IMAGE EDGE DETECTION SYSTEM

AZIMAH BINTI RAZALI

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2002/2003

Abstract

Image edge detection is not a new thing in image processing. It has been applied so many years ago as one of the technique or method to produce a variation of image display either for medical, research or art. For example in medical field, they used this method for X-Ray.

This thesis describes edge detection in details including the edge definition, the types of edge, detection methods, the problems with edge, the advantages and disadvantages and the most important is how to detect edge in digital image, either two-dimensional or three-dimensional images.

In this system, two main methods that are implemented are Laplacian and Gaussian. For the Gaussian method, it divides to two parts, the first one for one-dimensional image called One Dimensional Gaussian and the other one is fir two-dimensional image called Two Dimensional Gaussian. Beside these three methods, there are three other operators that are also used in image detection process. They are Robert operator, Sobel operator and Prewitt operator. But the user does not need to use all of these operators, they just need to chose any one of them to be applied on their scanned images. Though all of these operators are functioning in three different ways, the result is still the same.

In other word, this thesis explores the methods or techniques of image edge detection in detail until software that can detect edge was developed and available to use by users.

Acknowledgement

This thesis is written as lecturers, friends and family provide technical and emotional support. It is impossible to list here all those who helped to sustain the author during the development of this thesis and the author apologize in advance for any omissions.

First and foremost the author wishes to express her deepest gratitude to her supervisor, Puan Nornazlita Hussin, who had inspired and supervised the author form time to time until the completion of this thesis. Without her guidance, this thesis would not have propelled in the right direction to achieve the goals.

Particular thanks go to Miss Nur Aniza Abdullah as a moderator that willing to sit and discuss ideas and provided suggestions regarding the initial proceeding of system development. Also, I wish to express my appreciation to my new moderator, Mr. Phang Keat Keong, for his advice and pleasure to evaluate my thesis. The author also appreciates the guideline that the Faculty of Computer Science and Information Technology had in making this thesis possible.

Finally, the author want to express her deep appreciation for the support and encouragement of her parent, Encik Razali Mohd Yusof and Puan Noor Asiah Said, family, friends and as always, Maznan Mazlan.

List of content

List of table

List of figure

1.1 Project Overview

Image edge detection is a system which able to detect edge line or boundary of an images. It is one of the image manipulation processes in recent image processing to create an edge image that usually used in medical and educational field.

In this system, the two main methods that are implemented are Laplacian and Gaussian. For the Gaussian method, it divides to two parts; the first one for one-dimensional image called One Dimensional Gaussian and the other one is for two-dimensional image called Two Dimensional Gaussian. Beside these three methods, there are three other operators that are also used in image detection process. They are Robert operator, Sobel operator and Prewitt operator. But the user does not need to use all of these operators; they just need to choose any one of them to be applied on their scanned images. Though all of these operators are functioning in three different ways, the result is still the same.

For the development of this system, waterfall model has been chosen. It is because this waterfall methodology is easy to understand especially for those who really new in system development. The process of the development is shown clearly step-by-step, so it is much easier to follow.

All the source cord was written in Matlab 6.1, which is one of the programming languages that specialized in mathematical, based programming.

1.2 Goals and Objectives

The main purpose of this thesis to do a thorough study and analysis on image edge detection as well as their characteristic in order to have a clear understanding about edge detection method, their relation and importance in the real world images.

It is also to gain knowledge and experience on how to developed a system or software.

The objectives of this system are:

- To provide an easy way to detect edge
- To minimize the time spent of edge detection process
- To create a good but simple way to let the users express their idea about the edges they have in mind regarding a specific image
- To implement a method to detect the type of edges a user ordered

1.3 Scope

1.3.1 System scope

Generally, this system performed a process of image edge detection based on the value of threshold that included by user. Basically the process that includes in this system are input the image from user, pre-procession to make sure the detection is accurate, detection process, display the preview image, print out the original image and the edge image. There are two methods used in detection process, the Laplacian and the Gaussian. All of the coding was written in MATLAB, which is one of the programming languages that specialized in mathematical based programming.

1.3.2 User Scope

Image edge detection system consists of four main users, which are:

- Kindergarten teacher

  The edge detection is useful for touching up scanned drawing or cartoon pictures. It helps a teacher, which wants to scan pictures and get rid of the colour, so that the kids could then colour it in.

- Pictures editor

  It is very important to the pictures editor to be creative on displaying images in all types of printed media such as magazine and newspaper and also in virtual media like web page. They can manipulate the edge images to makes their pictures looks more interesting or artistic.

- University or College student

  Students who studying in image processing (usually in majoring of information technology) will have opportunities to practice the edge detection theory that they had learn in class by using this system.

1.4 Project Planning

The compulsory step of doing any project is project planning. After defining the project and the problems, setting the project objectives and ensuring the scope of the project, the next step is project planning. This step is very crucial in terms of getting on to the right course in the remaining studies of the new system.

There are a few planning that must be made here which are important to gather useful and related information in the development of the new system. The planning are:

i. Deciding the source of information
ii. Extracting only useful and related information
iii. Studies made on the information gathered
iv. Analysis and make system draft
v. Decide type of system development tools
vi. Design the system

vii. Test the new system

viii. Adjustment and enhancement

ix. Implement the system

x. Maintenance of the system

Usually the planning phase is carried out indirectly because some ways not prepare on paper and just base on the idea of the planning itself. However for this project, the planning is done quite formal and the follow-ups are very much according to the planning itself.

However there are some works that were carried out without planning or that is exclude from the planning lists. For example types of reading materials were not picked according to what is planned but based in what is available. But only materials that related to what we have planned is considered and extracted.

The unplanned activities can be managed properly with a project management. This is good for final review of the overall process of the project itself.

1.5 Project Schedule

Project schedule plays an important role in planning and developing the system. It specifies all the activities involve in system development and the duration of time for each activity to successfully implement the project. The project schedule for this project is shown in below:

Table 1.0 : Project Planned Schedule

Task Name	July, 2002				Aug, 2002				Sept, 2002				Oct, 2002				Nov, 2002				Dec, 2002				Jan, 2003				Feb, 2003				Duration (in weeks)
	week				week				week				week				week				week				week				week				
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	
1. Identifying problem, oppurtinities and needs																																	02
2. Determining information requirements																																	04
3. Analyzing system needs																																	06
4. Designing the recommended system																																	04
5. Developing and documenting software																																	16
6. Testing and maintaining the system																																	02
7. Implementing and evaluating system																																	02

1.6 *Chapters Summary*

a. Chapter 1 - Introduction

This chapter gives an introduction to the system and its objectives, as well as an overview of the proposal of the project. Sections contained here are the project overview, the goals and objectives, project scope, plan and last but not least the schedule of the project.

b. Chapter 2 - Literature Review

In this chapter, the review of literature will be presented and existing similar system will be analyzed. Research finding, review of literature survey, summarization on analysis and synthesis of all the literature reviewed will be indicated. This chapter required reference from the Internet, books, articles in journals, newspaper and other relevant sources.

c. Chapter 3 - Methodology

This chapter clearly identifies the methodology, mechanism and approach to be adopted. The quality of the proposed tools refers to the practicality of the chosen tools, effectiveness and appropriateness in solving the problem is presented.

d. Chapter 4 - System Analysis

This chapter describes all the system and user's requirement. In this phase, all the system requirements like functional and non-functional requirements; hardware and software requirements are identified and analyzed the problems possibilities.

e. Chapter 5 – System Design

The various components of the proposed system are clearly identified and explained in this chapter. The components include the architectural design, database design, functional design and also user interface design.

f. Chapter 6 – System Implementation

Under the specified design and development - operating environment and in accordance to the design blueprints, the system is developed. Following that, the system is implemented in the usual environmental.

g. Chapter 7 – Testing and Evaluation

The approaches for debugging and testing of the system are described here. The objectives, both achieved and unachieved are outlined and the proposals of future work are considered. The problem faced and solutions taken during the development period are highlighted.

h. Chapter 8 – Conclusion and Future Enhancements

Following the conclusion on the finished system, the strength and limitations of the final product are confirmed. A proposal for future enhancement is forwarded

here and also an overall conclusion based on the project development proposal is provided.

