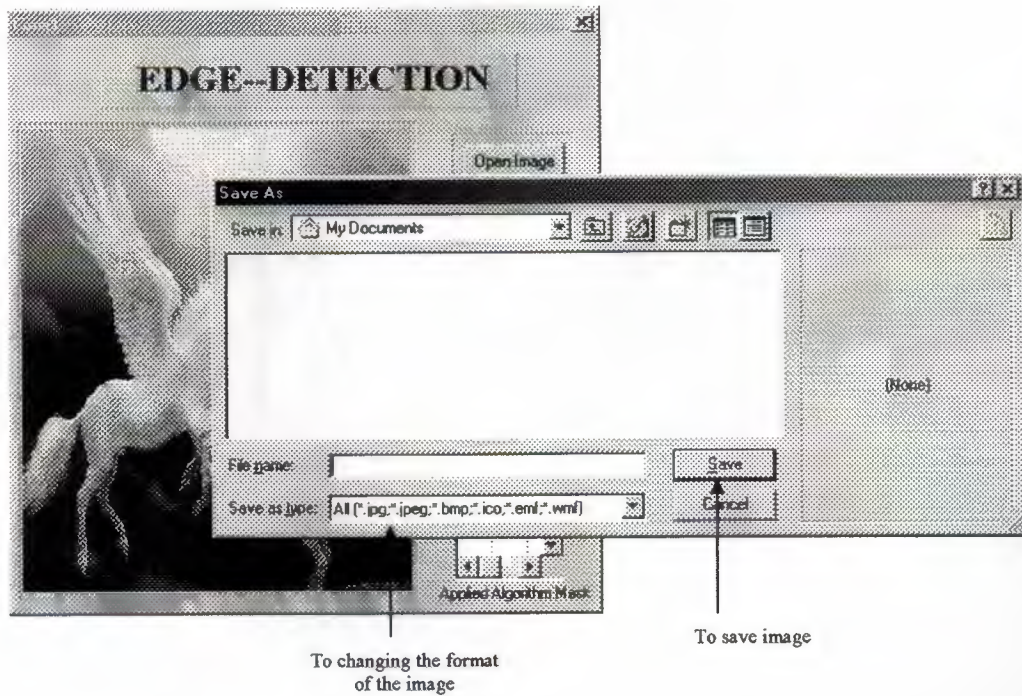
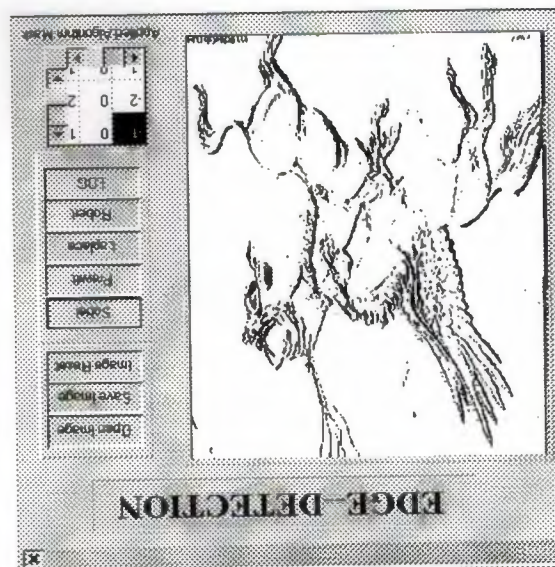


Figure 3.4. The function of SAVE IMAGE button

Figure 3.6. Applying Prewitt operator



Figure 3.5. Applying Sobel operator



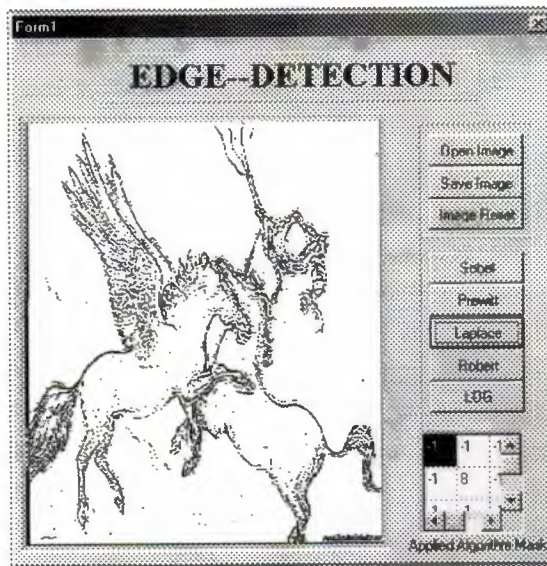


Figure 3.7. Applying Laplacian operator

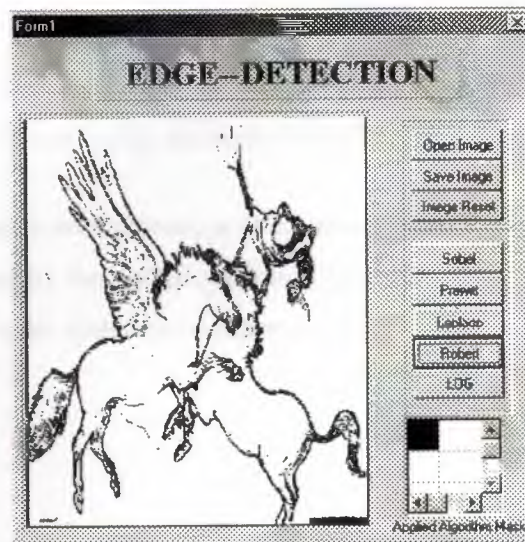


Figure 3.8. Applying Robert operator

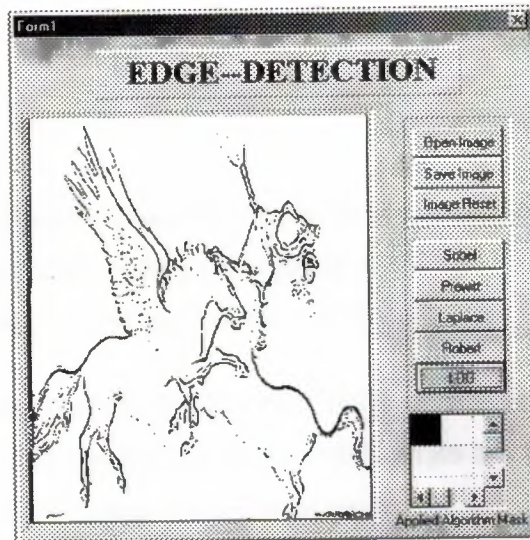


Figure 3.9. Applying LoG operator

3.5. Summary

This chapter presented the development of a software application that allows a user to apply edge detection using the most commonly edge detectors.

The next chapter will provide a comparison between these five edge detectors using images captured by the author. This will be followed by an analysis of detection results and the criteria for optimum edge detection.

CHAPTER 4 : RESULTS AND COMPARISON

4.1. Overview

This chapter presents practical implementation of the developed software on images, we used a grayscale image, we captured the two images then we applied on both of them each of the five detectors we used in the software.

The images were captured by the author and there will be a comparison between the five edge detectors by showing all the resulting images and analysing the detection results we had from each detectors for the same image.

The optimum edge detector will then be chosen according to the optimum edge detector criteria.

4.2. Image capturing environment

The images that we are using here were captured using a special image grabbing box. It is a box with 60 cm side which is painted black from inside with a hole at the upper side to settle the camera lens through it.

Also it has black base from inside which can be moved up and down by two holders in order to adjust the desired distance of the objects, which are going to be photographed. The maximum distance between the object which will be on the black plate and the camera lens is 60 cm and the minimum is 40 cm. See figure 4.1.

The other important factor here is the lighting inside the box. There was two sources of light which are fixed at the top of the box. It is 240 W 60 V fluorescent light which can be used by turning one of them or both of them at once according to desired lighting.