1.7 Chapter Summary

This chapter describes the project to be developed. The project overview or introduction described all the phase involved in system development. Each phase have different activities and all the activities are explained briefly in this part. The objectives, goals and scope were explained clearly so that the system will be developing based on these main thing. There is also project plan schedule in order to make sure the development process successfully complete in effective way.

2.1 Research methods

i. Book and references

Material such as books, magazines, journals, newspapers and thesis were read through for new ideas and to make comparisons. New methods were analyzed to see if they are suitable in the system environment.

ii. Internet research

Researched on the World Wide Web was done to look out for similar system and new technologies of the current software developments tools. The Internet search engines those were useful in the quest are as follow:

http://www.google.com

http://www.lycos.com

http://www.altavista.com

iii. Newsgroup

Newsgroups were also useful to discuss FAQs, topics such as development tools, system architectures, database, programming codes and others. Questions can be posted and respondents would give their ideas and suggestions. The useful newsgroups are as follow:

http://www.ask.yahoo.com

http://www.tanya@putera.com

http://www.e-pedoman.com

iv. Document room

There are a lot of theses from seniors stored at the document room of Faculty of Computer Science and Information Technology (FSKTM). Therefore, all the documents can provide some of the guidelines on how to do this thesis. These samples are useful to provide basic guideline and idea on how to generate a good report, by evaluating the strength and weakness of their work.

2.2 Introduction

Unlike the real world, images do not have edges. Images have abrupt changes in intensity. Therefore, the term edge detection is not actually an accurate phrase. But, since the overall goal is to locate edges in the real world via an image, the term edge detection is commonly used.

An edge is not a physical entity, just like a shadow. It is where the picture ends and the wall starts. It is where the vertical and the horizontal surfaces of an object meet. In reality, what appears to be an edge from the distance may even contain other edges when looked close-up. Edges are scale-dependent and an edge may contain other edges, but at a certain scale, an edge still has no width

2.3 What is edge detection?

Extracts and localizes points (pixels) around which a large change in image brightness has occurred. The performance of higher -level processes such as extraction of object contours and object recognition rely heavily on the correctness and completeness of edges. Noise produced by imaging and sampling processes causes the majority of problems in edge detection. There are two classes of edge detection algorithms with noise smoothing. One of these

classes is based on regularization, which is achieved by imposing smoothness constraints on the solution of edge points in various forms such as minimizing energy functional. Another class of edge detection algorithms employs various noise smoothing processes before the actual detection procedure. A low pass filter can achieve noise smoothing, which is a convolution with a kernel.

2.3.1 Edge types

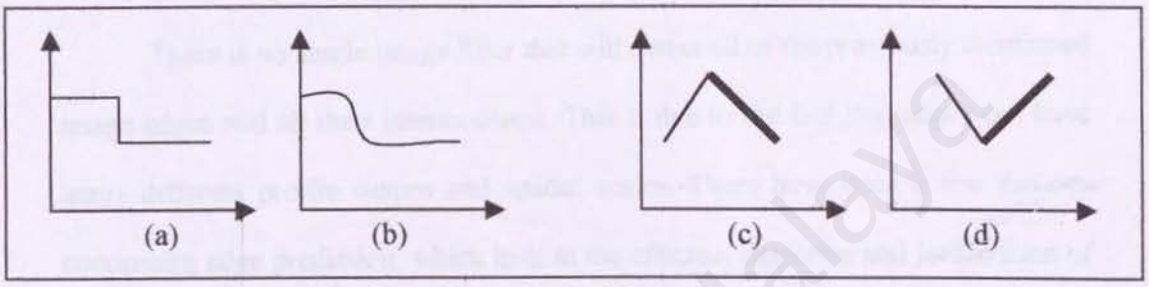


Figure 2.0 : Sharp Step, Gradual Step, Roof and Through.

All edges are locally directional. Therefore, the goal in edge detection is to find out what occurs perpendicular to an edge. The following is a list of commonly found edges. A Sharp Step, as shown in Figure 2.0(a), is an idealization of an edge. Since an image is always band limited, this type of graph cannot ever occur. A Gradual Step, as shown in Figure 2.0(b) is very similar to a Sharp Step, but it has been *smoothed out*. The change in intensity is not as quick or sharp. Many changes in image intensity will be a continuum of widths or spatial extents between the Sharp Step and the Gradual Step.

A Roof, as show in Figure 2.0(c) is different than the first two edges. The derivative of this edge is discontinuous. A Roof can have a variety of sharpness, widths, and spatial extents. The Trough, also shown in Figure 2.0(d) is the inverse of a Roof.

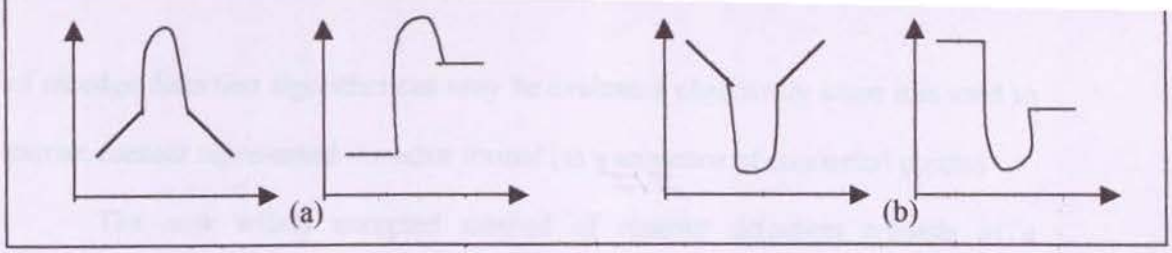


Figure 2.1 : Highlights and Lowlights

Edges can also found to be any combination of all the above. Figure 2.1 shows a few such variations.

2.3.2 Finding edges

There is no single image filter that will detect all of the previously mentioned image edges and all their intermediates. This is due to the fact that edges can have many different profile shapes and spatial scales. There have been a few theories concerning edge prediction, which help in the efficient detection and localization of edges.

2.3.3 The importance of edge

Edges contain most of the information in an image while being represented far more compactly than the image itself and the first step in image *segmentation*, the partitioning of an image into meaningful regions.

2.3.4 Edge detection concepts

Edge detection is a hill defined term, because it makes think that the algorithm gives contours as result. In fact these algorithms give images, which show higher intensity in pixel near gray value transitions. In some way, this task is only *contour enhancement*. Our conclusion from this experience is that real edge detection must produce as output: vector data representing contours. Any other results are only pretty images to make demos for visitors. In other word, the quality

of an edge detection algorithm can only be evaluated objectively when it is used to extract contour represented in vector format (as a sequence of connected points).

The now widely accepted method of contour detection consists of a smoothing filtering followed by a derivative filtering. Smoothing is intended to reduce noise in the image without eliminating contours, usually a compromise should be found to the degree of smoothing with respect to the type of noise present in the image and the sharpness of contours. The derivative filter detects transitions or changes in gray levels in the image. Usually second derivative are used to determine precisely the location of maximum rate of change in gray level.

2.3.5 Edge Detectors

Edges are places in the image with strong intensity contrast. Since edges often occur at image locations representing object boundarie

s, 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 highpass 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 2.2.

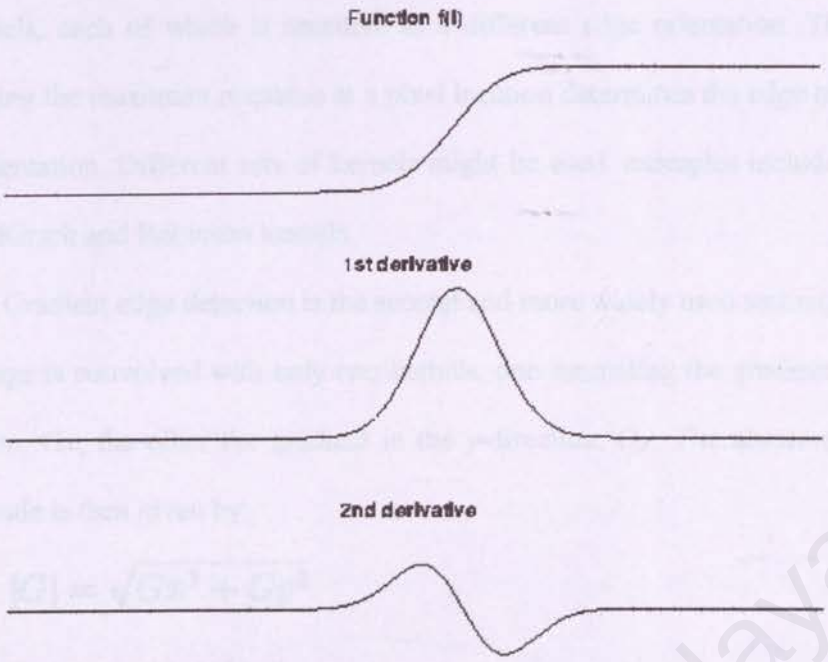


Figure 2.2 1st and 2nd derivative of an edge illustrated in one dimension.