Figure 4.1 The Grabbing Box

4.3. Camera details

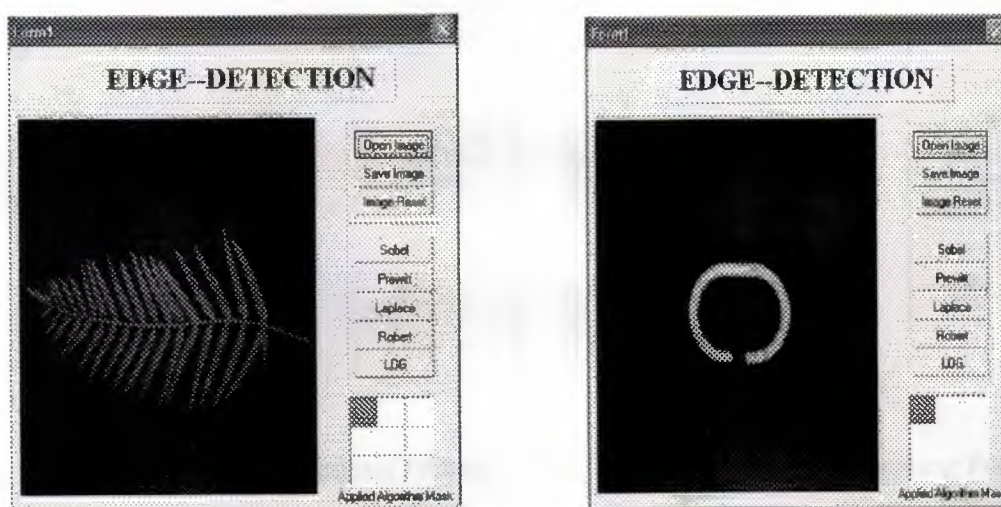
The used camera were a BENQ digital camera-1300 is equipped with the following features:

Total pixels	1.36M
Effective Pixels	1.3M
PC Camera function	Yes
Digital Camera Function	Yes
Image Resolution	1280 * 1024 Pixel, 1024 * 768 Pixel, 640 * 480 Pixel
Lens	Free focus, 50 mm equivalent
Focusing Range	Normal: 1.5m ~ Infinity , Macro: 30~50cm, Text: 11~13cm
Digital Zoom	No
Shutter Speed	No
Flash	Built-in
White Balance	Auto (Daylight, Overcast, Cloudy, Tungsten, Fluorescent)
Exposure	Programmed
Metering	Programmed
View Finder	True Image Optical

Burst Mode	Yes
Movie Mode	AVI format (up to 90sec, including audio)
Self-Timer	10 sec. Adjustable
File Format	Compressed JPEG
Connectivity	Mini USB
Power Source	2x AAA Alkaline battery
LCD Monitor	Status LCD
Weight	63 g (with battery)
Dimensions(W x H x D)	87 mm x 57mm x 26 mm
Battery charge	No
System Requirements	Win98/2000/ME/XP

4.4. Applying the developed software to the images

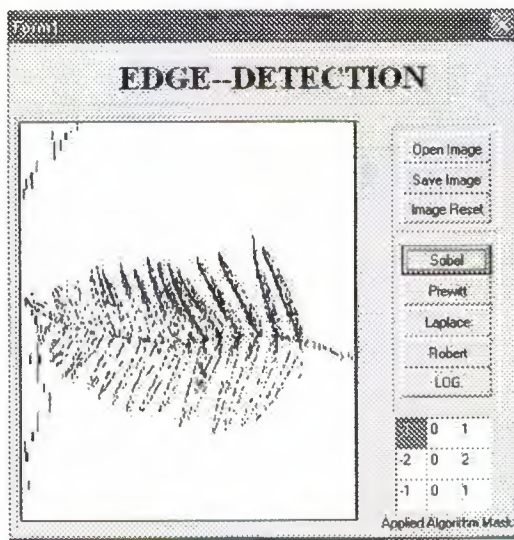
Here the author used two images took by the specified camera and he applied the five detectors on it in order to be able to have more fair comparison between all of the detectors. The objects he used were a leaf and ring. He chosed a simple two images with clear edges like in case of the ring and with many edges like the case of the leaf



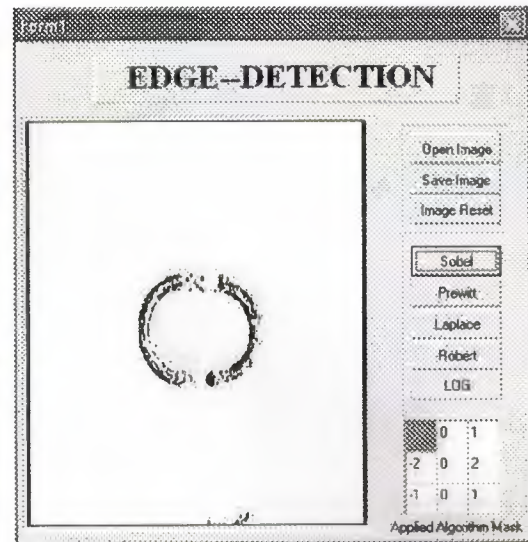
a) Original Leaf

b) Original Ring

Figure 4.2. The Original Images

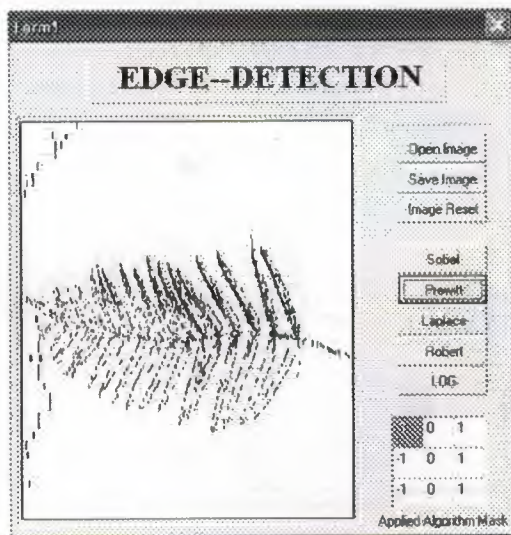


a) Leaf edges after applying Sobel operator

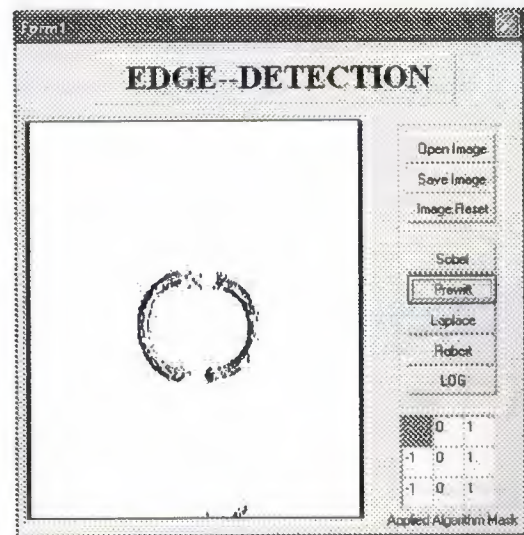


b) Ring edges after applying Sobel operator

Figure 4.3. Sobel operator

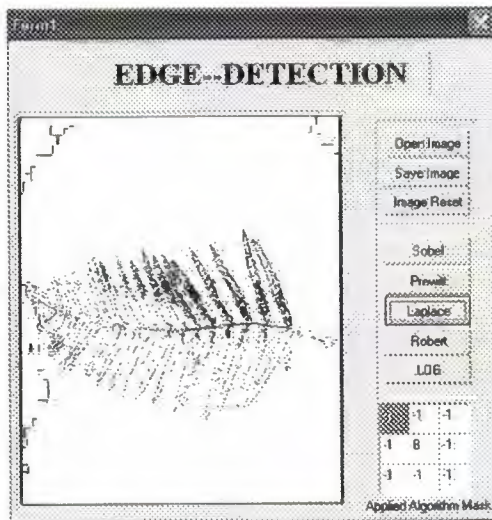


a) Leaf edges after applying Prewitt operator

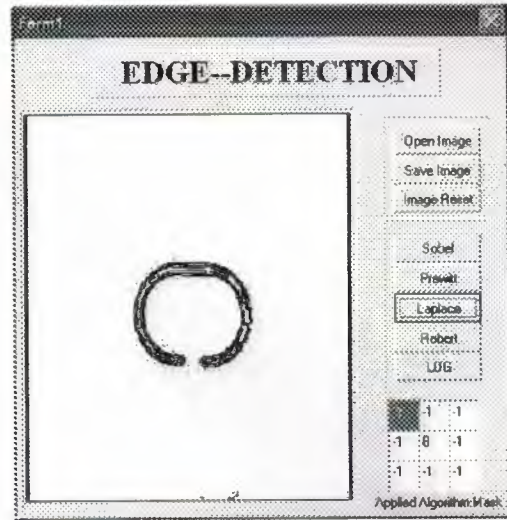


b) Ring edges after applying Prewitt operator

Figure 4.4. Prewitt Operator

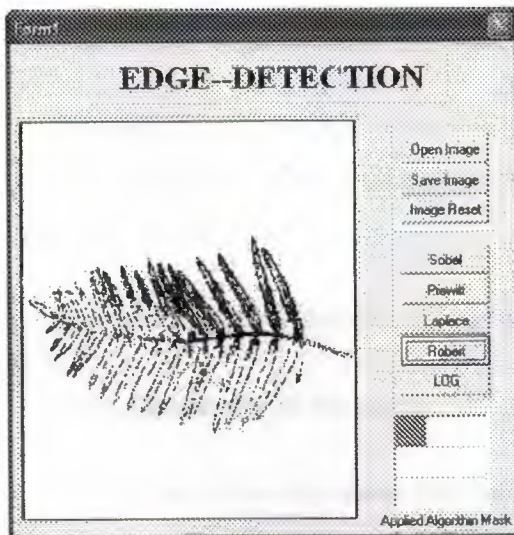


a) Leaf edges after applying Laplace operator

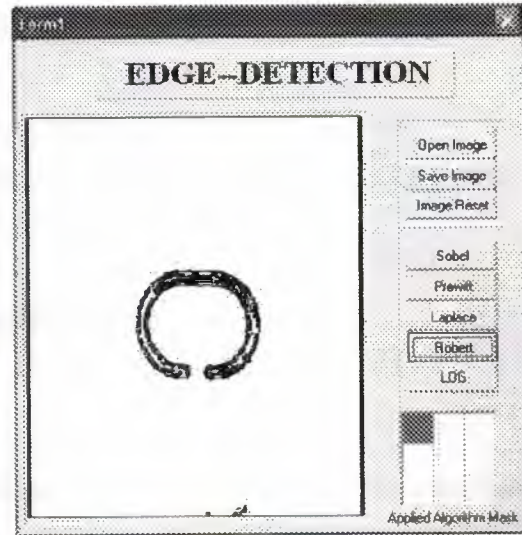


b) Ring edges after applying Laplace operator

Figure 4.5. Laplace Operator

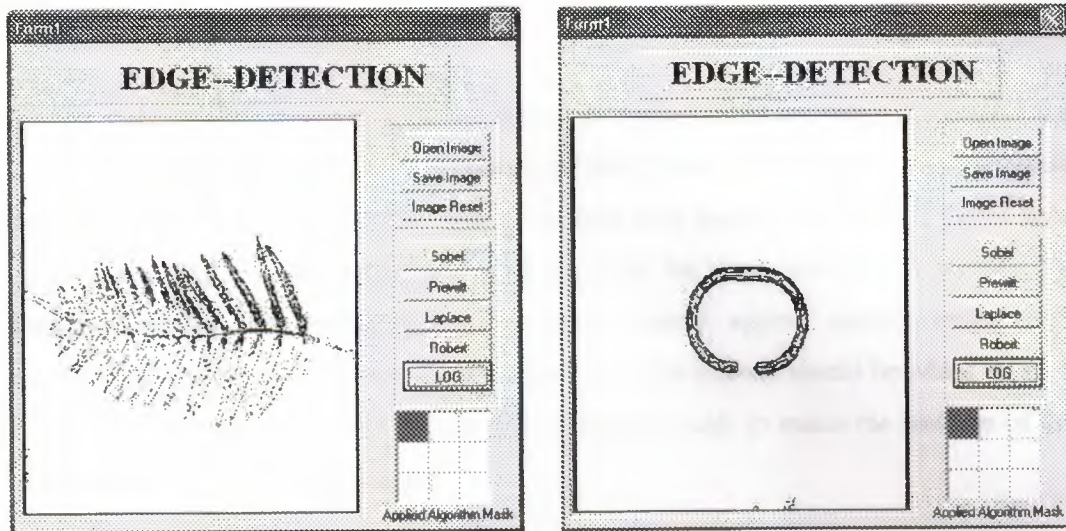


a) Leaf edges after applying Robert operator



b) Ring edges after applying Robert operator

Figure 4.6. Robert Operatore



a) Leaf edges after applying LoG operator

b) Ring edges after applying LoG operator

Figure 4.7. LoG Operatore

4.5. Optimum edge detector criteria

The aim of anaysis is to determine the optimum edge detector out of the five considered operators. The criteria for detecting the optimum detector is based on visuall inspection of the result images.

The optimum edege detector criteria includes:

4.5.1. Continuity of the edges

Continuity of the edge mean that there is no cut in some aera along the edge line. The comparison is considering the detector with less dicontinuity in the edges

4.5.2. Noise reduction

It is comparing which resulting image has less noise around the detected edges or not. In other words, which detector reduced most of the noise around the edges so the detected edges clear enough to identify the object.

4.5.3. Lighting

Since this system is based on the detection of edges within an image projected from a 3D object, the lighting of the 3D object is of the utmost importance for the quality of the resulting images. The default lighting consists of a horizontal and directional light "behind the user". Such lighting gives good results for most areas of the terrain. For instance, terrain comprising a lot of objects would appear mostly bright and, accordingly, the areas of the image corresponding to the objects would be white. In such a case, the user would have to change the lighting in order to make the features of the cliffs appear in the resulting images.

4.5.4. Contrast

Is the difference in brightness between two adjacent pixels. So suppose you want to compute a monotonically-increasing brightness curve that maximizes contrast for a particular monochrome image.

An optimum detector should care with images of varying contrasts.

4.5.5. convolution speed

the convolution speed depends mainly on the size of the mask you are using for detecting the edge. The smaller the mask size is the faster the speed of edge detection.

4.6 Comparison and Analysis

This section is making a comparison and analysis between the result of the ring and leaf images as detected by the five operators using the optimum edge detection criteria.

Continuity of the edges

For Sobel operator, the edges for both ring and leaf images have a lot of discontinuities on it. It is not appearing as a one line or curve most of the time.

For Prewitt operator, again the edges for both ring and leaf images are full with discontinuities.

For Laplace operator, the edges appears more smooth in ring image and there is almost no discontinuity on it. for leaf image it is full with discontinuities and you can hardly identify the image.

For Robert operator, the edges are the best here. No discontinuities were detected at all for ring image. However, for leaf image, the upper side edges of the image is good but there are some discontinuities on the lower side of the leaf.

For LoG operator, we have clear edges for the ring and no discontinuity at all. For leaf the detection was very poor and leaf edges had a lot of discontinuities on it.

Noise reduction

For Sobel operator, noise reduced inside leaf and ring but it seems noise threshold was high enough to start cutting the edges themselves.

For Prewitt operator, results were exactly same as for Sobel operator. Noise reduction was good inside the image but it cut some of the edges.

For Laplace operator, noise didn't reduce inside the ring. Which made the edges appear more thick.

For Robert operator, noise was too much inside the ring. on the other hand, it reduced in the lower side of the leaf more than its upper side.

For LoG operator, reduced some of the noise and kept the edges define better than the other operators.

Lighting

There were two light sources inside the grabbing box. If you look carefully at the images you can see them at the upper and lower side of both objects. The two lights did not have the same power which make the detection for edges more difficult in some operators to detect the objects especially for the lower leaf side.

For Sobel and Prewitt operators, the lighting was strong and it caused high brightness in the upper side of the ring and lower side of the leaf which let the operator fail to detect that area.

For Laplace and LoG operators, the problem was again in the lower side of the leaf which didn't appear nice because of the brightness from the lights.

For Robert operator, the lighting was distributed in a good manner that allowed the operator to detect the images.

Contrast

For Sobel operator, both images had very low contrast. Lines and edges are very thin and poorly appear.

For Prewitt operator, images result are too much close to that of sobel operator, low contrast and very light lines for the edges.

For Laplace operator, ring image had a high contrast and edges appear clearly but on the other hand the leaf image had extremely low contrast and edges were unclear.

For Robert operator, the two images had very high contrast but it appeared more in the ring image which had more thick edges and interior lines.

For LoG operator, the contrast was fair enough to show the ring clearly. But it wasn't good in the leaf image because it was again very low and led to a light edges.