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:

$$\frac{d\,f(i)}{d(i)} = f(i+1) - f(i)$$

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 a 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, *Gx*, the other the gradient in the *y*-direction, *Gy*. The absolute gradient magnitude is then given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

and is often approximated with:

$$|G| = |Gx| + |Gy|$$

In many implementations, the gradient magnitude is the only output of a gradient edge detector, however the edge orientation might be calculated with:

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

The most common kernels used for the gradient edge detector are the Sobel, Roberts Cross and Prewitt operators.

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 *hysteresis 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. This problem is addressed in the Canny and Marr operators by convolving the image with a smoothing operator (Gaussian) before calculating the derivative.

2.3.6 Other Methods of Edge Detection

There are many ways to perform edge detection. However, the most may be grouped into two categories, gradient and Laplacian. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. The Laplacian method searches for zero crossings in the second derivative of the image to find edges. This first figure (figure 2.3) shows the edges of an image detected using the gradient method (Roberts, Prewitt, Sobel) and the Laplacian method (Marrs-Hildreth).

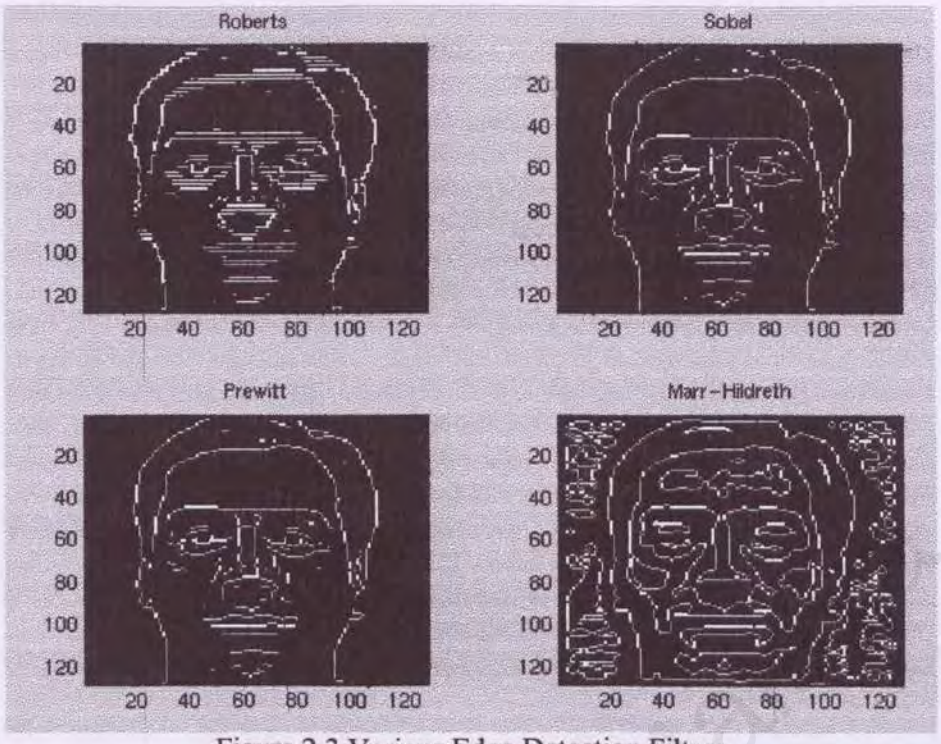


Figure 2.3 Various Edge Detection Filters

Notice that the facial features (eyes, nose, mouth) have very sharp edges. These also happen to be the best reference points for morphing between two images. Notice also that the Marr-Hildreth not only has a lot more noise than the other methods, the low-pass filtering it uses distorts the actual position of the facial features. Due to the nature of the Sobel and Prewitt filters we can select out only vertical and horizontal edges of the image as shown below. This is very useful since we do not want to morph a vertical edge in the initial image to a horizontal edge in the final image. This would cause a lot of warping in the transition image and thus a bad morph.

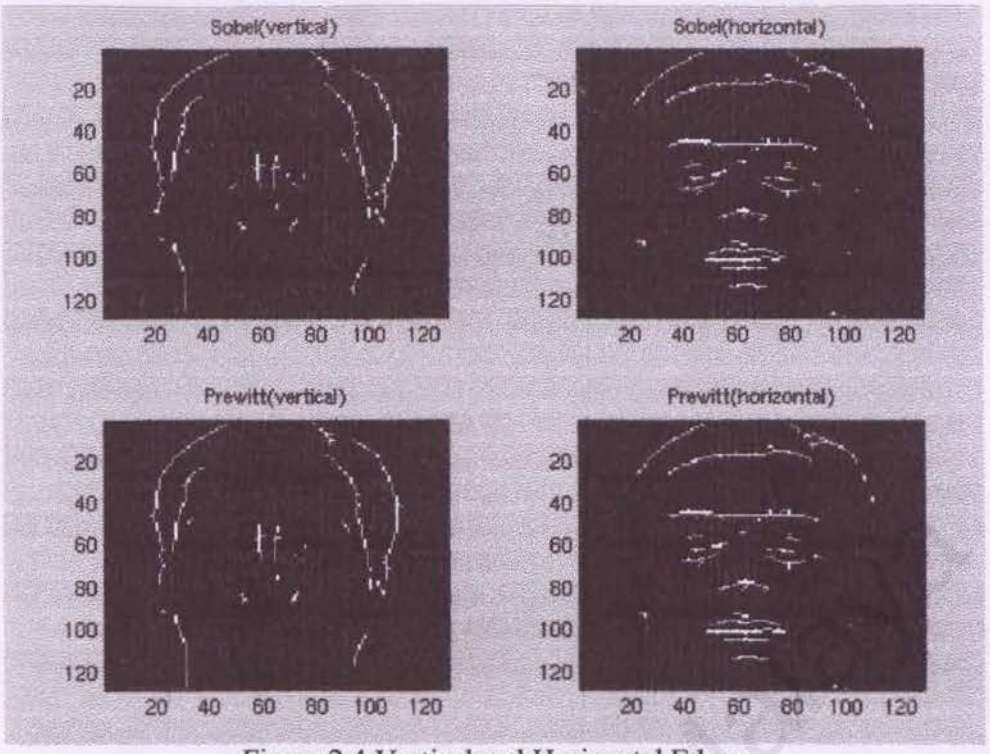


Figure 2.4 Vertical and Horizontal Edges

The next pair of images (figure 2.4) shows the horizontal and vertical edges selected out of the group members images with the Sobel method of edge detection. You will notice the difficulty it had with certain facial features, such as the hairline of Sri and Jim. This is essentially due to the lack of contrast between their hair and their foreheads.

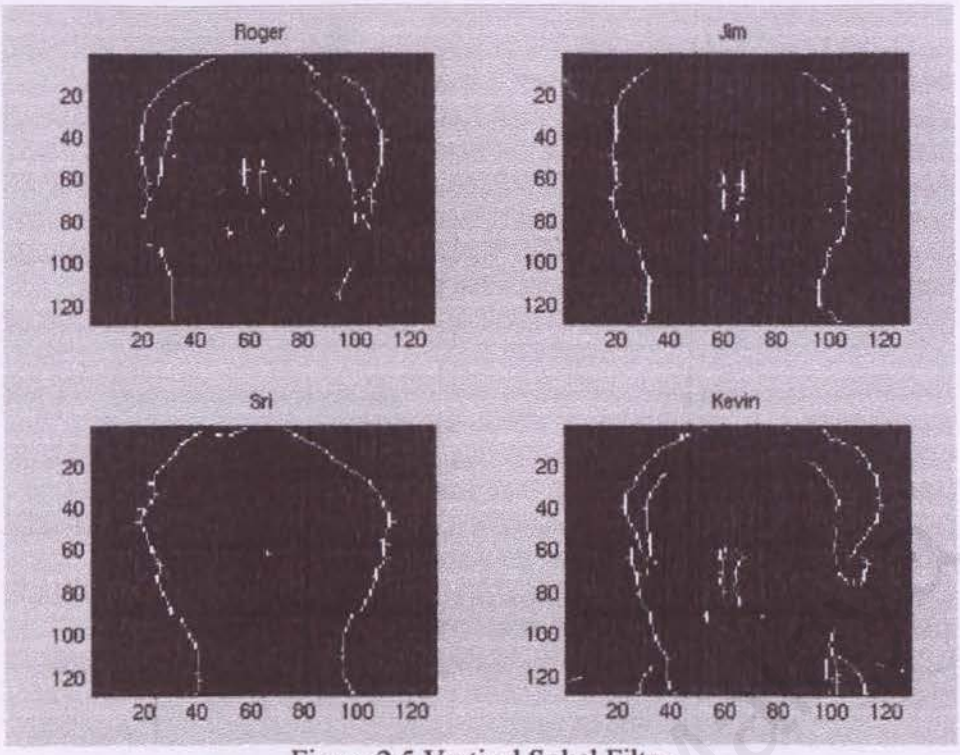


Figure 2.5 Vertical Sobel Filter

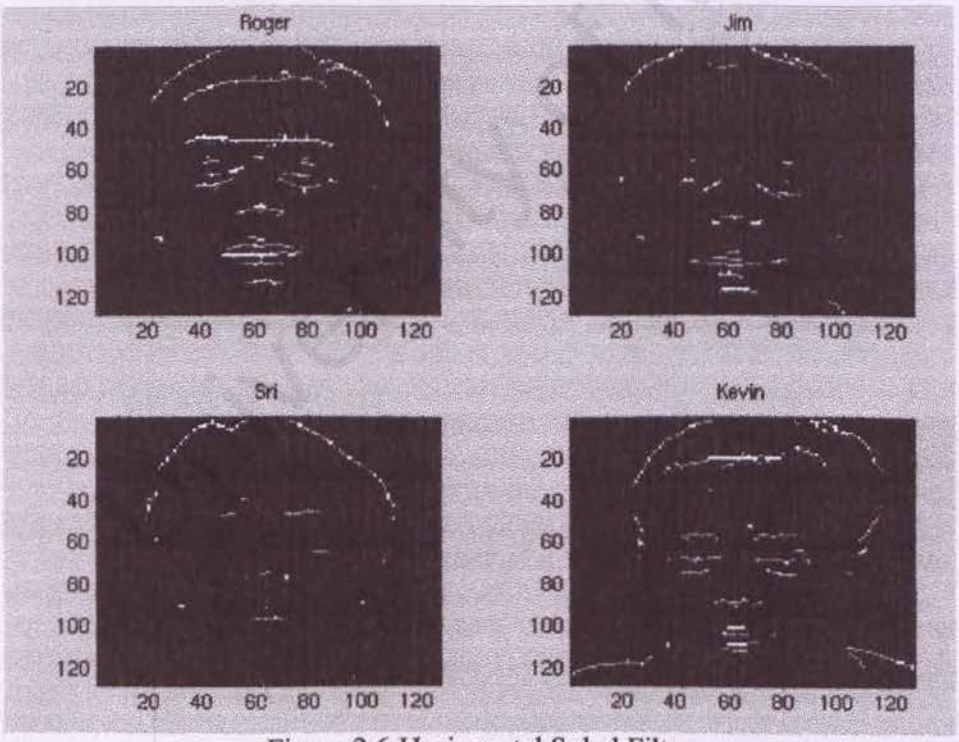


Figure 2.6 Horizonatal Sobel Filter

We can then compare the feature extraction using the Sobel edge detection to the feature extraction using the Laplacian.

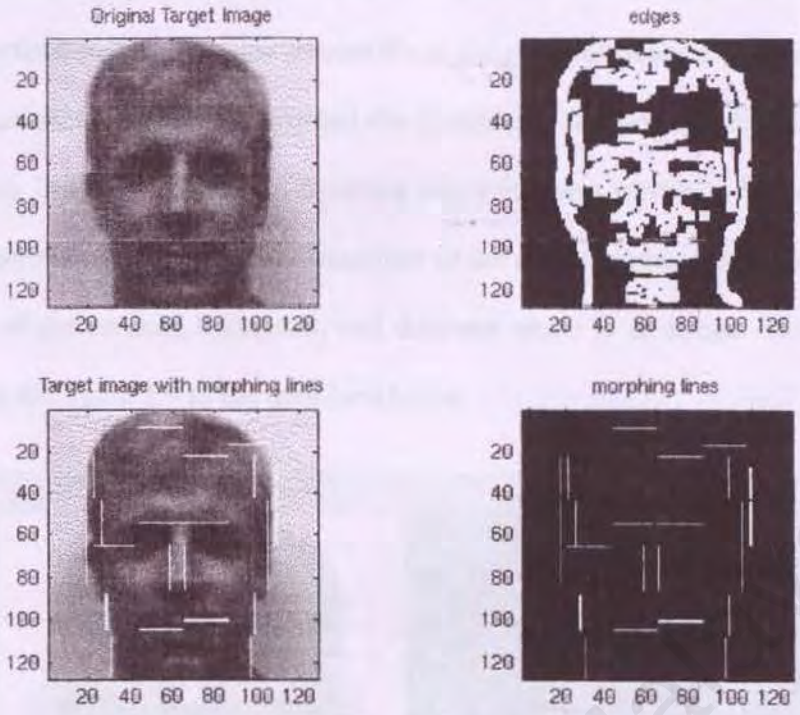


Figure 2.7 Sobel Filtered Common Edges: Jim

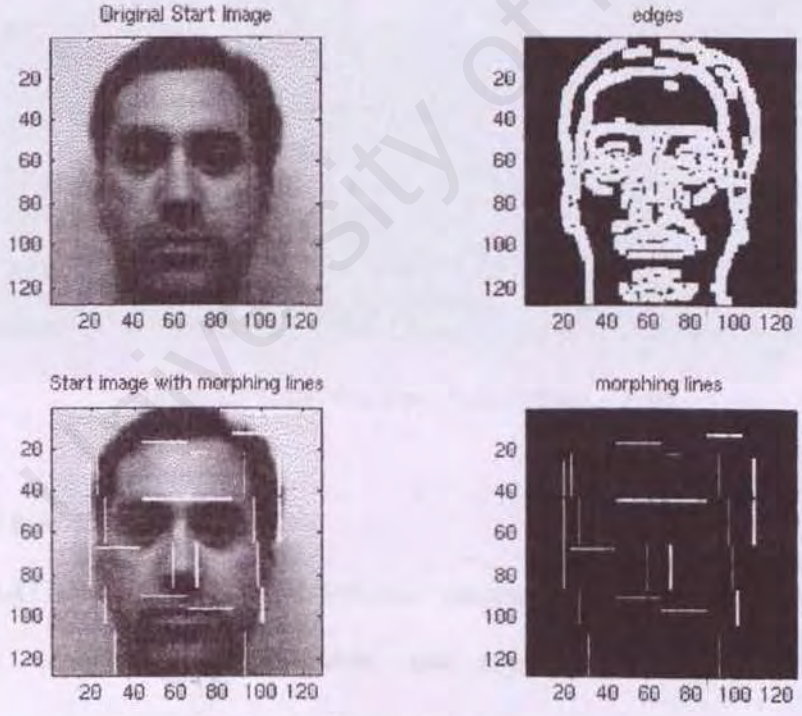


Figure 2.8 Sobel Filtered Common Edges: Roger

We see that although it does do better for some features (ie. the nose), it still suffers from mismapping some of the lines. A morph constructed using

individually selected points would still work better. It should also be noted that this method suffers the same drawbacks as the previous page; difficulties due to large contrast between images and the inability to handle large translations of features. Another method of detecting edges is using wavelets. Specifically a two-dimensional Haar wavelet transform of the image produces essentially edge maps of the vertical, horizontal, and diagonal edges in an image. This can be seen in the figure 2.9 of the transform below.