Convolution speed

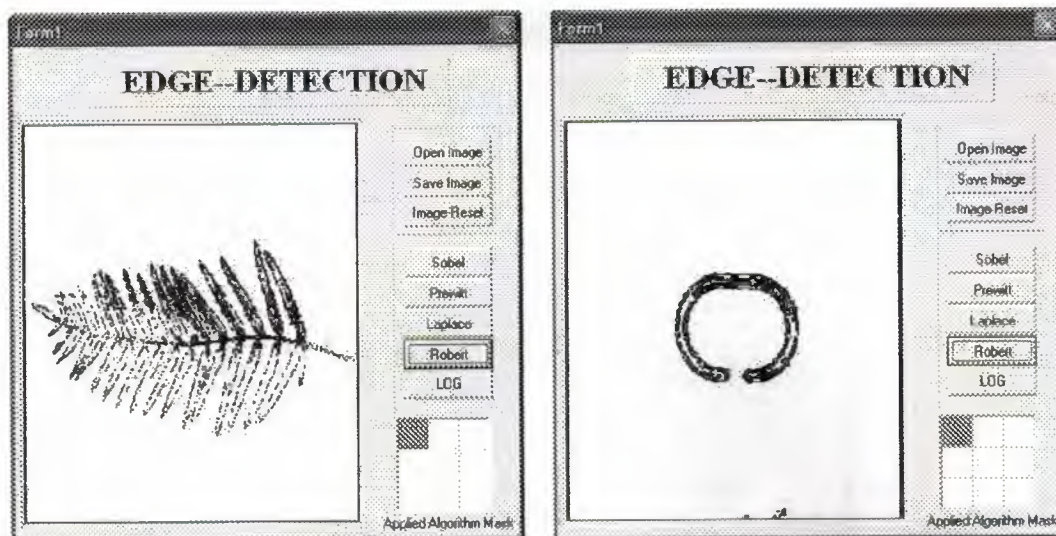
Sobel, Prewitt, and Laplace operators use a 3×3 mask which makes their detection speed slower than the other two operators which use a 2×2 mask.

Finally we can sum up all these criteria and the comparison between the five detectors in the following table.

Table 4.1. Comparison and analysis between the five operators

Criteria Operator	Continuity of the edges	Noise reduction	Lighting	Contrast	Convolution speed	Result
Sobel	x✓	x✓	x✓	x✓	x✓	xxxxx✓✓✓✓✓
Prewitt	xx	xx	x✓	x✓	x✓	xxxxxxx✓✓✓
Laplace	x✓	x✓	x✓	x✓	x✓	xxxxx✓✓✓✓✓
Robert	✓✓	x✓	✓✓	x✓	✓✓	xxxxx✓✓✓✓✓
LoG	x✓	✓✓	x✓	x✓	✓✓	xxx✓✓✓✓✓✓✓
Key: (x) means insufficient result. (✓) means sufficient result.						

From the table it is clear that Prewitt operator had the insufficient results, while the optimum edge detector is LoG operator.



a) Leaf edges after applying LoG operator

b) Ring edges after applying LoG operator

Figure 4.8. Optimum edge detector

4.7. Additional Example for Edge Detection

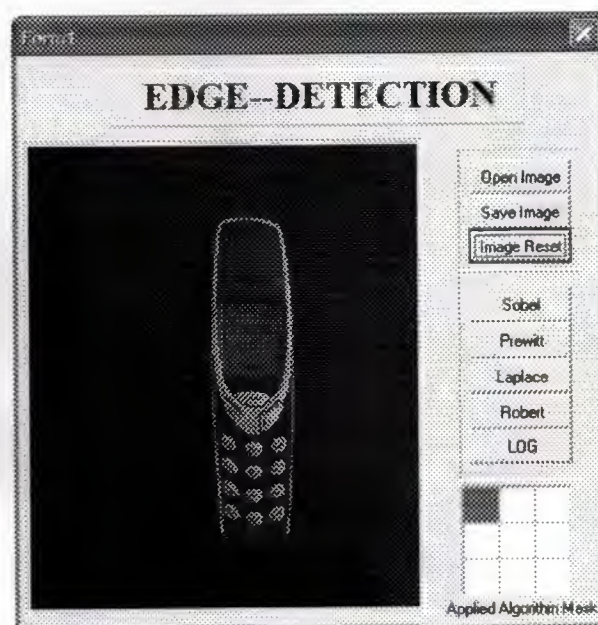


Figure 4.9. The Original Image



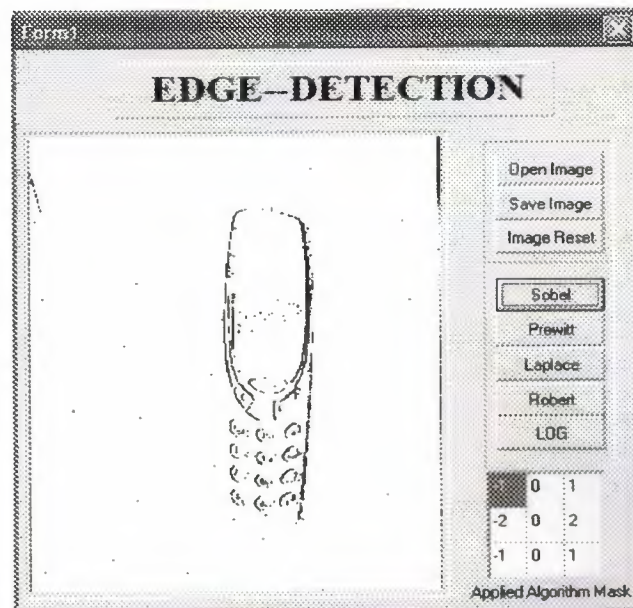


Figure 4.10. edges after applying Sobel operator

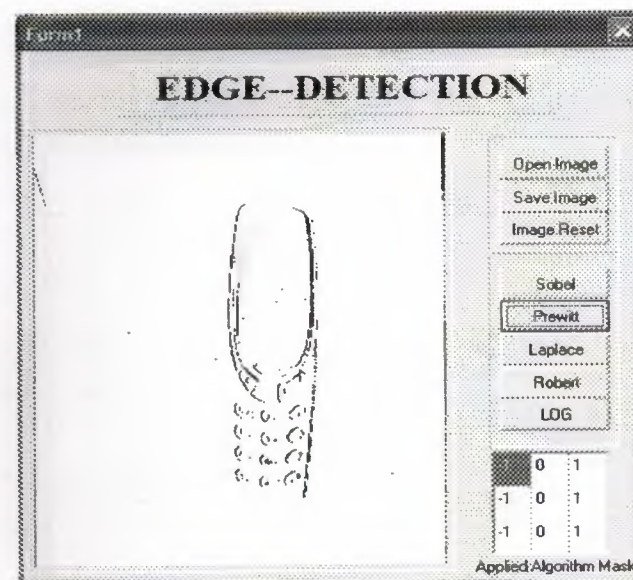


Figure 4.11. edges after applying Prewitt operator

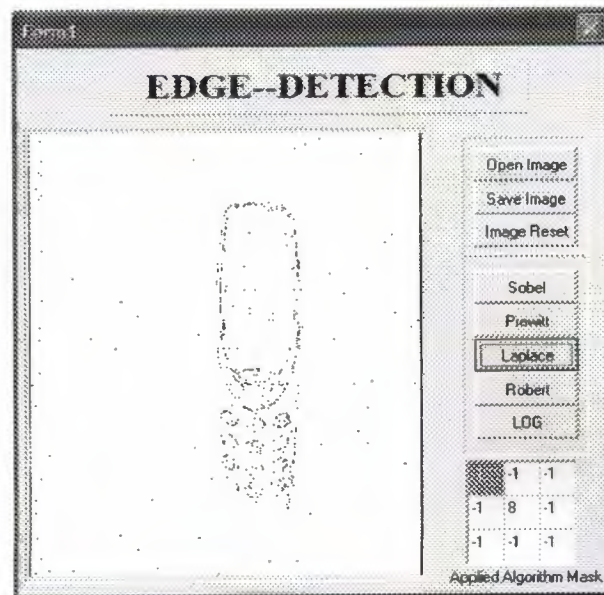


Figure 4.12. edges after applying Laplace operator

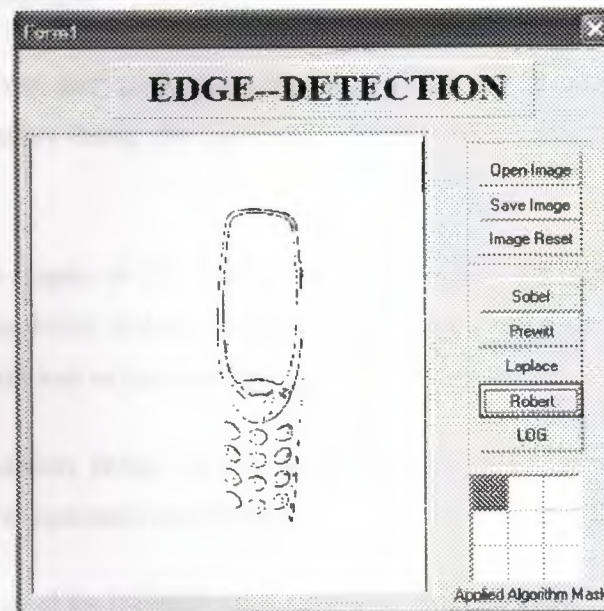


Figure 4.13. edges after applying Robert operator

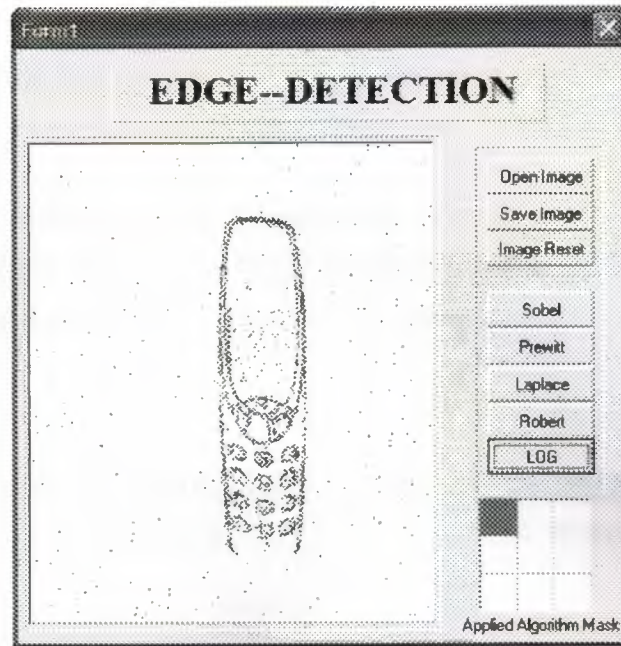


Figure 4.14. edges after applying LoG operator

4.7.1. Results Analysis

a comparison and analysis between the result of the mobile phone example detected by the five operators using the optimum edge detection criteria is made again as follows.

From the five graphs of the operators we can see that the Continuity of the edges was not enough in Sobel, Roberts and prewitt operators and poor in Laplace operator, while the best result was in the LoG operator's results.

For Noise reduction, Sobel, Laplace and LoG operators had the bad results while Roberts and prewitt operators were better.

Lighting was good for bringing good results for LoG operator.

About Contrast, the best results was in LoG operator too while the worst was in Laplace operator.

In Convolution speed, Sobel, Prewitt, and Laplace operators use a 3×3 mask which makes their detection speed slower than the other two operators which use a 2×2 mask.

So as a result we can summarize that the *LoG operator* was the best among the five operators we used in our program according to the optimum edge detector criteria we mentioned above.

As LoG was selected as the optimum edge detector, another example of three different mobile types like siemens, Nokia, and Sony Ericsson are shown in figure 4.15 below. The results are respectively shown in figure 4.16.

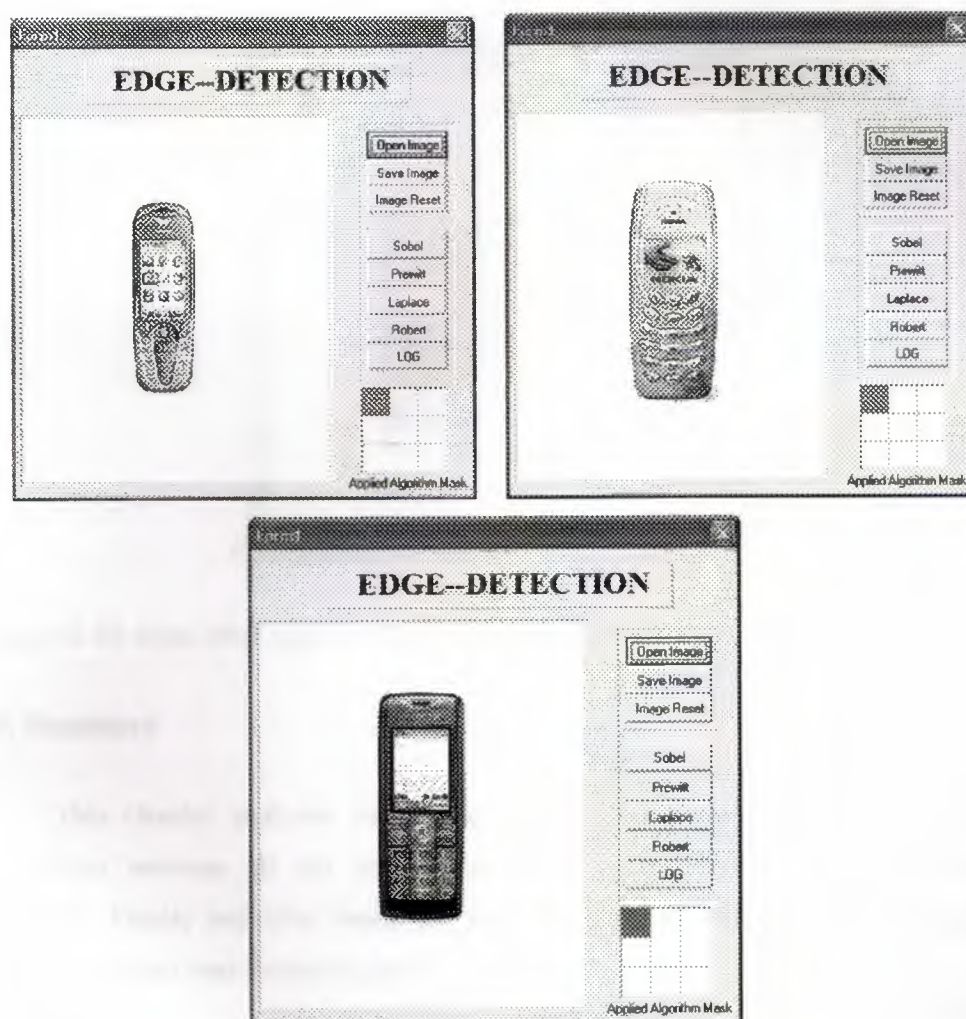


Figure 4.15 Three different mobile types (siemens, Nokia, and Sony Ericsson)