Figure 2.9 Haar Wavelet Transformed Image

2.4 What is MATLAB?

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB uses software developed by the LAPACK and ARPACK projects, which together represent the state-of-the-art in software for matrix computation. MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing,

control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

2.4.1The MATLAB System.

The MATLAB system consists of five main parts:

i. Development Environment.

This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, and browsers for viewing help, the workspace, files, and the search path.

ii. The MATLAB Mathematical Function Library.

This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

iii. The MATLAB Language.

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

iv. Handle Graphics®.

This is the MATLAB graphics system. It includes high-level commands for two dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that

allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

v. The MATLAB Application Program Interface (API).

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

2.5 Current available system

One of the application that used edge detection is Adobe Photoshop 5.5, which is one of the hottest authoring tools nowadayas. This application does not just detect edge, but it manipulated the detection of edge as a way to produce a variation of image display. It provides some types of filters function that implemented the edge detection process. The types of filters function that used edge detection process are :

i. Blur named as "Gaussian blur"

Origimal Image

After "Gaussian Blur"

ii. Brush strokes named as "Accented edges"

Origimal Image

After "Accented edges"

a. Sharpen named as "Sharpen edges"

Origimal Image

After "Sharpen edges"

b. Stylize named as:

a) "Find edges"

Origimal Image

After "Find edges"

b) "Glowing edges"

Origimal Image

After "Glowing edges"

2.6 Chapter Summary

In this chapter, literature review is carried out. Literature review helps to understand the system to be developed as well as the tools using to develop this system. The comparison was being carried out between the current system and the system to be developed in order to identify the main function, how the function operated and the application of the functions. All this review will then give some view for the developers to build their system and also to improve the system so that it can cover the current system constraints or drawbacks.

A few technologies are reviewed in order to choose the best approach to implement the system. There are many development tools that are available in the market. Different development tool offers different approach. Each tool has its own strengths and weakness. So, this chapter intends to present some of the features offered by certain development tools and platforms. The development tool reviewed is programming language, which is MATLAB.

After the literature review was carried out, it is definitely resulted a more understanding of the system to be developed.

3.1 Introduction and concept of methodology.

Methodology is the study that deals with the science of method concerned with the application or the principles of reasoning scientific and philosophical enquiry. It deals with the philosophical assumptions underlying the development process.

A system development methodology is a very formal and precise system development process that defines a set of activities, methods, best practices, deliverables and automated tools for system developers to use to develop and maintain most or all information system and software. Therefore, any methodologies chosen for image edge detection will ensure that a consistent and reproducible approach is applied to this project. Methodologies will reduce the risk associated with shortcuts and mistakes as well as produces a complete and consistent documentation.

Methodologies enable one to follow a certain procedures, where it basis is laid down in the way a problem is encountered. Methodologies are also flexible enough to provide for different types of projects and strategies. In general, a methodology should fulfill two basic requirements that are effective support of design process and efficient control of project.

1. Effective support of design process.
    - Provide the means to identify the different steps in the system development process. It involved from system logic reasoning, semantic modeling towards syntax specification.

- Provide the means to set boundaries to system environment. It is required to set boundaries and to take relevant aspects into consideration.
- An immediate consequence of the previous requirement that the methodology should support stepwise refinement in the design process. The concept of decomposition of a total system into subsystems is required in order to reduce complexity.

2. Efficient control of project.
   - Provide tools to an efficient control of a project. Most of the time, a methodology uses the well-known concept of activity planning, deliverable and milestone definition in the different stages of a system development project, no matter how it applies to the design phase or in the implementation phase.

3.2 System Development Methodology.

For the development of image edge detection, waterfall model has been chosen to assist in this project. The waterfall methodology is an industry standard, a time-proven approach to system development. This is very powerful methodology. It simply states that first, one should think about what is being built, and then establish the plan for how it should be built and then built it.

The waterfall methodology forces analysis and planning before actions are taken. This is good advice in many situations. The process forces analysis to be done and to be precisely define the requirements of the system. It is much easier to build something if it is known what that something is. It forces a discipline process to avoid the pressure of writing code long before it is known what is to be built.

This provides an orderly sequence of development steps and helps ensure the adequacy of documentation and design reviews to ensure the quality, reliability and maintainability of the development system. It is called the waterfall methodology because each phase flows naturally into the next phase like water over a series of falls. Some of the advantages of waterfall methodology are:

- Easy to follow during system development and maintenance could be done at each phase.
- The process of the development is systematic as each phase is fix and determine in the beginning of the system development.
- Good visibility because each process produce some deliverables.

The waterfall methodology does one thing very nicely that is to outlines the steps required for developing software. What it does not do is take into account truly iterative development or properly model the process of software creation. Many methodologies start with the waterfall process with modifications to address these problems. Figure 3.0 below illustrates this methodology:

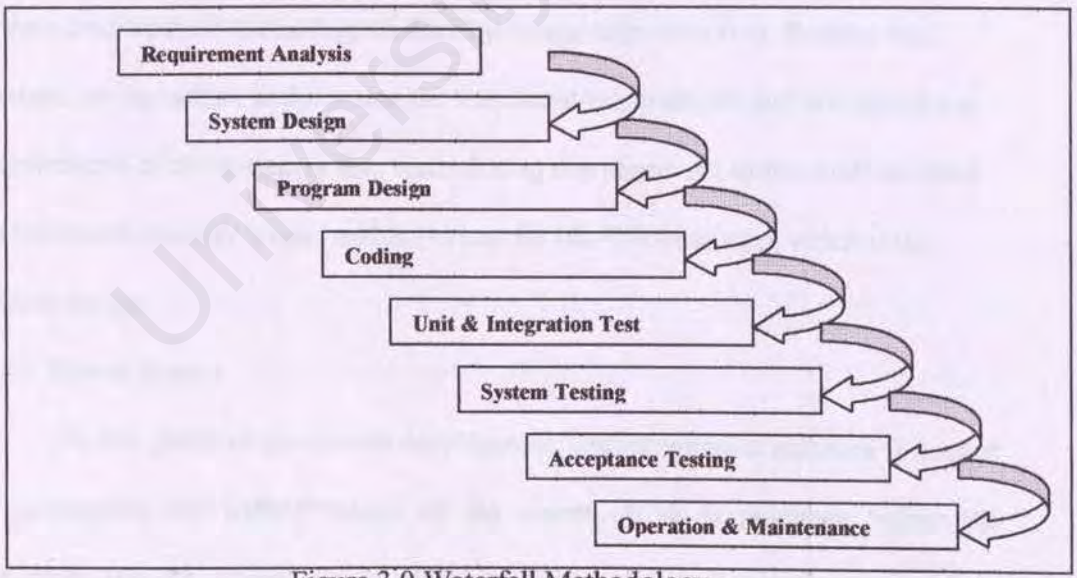


Figure 3.0 Waterfall Methodology

From figure 3.0, the waterfall methodology is very important in order to make sure that the project has been well planned from the beginning stage until the end of this project. To guarantee the success of this project, researches have been made on the related fields and system planning has been done based on the approaches provided by this methodology. Besides that, experiment could also be done to determine the usage of techniques and design ideas. All the steps are explained and elaborated in the followings:

3.2.1 Requirement Analysis

The first is to identify the problems, objective and the scope of developing image edge detection system. This step is very important because addressing the wrong objectives and scope of this system will pretty much affect the outcome of the project. Besides that, other task carried out in this phase will be literature review on edge such as the types of edge, how to finding edge, and edge detection concept.

3.2.2 System Analysis

The next phase is to analysis the system needs and chooses the suitable system development technology to develop image edge detection. Besides that, analysis on the system architecture, the functional requirements and non-functional requirements of the system is also made during this phase. All of the analyses done on this phase are very crucial and important for the following step, which is the system design.