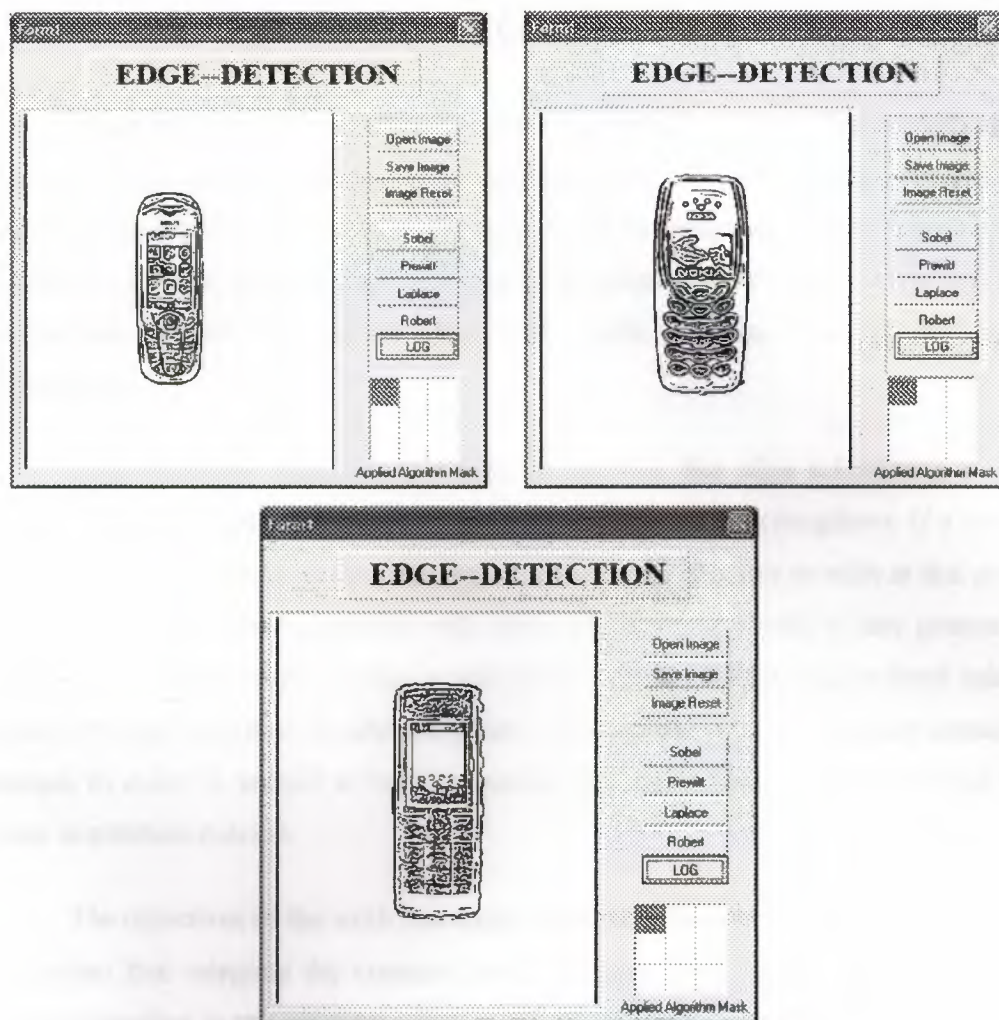


Figure 4.16 edges after applying LoG operator on three different mobile types

4.8. Summary

This chapter analyzed the results of the implemented program. It made a comparison between all the images according to the five criteria that has been developed. Finally and after check the result of the comparison it is found that the optimum operator was Robert operator.

The program aim wasn't to discover new thing or to introduce a new method of detecting edges. The program was simply trying to program some existing detection methods and put them in a way that you can make edge detection easily and compare the results to take the best of them using the developed criteria.

CONCLUSION

Edges are places in the image with strong intensity contrast. Since edges often occur at image locations representing object boundaries, edge detection is extensively used in image segmentation when we want to divide the image into areas corresponding to different objects. Representing an image by its edges has the further advantage that the amount of data is reduced significantly while retaining most of the image information.

Edge detection operators are based on the idea that edge information in any image is found by looking at the relationship a pixel has with its neighbors. If a pixel's gray-level value is similar to those around it, there is probably not an edge at that point. However, if a pixel has neighbours with widely varying gray levels, it may present an edge point. In other words, an edge is defined by a discontinuity in gray-level values. Ideally, an edge separates two distinct objects. In practice, apparent edges are caused by changes in color or texture or by the specific lighting conditions present during the image acquisition process.

The objectives of the work presented within this thesis are to develop a software application that integrates the common edge detectors. It collected the most famous detectors together in one program which makes it easy for the one who will work on it to compare the resulting images. It also provides a user-friendly GUI for further uses and to assist further work. It is possible to develop the program and make it more complicated by adding more masks such as Krisch mask and more functions on it such as ability to open images in format other than jpg, jpeg, and bmp, or add new buttons like print and help buttons.

This thesis compared & analyzed the five edge operators using own captured unique images, the leaf and the ring images which were captured by digital camera using grabbing box. The comparison criteria for optimum edge detection included edge continuity, contrast, lighting, noise reduction, and convolution speed. All detectors were compared and analyzed according to these criteria.

REFERENCES

- [1] E. Dougherty, C. Giardina, Matrix Structured Image Processing, Prentice Hall, Inc. 1987, New Jersey.
- [2] S. Banks, Signal Processing, Image Processing And Pattern Recognition, Prentice Hall, Inc. 1990, Hertfordshire.
- [3] A. Kulkarni, Computer Vision and Fuzzy-Neural Systems, Prentice Hall, Inc. 2001, New Jersey.
- [4] A. Khashman, Lecture Notes, NEU, Nicosia,
- [5] A. Rosenfeld, Digital Image Processing, 1982, Academic Press, Orlando
- [6] H. Andrews, Computer Techniques In Image Processing, Academic Press, 1970, New York
- [7] R. Gonzalez, Digital Image Processing, Addison Wesley, 1987, Reading.
- [8] R. Haralick, Edge and Region Analysis for Digital Image Data, Computer Vision and Graphics and Image Processing. Vol. 12, 1980.
- [9] R. Haralic, The Digital Edge. Processing of IEEE Computer Society Conference On Pattern Recognition And Image Processing, 1981, New York.
- [10] C. Chittineni, Edge and Line Detection In Multidimensional Noisy Imagery Data. IEEE Transactions on Geosciences and Remote Sensing. 1983, Vol. 21.
- [11] D. Marr, vision : A computer Vision Investigation Into Human Representation And Processing Of Visual Information, Freeman, New York
- [12] Ş. Bektaş, F. Mamedov, A. Khashman. Graduate studies: a complete reference, NEU. 2001, Nicosia.

APPENDIX A

The Program Source Code

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, ExtDlgs, Grids, jpeg;
type
  TForm1 = class(TForm)
    Sobel_btn: TBitBtn;
    Image1: TImage;
    Prewitt_btn: TBitBtn;
    Laplace_btn: TBitBtn;
    BitBtn4: TBitBtn;
    OpenPictureDialog1: TOpenPictureDialog;
    BitBtn5: TBitBtn;
    SavePictureDialog1: TSavePictureDialog;
    Robert_btn: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Bevel3: TBevel;
    StringGrid1: TStringGrid;
    Label3: TLabel;
    BitBtn7: TBitBtn;
    Bevel4: TBevel;
    LOG_btn: TBitBtn;
    procedure Sobel_btnClick(Sender: TObject);
    procedure Prewitt_btnClick(Sender: TObject);
    procedure Laplace_btnClick(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure Robert_btnClick(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure BitBtn7Click(Sender: TObject);
```



```

    procedure LOG_btnClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

type
    tMaskArray=array[0..8]of real;

var
    Form1: TForm1;
    mem_resident:tbitmap;

implementation

{$R *.dfm}

procedure setImage(var iniimage :tbitmap; srsImage:tbitmap);
begin

    iniimage:=tbitmap.create();
    iniimage.Height:=srsImage.height;
    iniimage.Width:=srsImage.Width;
    iniimage.Assign(srsImage);

    // ** CHANGE IMAGES TO 24 BIT PIXEL
    iniimage.HandleType:=bmdib;
    iniimage.PixelFormat:=pf8bit;
end;

function edgedetection(SRSImage:tbitmap):tbitmap;
var
    m:pbytearray;
    x,y:integer;
begin
    For Y:=0 to SRSImage.Height-3 do
        begin
            m:=SRSImage.ScanLine[Y+1];
            x:=3;

```

```

    while (x < SRSImage.Width -3) do
    begin
        if m[x]=0 then m[x]:=1;
        if (m[x] > 40) then
        begin
            m[x] :=0;
        end
        else
        begin
            m[x] :=255;
        end;
        x:=x+1;
    end;
end;
result:=srsimage
end;

function convolution(image:tbitmap; mask:tmaskarray;
offset,scale:integer):tbitmap;
var
    y,x,tno :integer;
    m,P,p1,p2 :PByteArray;
    temp_image:tbitmap;
begin
    setImage(temp_image,image);
    For Y:=0 to image.Height-3 do
    begin
        m :=temp_image.ScanLine[Y+1];
        p := image.ScanLine[y];
        p1:= image.ScanLine[Y+1];
        p2:= image.ScanLine[Y+2];
        x:=3;
        while (x < temp_image.Width -3) do
        begin
            tno:=round(( mask[0]*p[x-1] + mask[1]*p[x] + mask[2]*p[x+1]
+
            mask[3]*p1[x-1] + mask[4]*p1[x] + mask[5]*p1[x+1]
+

```

```

        mask[6]*p2[x-1] + mask[7]*p2[x] + mask[8]*p2[x+1]
+
        offset)/scale);
        if tno>255 then begin tno:=255; end ;
        if tno<0 then begin tno:=0 ; end ;
        m[x+2]:=tno;
        x:=x+1;
    End;
End;
result:=temp_image;
end;

```

```

procedure TForm1.Sobel_btnClick(Sender: TObject);
var
    ray:tmaskarray;
begin
    Image1.Picture.Bitmap.HandleType:=bmdib;
    Image1.Picture.Bitmap.PixelFormat:=pf8bit;

    ray[0]:=-1; ray[1]:= 0; ray[2]:= 1;
    ray[3]:=-2; ray[4]:= 0; ray[5]:= 2;
    ray[6]:=-1; ray[7]:= 0; ray[8]:= 1;

    Image1.Picture.Bitmap:=convolution(Image1.Picture.Bitmap,ray,0,1);
    Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
    Image1.Refresh;