3.2.3 System Design

In this phase of the system development, the information gathered is needed to accomplish the logical design of the system. It is as guidance before the implementation of the real system. Here, the design of overall system structure, flow charts, page flows and accurate data flow diagrams are planned, so that the processes

within the system are functional and correct. The objective of this phase is to visualize each module to be developed in the system.

3.2.4 Program Design

Program design of the system involved the translation of the software representation produce by the design phase into a computer readable form. Therefore, this portion involves the coding of the system and also the use of development tools.

3.2.5 Coding

After all the codes or scripts and the system requirements are ready, the next step is to implement that codes. Implementation is a procedure to integrate the entire system that is being developed, which includes all the hardware and software in order for it to function properly and as a complete system.

3.2.6 Unit and Integration Testing

During this phase, the software design is realized as a set of program units. Unit testing involves verifying that each unit meets its specifications. The individual program units are integrated and tested as a complete system to ensure that the system requirements have been met. After testing, the system is approved by supervisor.

3.2.7 System testing

System testing is very important to assure the quality of the system. The main objectives of system testing are to detect the faults or errors in developed system so that it can be corrected before the system is fully operational.

3.2.8 Acceptance Testing.

After the system is fully operational and ready to use by users, its need to be test by some users first. Usually, users were chosen randomly. They will use the

completed system and give feedback to the developer whether the system fulfill their specification needs or not.

3.2.9 Operation and Maintenance

Maintenance of the system can be described as the process of changing the system after it is under operation. The changes may involve simple changes to correct existing errors, more extensive changes to correct design errors or significant enhancement to correct certain specification.

3.3 Chapter Summary

The methodology chosen to develop Image Edge Detection System is Waterfall Model. This methodology will serve as the base of the whole development. In this chapter, the reason why this methodology has been chosen was explained in detail. Each steps in this methodology is briefly described, so that the developer can easily understand the concept of this methodology.

The advantages of Waterfall Model also included in this chapter. Some of the advantages of waterfall methodology are easy to follow during system development and maintenance could be done at each phase, the process of the development is systematic as each phase is fix and determine in the beginning of the system development and good visibility because each process produce some deliverables.

4.1 Problem Analysis

The first step that the developers must do before deciding a system and user requirements is defining the possibility of problems that will exist in the system. The problems defined then should be as one of the developer's guideline in developing their system. In this image edge detection system, there are a few problems that will possibly faces by user, which are:

i. The system does not have ability to support all the types of images that input by user. It depends on the texture or the pattern of the image.

ii. The system cannot detect edge accurately if there have any noises on the image. Usually, noises are difficult to detect by user's naked eye.

iii. The end-user will not know the use of threshold value, what is applicable value that they suppose to key in, or the importance of that in edge detection process; how's the value impressed their edge detection process.

iv. This system provided a variety of edge detection methods or operators that are Laplacian, Gaussian, Sobel, Prewitt and Robert operators. Although all these operators produce the same result, which is edge image, there will be some confusing in user's thoughts in deciding which operators that they must use and what's the difference between all the operators.

v. Extensive user's learning time. The user needs to explore all the operators provided in the system.

Based on these problems, the approaches or the solutions that suitable to implement in this system are:

i. Give a brief description on edge detection process in general way includes the concept of edge detection, the threshold value and the types of image that system supported. This description will be one of the elements in user interface.

ii. Provide a pre-processor to clean out the image noises. This process should be able to running automatically before the edge detection process. It seems like the user does not realize the existence of this process in the system.

iii. Give a short description about each operator that contains any related information like the advantages and drawbacks of the operators.

4.2 Requirement Analysis

A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose or in other word, requirements are description about what system can do and their constraint.

To determine the system requirement, first the developer must do an interaction between customers to elicit the requirements, by asking questions, demonstrating similar or even developing prototypes of all or part of the proposed system. Next, capturing those requirements in a document or database. The requirements are written first so that the developers and their customers agree on what the system should do. Then, the requirements are often rewritten, usually in a more mathematical representation, so that the designers can transform the

requirements into a good system design. A verification step ensures that the requirements are complete, correct and consistent and a validation step makes sure that the developer described what the customer intends to see in the final product.

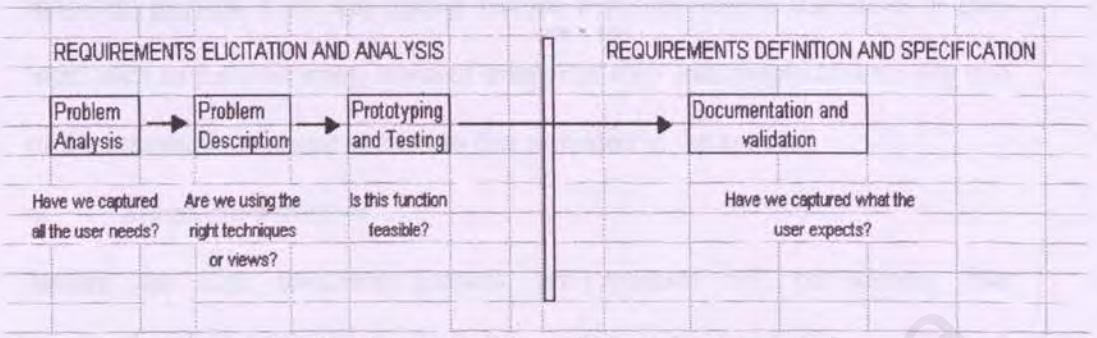


Figure 4.0 The process of determining requirements.

The requirement analysis can never be skipped but it can integrate with the problem analysis into a single phase. New system will always be evaluated on whether or not they fulfilled the system's objectives and requirements, regardless of how impressive or complex the technological solution might be. It can be divided into functional requirements and non-functional requirements.

4.2 1 Functional Requirements

A functional requirement is a description of activities and services a system must provide. Functional requirements are frequently identified in terms of inputs, outputs, processes and stored data that are needed to satisfy the system improvement objectives.

Besides that, functional requirements are functions, features and subsystems that must be included in an information system to satisfy the system needs and be accepted to the users. The absence of the functional requirements will make the whole system incomplete. The following are the functional requirements of image edge detection:

i. Upload images

The user needs to upload image first before the system running the edge detection process. User can upload images from any source they have or they want such as from Internet, scanned images or they just simply choose any one of the images from image collection that provided in the system.

ii. Image Pre-processor

Before the edge detection process, pre-processor will be running first automatically in order to makes the edge are detected precisely.

iii. Detect edge

This is the main function in this system and that is the reason why this system built up; to detect edge. The system detects an edge based on the value of threshold. This value is determined by user due on the level of detection that they want, but for those user who are not familiar with edge detection or the using of threshold value, they can ignore that value because the system will still running the detection process based on the default value that was programmed by system developer.

iv. Preview image

It is important to know how's the image looks like after detection process generated. If the result does not meet the user needs, they will be able to undo the process and do it again by trying key in another threshold value till they satisfied with the result. This approach is usually referred as "try and error" method.

v. Print image

This function will be optional to user, they can choose either to print out their edge image or they just simply save the image in their own output device such as floppy disk or compact disk.

4.2.2 Non-functional Requirements

A non-functional requirement is a description of other features, characteristics, attributes of the system as well as constraints that way limit the boundaries of a satisfactory system. It is an essential definition of requirements which how a system must operate.

While these criteria are not actual actions taken by the system, they are further restrictions on what the system must be able to handle. The non-functional requirements that have to embedded into image edge detection include the following aspects:

i. User interface

A standard user interface refers to the consistency usage of colour, font size, position of text, graphics and also functional menus. It helps users get the result they need in and out of the system by addressing user interface objectives, which also results in being a user-friendly interface to user.

ii. User-friendliness

Building a good flow of navigation can help users to be able to understand with little effort or at ease about what is going on as users navigate through the system.

iii. Reliability

A system is considered reliable if it does not produce dangerous or costly failures when used in ways that the designer might not expect it to be used and the system must be able to handle these situations.

iv. Accuracy

Accuracy refers to the precision of the edge detection process provided. It provides various accuracy measures to maintain the accuracy of detection process.

v. Maintainability

Maintainability can be defined as the ease with which software can be understood, corrected, adapted or enhanced in the future.

vi. Efficiency

Efficiency in computer technology means a procedure that able to called or accessed in many times and it will give that same outputs, not the different outputs.

4.3 Development tools analysis

4.3.1 Hardware specifications:

Processor : Pentium II 166 Mhz and above.

Memory : 56.0 Mbytes of RAM.

Hard disk : at least 4.75 Gigabytes.

Input device : Mouse, keyboard, scanner, floppy drive and disk drive.

Output device : Printer, monitor.

Graphics adapter : 8-bit (for 256-simultaneous colours)

4.3.2 Software Specifications:

Operation system : Microsoft Windows 95, Windows 98, Windows Millennium Edition (ME), Windows NT 4.0, or Windows 2000.

Development Tools : Visual Basic.

Programming Language : MATLAB 6.1.

4.4 Chapter Summary

Before the system requirements are identified, the possibility problems have to be analyzing first so that the requirement can be determined based on the problems. Requirement analysis was done in this chapter as well. The requirements are categorized into a few sub-functions that are upload image, pre-processor, detect edge, preview and print image. These will be the functions that this system provides. Whereas, non-functional defines the description of other features and system constraints that define a satisfactory system.

This chapter also concludes the hardware and software requirements that will be using in the development phase.

5.1 Introduction

System design is defined as those tasks that focus on the specification of a detailed computer-based solution, on the technical or implementation. It is also called physical design. The purpose of system design is to determine how to construct the information system to best satisfy the document requirements. Meanwhile, the goal of the system design is to design the detection system that is effective, reliable and maintainable. In general, the design processes begin with output progresses to input, then data storage and system processing. The design phase are focused on architectural or process design, and user interface design.

5.2 Architectural or process design

In architectural design, the system is decomposed to sub-system that provide almost all related set of services. This is the initial design of identifying sub-systems, establishing a framework for sub-system control and communication. Besides, the sub-systems that make up the whole system and their relationship are identified and documented. The following designs are:

5.2.1 Context Diagram of Image Edge Detection System

The diagram in figure 5.0 shows a general interaction or communication between system and its environment, identified all the entities related to system and what's the system's input and output.

5.2.2 Structure Chart of Image Edge Detection System

In this chart that shows in figure 5.1, the system's functions are specified where all the function that be able to run by user are listed down.

5.2.3 Structure Chart of Image Edge Detection System

In this chart (figure 5.2), the system's functions are specified where all the function that be able to run by user are listed down.