    StringGrid1.Cells[0,0]:='-1';
    StringGrid1.Cells[1,0]:='0';
    StringGrid1.Cells[2,0]:='1';

    StringGrid1.Cells[0,1]:='-2';
    StringGrid1.Cells[1,1]:='0';
    StringGrid1.Cells[2,1]:='2';

    StringGrid1.Cells[0,2]:='-1';
    StringGrid1.Cells[1,2]:='0';

```



```

    StringGrid1.Cells[2,2]:='1';
end;

procedure TForm1.Prewitt_btnClick(Sender: TObject);
var
    ray:tmaskarray;
begin
    Image1.Picture.Bitmap.HandleType:=bmdib;
    Image1.Picture.Bitmap.PixelFormat:=pf8bit;

    ray[0]:=-1; ray[1]:= 0; ray[2]:= 1;

    ray[3]:=-1; ray[4]:= 0; ray[5]:= 1;

    ray[6]:=-1; ray[7]:= 0; ray[8]:= 1;

    Image1.Picture.Bitmap:=convolution(Image1.Picture.Bitmap,ray,0,1);
    Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
    Image1.Refresh;

    StringGrid1.Cells[0,0]:='-1';
    StringGrid1.Cells[1,0]:='0';
    StringGrid1.Cells[2,0]:='1';

    StringGrid1.Cells[0,1]:='-1';
    StringGrid1.Cells[1,1]:='0';
    StringGrid1.Cells[2,1]:='1';

    StringGrid1.Cells[0,2]:='-1';
    StringGrid1.Cells[1,2]:='0';
    StringGrid1.Cells[2,2]:='1';
end;

procedure TForm1.Laplace_btnClick(Sender: TObject);
var
    ray:tmaskarray;
begin
    Image1.Picture.Bitmap.HandleType:=bmdib;
    Image1.Picture.Bitmap.PixelFormat:=pf8bit;

```

```

ray[0]:=-1; ray[1]:=-1; ray[2]:=-1;

ray[3]:=-1; ray[4]:= 8; ray[5]:=-1;

ray[6]:=-1; ray[7]:=-1; ray[8]:=-1;

Image1.Picture.Bitmap:=convolution(Image1.Picture.Bitmap,ray,0,1);
Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
Image1.Refresh;

StringGrid1.Cells[0,0]:='-1';
StringGrid1.Cells[1,0]:='-1';
StringGrid1.Cells[2,0]:='-1';

StringGrid1.Cells[0,1]:='-1';
StringGrid1.Cells[1,1]:='8';
StringGrid1.Cells[2,1]:='-1';

StringGrid1.Cells[0,2]:='-1';
StringGrid1.Cells[1,2]:='-1';
StringGrid1.Cells[2,2]:='-1';
end;

procedure TForm1.BitBtn4Click(Sender: TObject);
var
    jpgimage:TJPEGImage;
    test:tbitmap;
    str:string;
begin
    if OpenPictureDialog1.Execute then
        begin
            str:=OpenPictureDialog1.FileName;
            image1.Picture.LoadFromFile(str);
            if (str[length(str)-2]<>'B') and (str[length(str)-2]<>'b') then
                begin
                    //(-- Converting from Jpg to BMP--
                    jpgimage:=tjpegimage.Create ;
                    jpgimage.Assign(image1.Picture);

```

```

        test:=tbitmap.Create ;
        test.Assign(jpgimage);
        jpgimage.Destroy;
        //-----}
        imagel.Picture.Bitmap.Assign(test);
        test.Destroy;
    end;
    setImage(mem_resident, imagel.Picture.Bitmap);
end;
end;

procedure TForm1.BitBtn5Click(Sender: TObject);
begin
    if SavePictureDialog1.Execute then
        begin
            imagel.Picture.SaveToFile(SavePictureDialog1.FileName+'.bmp');
        end;
end;

procedure TForm1.Robert_btnClick(Sender: TObject);
var
    y,x    :integer;
    m,P,p1 :PByteArray;
    temp_image:tbitmap;
begin
    Imagel.Picture.Bitmap.HandleType:=bmdib;
    Imagel.Picture.Bitmap.PixelFormat:=pf8bit;

    setImage(temp_image, imagel.Picture.Bitmap);
    For Y:=0 to temp_image.Height-3 do
        begin
            m :=temp_image.ScanLine[Y+1];
            p := imagel.Picture.Bitmap.ScanLine[y];
            p1:= imagel.Picture.Bitmap.ScanLine[Y+1];
            x:=1;
            while (x < temp_image.Width -1) do
                begin
                    m[x]:=abs(p1[x]-p[x-1]) + abs(p1[x-1]-p[x]);
                    x:=x+1;
                end
            end
        end
    end

```



```

    end;
end;
    Image1.Picture.Bitmap.Assign(temp_image);
    Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
    Image1.Refresh;

StringGrid1.Cells[0,0]:='';
StringGrid1.Cells[1,0]:='';
StringGrid1.Cells[2,0]:='';

StringGrid1.Cells[0,1]:='';
StringGrid1.Cells[1,1]:='';
StringGrid1.Cells[2,1]:='';

StringGrid1.Cells[0,2]:='';
StringGrid1.Cells[1,2]:='';
StringGrid1.Cells[2,2]:='';
end;

procedure TForm1.FormResize(Sender: TObject);
begin
    exit;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    setImage(mem_resident, Image1.Picture.Bitmap);
end;

procedure TForm1.BitBtn7Click(Sender: TObject);
begin
    Image1.Picture.Bitmap.Assign(mem_resident) ;
end;

procedure TForm1.LOG_btnClick(Sender: TObject);
var
    ray:tmaskarray;
begin
    Image1.Picture.Bitmap.HandleType:=bmdib;
    Image1.Picture.Bitmap.PixelFormat:=pf8bit;

```

```

ray[0]:= 1; ray[1]:= 2; ray[2]:= 1;

ray[3]:= 2; ray[4]:= 4; ray[5]:= 2;

ray[6]:= 1; ray[7]:= 2; ray[8]:= 1;

Imagel.Picture.Bitmap:=convolution(Imagel.Picture.Bitmap,ray,0,16);

ray[0]:=-1; ray[1]:=-1; ray[2]:=-1;

ray[3]:=-1; ray[4]:= 8; ray[5]:=-1;

ray[6]:=-1; ray[7]:=-1; ray[8]:=-1;
Imagel.Picture.Bitmap:=convolution(Imagel.Picture.Bitmap,ray,0,1);
Imagel.Picture.Bitmap:=edgedetection(Imagel.Picture.Bitmap);
Imagel.Refresh;
end;
end.

```

Appendix B

How It Works

SetImage Function

This function prepares the image to be used in edge detection

```
procedure setImage(var iniimage :tbitmap; srsImage:tbitmap);  
begin  
    iniimage:=tbitmap.create();  
    iniimage.Height:=srsImage.height;  
    iniimage.Width:=srsImage.Width;  
    iniimage.Assign(srsImage);  
  
    // ** CHANGE IMAGES TO 24 BIT PIXEL  
    iniimage.HandleType:=bmdib;  
    iniimage.PixelFormat:=pf8bit;  
end;
```

EdgeDetection Function

This function uses line-scanning algorithm to detect edges.

```
function edgedetection(SRSImage:tbitmap):tbitmap;  
var  
    m:pbytearray;  
    x,y:integer;  
begin  
    For Y:=0 to SRSImage.Height-3 do  
        begin  
            m:=SRSImage.ScanLine[Y+1];  
            x:=3;  
            while (x < SRSImage.Width -3) do  
                begin  
                    if m[x]=0 then m[x]:=1;  
                    if (m[x] > 40) then  
                        begin  
                            m[x] :=0;  
                        end  
                    else  
                        begin  
                            m[x] :=255;  
                        end  
                end  
            end  
        end
```



```

        end;
        x:=x+1;
    end;
end;
result:=srsimage
end;

```

Convolution Function

This function applies a given mask to the image, with the given offset and scale.

```

function convolution(image:tbitmap; mask:tmaskarray;
offset,scale:integer):tbitmap;
var
    y,x,tno    :integer;
    m,P,p1,p2  :PByteArray;
    temp_image:tbitmap;
begin
    setImage(temp_image,image);
    For Y:=0 to image.Height-3 do
        begin
            m:=temp_image.ScanLine[Y+1];
            p:=image.ScanLine[Y];
            p1:=image.ScanLine[Y+1];
            p2:=image.ScanLine[Y+2];
            x:=3;
            while (x < temp_image.Width-3) do
                begin
                    tno:=round(( mask[0]*p[x-1]  + mask[1]*p[x]  + mask[2]*p[x+1]
+
                    mask[3]*p1[x-1] + mask[4]*p1[x] + mask[5]*p1[x+1]
+
                    mask[6]*p2[x-1] + mask[7]*p2[x] + mask[8]*p2[x+1]
+
                    offset)/scale);
                    if tno>255 then begin tno:=255; end ;
                    if tno<0    then begin tno:=0  ; end ;
                    m[x+2]:=tno;
                    x:=x+1;
                end
            end
        end
    end

```

```

    End;
End;
result:=temp_image;
end;

-- OpenDialog
-- OpenDialog
begin
  if OpenDialog(OpenDialog) then
    begin
      str:=OpenDialog;
      Image1.Picture:=Picture;
      if str[Length] = '\0' then
        begin
          -- CloseDialog
          CloseDialog;
          Image1.Picture:=
            Image1.Picture;
          Image1.Refresh;
          // CloseDialog
          Image1.Picture:=
            Image1.Picture;
          Image1.Refresh;
        end;
      end;
    end;
  end;
  Code For Button 2:
begin
  if ShowPictureDialog(ShowPictureDialog) then
    begin
      Image1.Picture:=Picture;
      Image1.Refresh;
    end;
  end;
  Code For Button 3:
begin
  if ShowPictureDialog(ShowPictureDialog) then
    begin
      Image1.Picture:=Picture;
      Image1.Refresh;
    end;
  end;
end;

```

Button Codes

Code for Button 1:

```
var
  jpgimage:TJPEGImage;
  test:tbitmap;
  str:string;
begin
  if OpenPictureDialog1.Execute then
    begin
      str:=OpenPictureDialog1.FileName;
      image1.Picture.LoadFromFile(str);
      if (str[length(str)-2]<>'B') and (str[length(str)-2]<>'b') then
        begin
          //(-- Converting from Jpg to BMP--
          jpgimage:=tjpegimage.Create ;
          jpgimage.Assign(image1.Picture);

          test:=tbitmap.Create ;
          test.Assign(jpgimage);
          jpgimage.Destroy;
          //-----}
          image1.Picture.Bitmap.Assign(test);
          test.Destroy;
        end;
        setImage(mem_resident, image1.Picture.Bitmap);
      end;
    end;
```

Code for Button 2:

```
begin
  if SavePictureDialog1.Execute then
    begin
      image1.Picture.SaveToFile(SavePictureDialog1.FileName+'.bmp');
    end;
  end;
```

Code for Button 3:

```
begin
  Image1.Picture.Bitmap.Assign(mem_resident) ;
end;
```


Code for Button 4:

```
var
    ray:tmaskarray;
begin
    Image1.Picture.Bitmap.HandleType:=bmdib;
    Image1.Picture.Bitmap.PixelFormat:=pf8bit;

    ray[0]:=-1; ray[1]:= 0; ray[2]:= 1;

    ray[3]:=-2; ray[4]:= 0; ray[5]:= 2;

    ray[6]:=-1; ray[7]:= 0; ray[8]:= 1;

    Image1.Picture.Bitmap:=convolution(Image1.Picture.Bitmap,ray,0,1);
    Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
    Image1.Refresh;

    StringGrid1.Cells[0,0]:='-1';
    StringGrid1.Cells[1,0]:='0';
    StringGrid1.Cells[2,0]:='1';

    StringGrid1.Cells[0,1]:='-2';
    StringGrid1.Cells[1,1]:='0';
    StringGrid1.Cells[2,1]:='2';

    StringGrid1.Cells[0,2]:='-1';
    StringGrid1.Cells[1,2]:='0';
    StringGrid1.Cells[2,2]:='1';
end;
```

Code for Button 5:

```
var
    ray:tmaskarray;
begin
    Image1.Picture.Bitmap.HandleType:=bmdib;
    Image1.Picture.Bitmap.PixelFormat:=pf8bit;

    ray[0]:=-1; ray[1]:= 0; ray[2]:= 1;

    ray[3]:=-1; ray[4]:= 0; ray[5]:= 1;
```

```

ray[6]:=-1; ray[7]:= 0; ray[8]:= 1;

Image1.Picture.Bitmap:=convolution(Image1.Picture.Bitmap,ray,0,1);
Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
Image1.Refresh;

StringGrid1.Cells[0,0]:='-1';
StringGrid1.Cells[1,0]:='0';
StringGrid1.Cells[2,0]:='1';

StringGrid1.Cells[0,1]:='-1';
StringGrid1.Cells[1,1]:='0';
StringGrid1.Cells[2,1]:='1';

StringGrid1.Cells[0,2]:='-1';
StringGrid1.Cells[1,2]:='0';
StringGrid1.Cells[2,2]:='1';
end;

Code for Button 6:
var
ray:tmaskarray;
begin
Image1.Picture.Bitmap.HandleType:=bmdib;
Image1.Picture.Bitmap.PixelFormat:=pf8bit;

ray[0]:=-1; ray[1]:=-1; ray[2]:=-1;

ray[3]:=-1; ray[4]:= 8; ray[5]:=-1;

ray[6]:=-1; ray[7]:=-1; ray[8]:=-1;

Image1.Picture.Bitmap:=convolution(Image1.Picture.Bitmap,ray,0,1);
Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
Image1.Refresh;

StringGrid1.Cells[0,0]:='-1';
StringGrid1.Cells[1,0]:='-1';
StringGrid1.Cells[2,0]:='-1';

```

```

StringGrid1.Cells[0,1]:='-1';
StringGrid1.Cells[1,1]:='8';
StringGrid1.Cells[2,1]:='-1';

StringGrid1.Cells[0,2]:='-1';
StringGrid1.Cells[1,2]:='-1';
StringGrid1.Cells[2,2]:='-1';
end;

Code for Button 7:

begin
  Image1.Picture.Bitmap.HandleType:=bmdib;
  Image1.Picture.Bitmap.PixelFormat:=pf8bit;

  setImage(temp_image,image1.Picture.Bitmap);
  For Y:=0 to temp_image.Height-3 do
  begin
    m:=temp_image.ScanLine[Y+1];
    p:= image1.Picture.Bitmap.ScanLine[y];
    p1:= image1.Picture.Bitmap.ScanLine[Y+1];
    x:=1;
    while (x < temp_image.Width -1) do
    begin
      m[x]:=abs(p1[x]-p[x-1]) + abs(p1[x-1]-p[x]);
      x:=x+1;
    end;
  end;

  image1.Picture.Bitmap.Assign(temp_image);
  Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
  image1.Refresh;

  StringGrid1.Cells[0,0]:='';
  StringGrid1.Cells[1,0]:='';
  StringGrid1.Cells[2,0]:='';

  StringGrid1.Cells[0,1]:='';
  StringGrid1.Cells[1,1]:='';
  StringGrid1.Cells[2,1]:='';

```



```

StringGrid1.Cells[0,2]:='';
StringGrid1.Cells[1,2]:='';
StringGrid1.Cells[2,2]:='';
end;

```

Code for Button 8:

```

var
    ray:tmaskarray;
begin
    Image1.Picture.Bitmap.HandleType:=bmdib;
    Image1.Picture.Bitmap.PixelFormat:=pf8bit;

    ray[0]:= 1;  ray[1]:= 2;  ray[2]:= 1;

    ray[3]:= 2;  ray[4]:= 4;  ray[5]:= 2;

    ray[6]:= 1;  ray[7]:= 2;  ray[8]:= 1;

    Image1.Picture.Bitmap:=convolution(Image1.Picture.Bitmap,ray,0,16);

    ray[0]:=-1;  ray[1]:=-1;  ray[2]:=-1;

    ray[3]:=-1;  ray[4]:= 8;  ray[5]:=-1;

    ray[6]:=-1;  ray[7]:=-1;  ray[8]:=-1;

    Image1.Picture.Bitmap:=convolution(Image1.Picture.Bitmap,ray,0,1);
    Image1.Picture.Bitmap:=edgedetection(Image1.Picture.Bitmap);
    Image1.Refresh;
    StringGrid1.Cells[0,0]:='';
    StringGrid1.Cells[1,0]:='';
    StringGrid1.Cells[2,0]:='';
    StringGrid1.Cells[0,1]:='';
    StringGrid1.Cells[1,1]:='';
    StringGrid1.Cells[2,1]:='';
    StringGrid1.Cells[0,2]:='';
    StringGrid1.Cells[1,2]:='';
    StringGrid1.Cells[2,2]:='';
end;

```