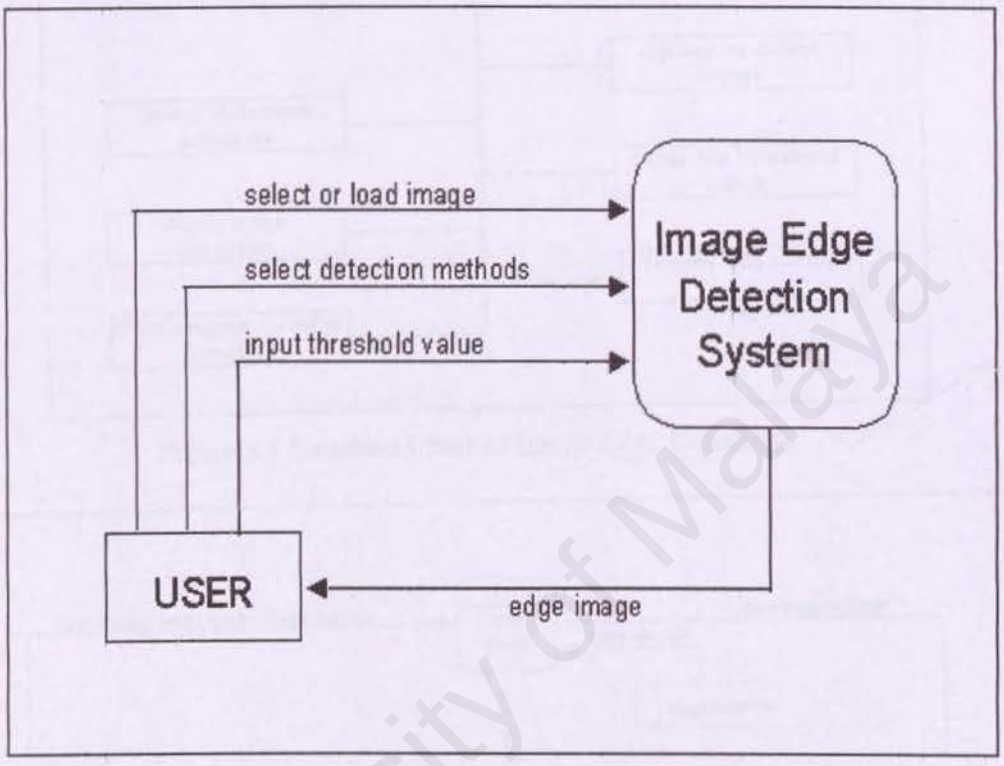


Figure 5.0 Context Diagram of Image Edge Detection System

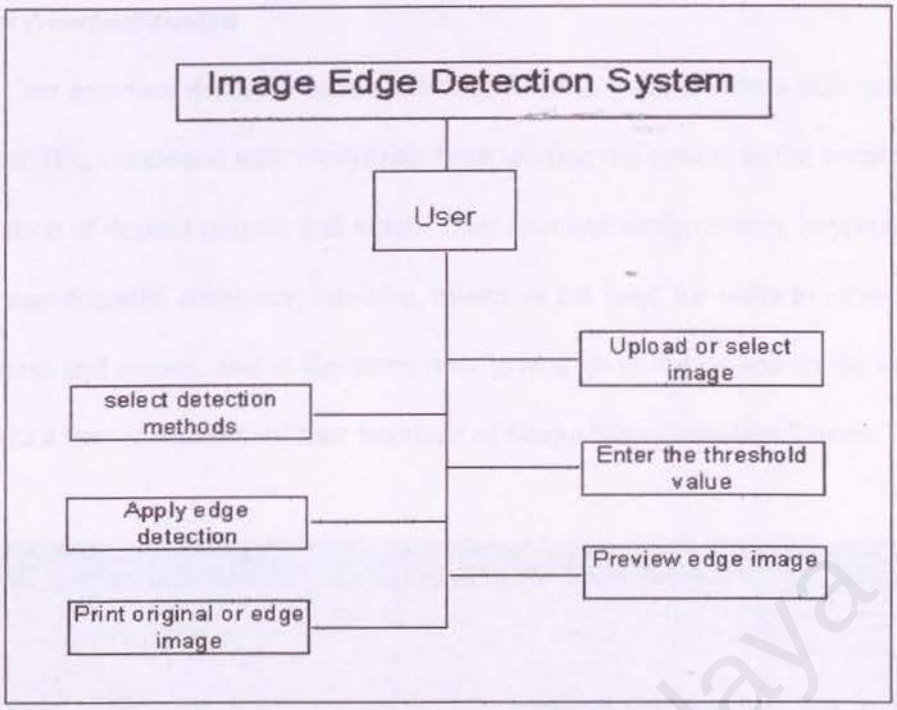


Figure 5.1 Structure Chart of Image Edge Detection

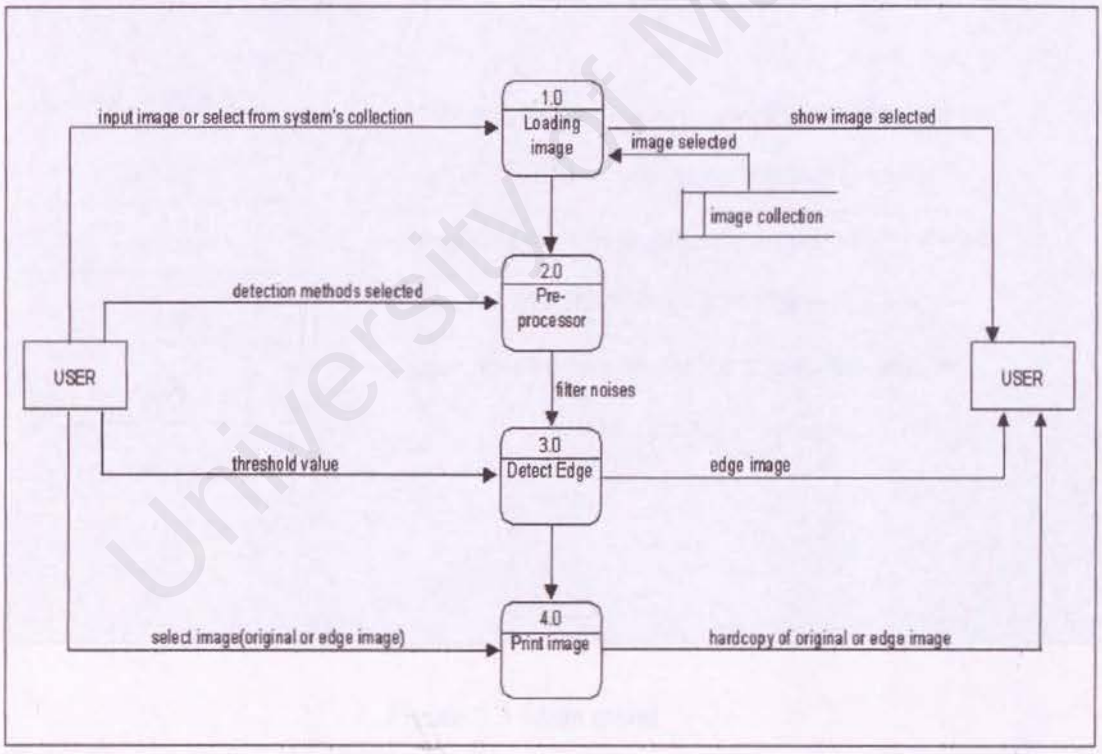


Figure 5.2 Data Flow Diagram of Image Edge Detection

5.3 User Interface Design

User interface design is concerned with the dialogue between a user and the computer. It is concerned with everything from starting the system to the eventually presentation of desired outputs and inputs. User interface design is very important to offer a user-friendly, reliability, intuitive, minimize the need for users to memorize the process and events, and at the same time give a good impression to the users. Below are a few screenshots of user interface of Image Edge Detection System.

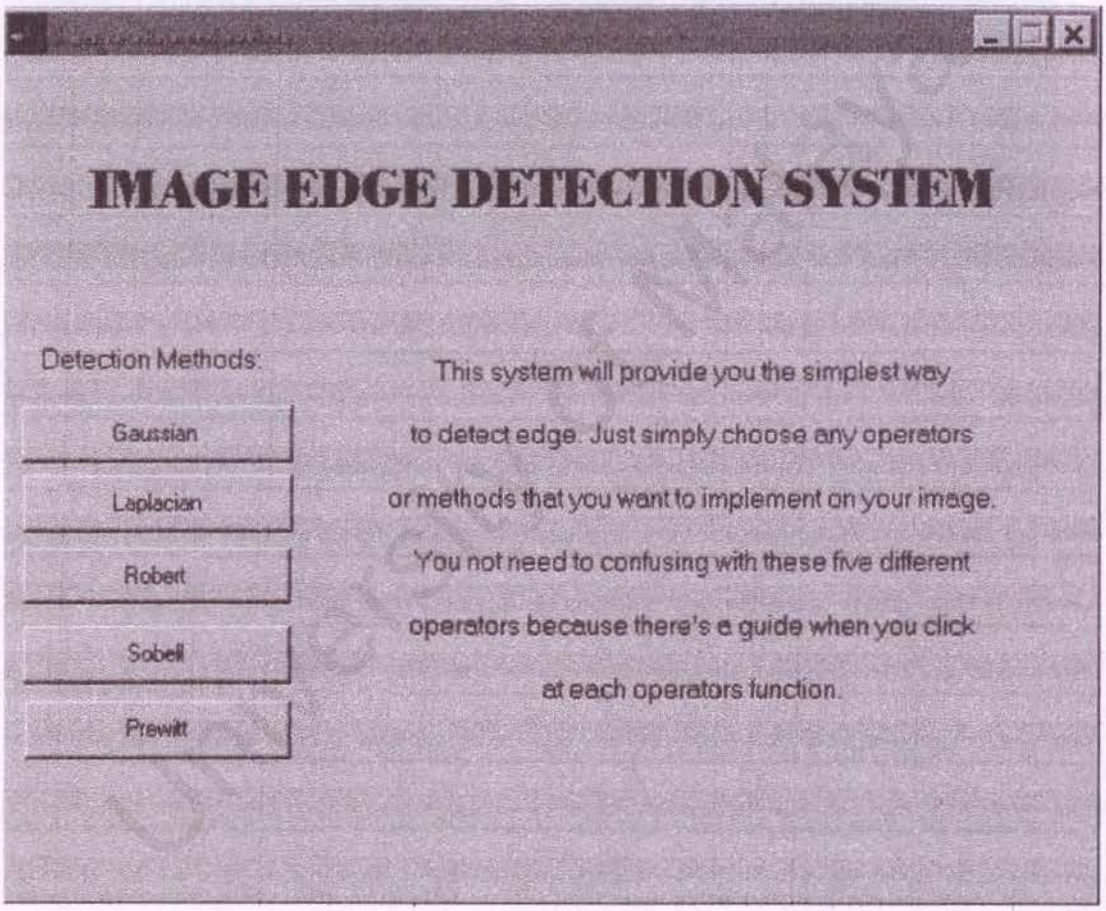


Figure 5.3 Main menu

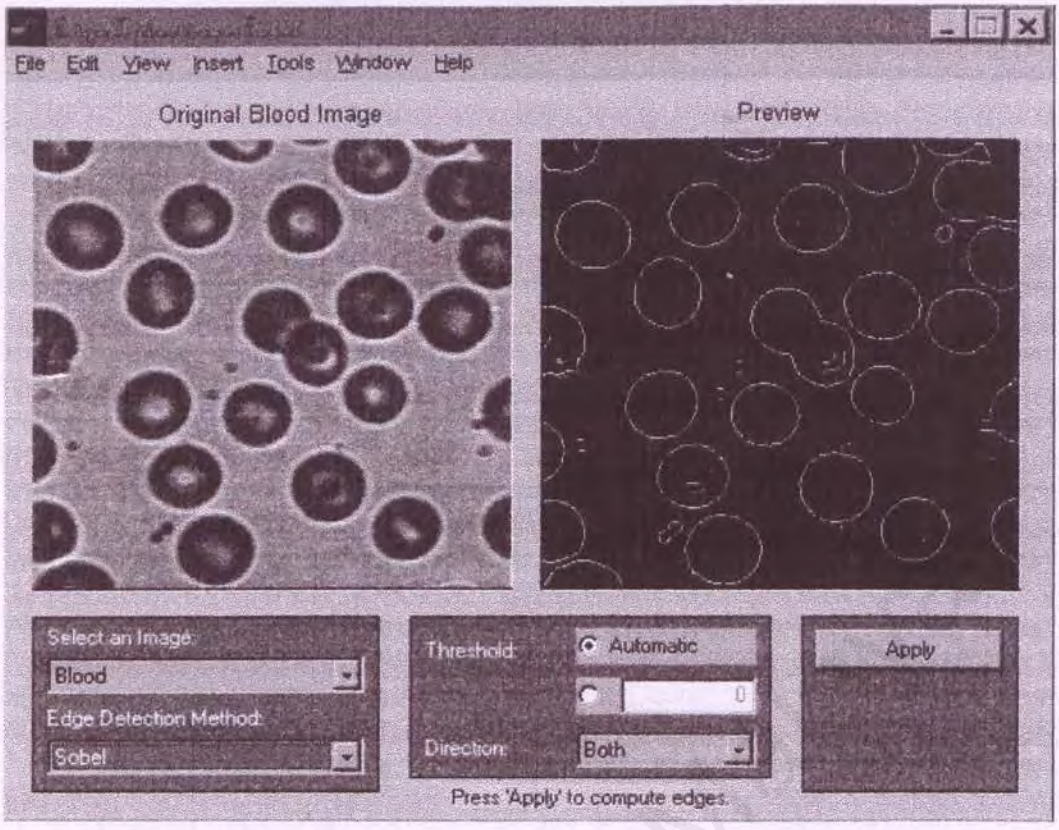


Figure 5.4 Detection page

5.4 Chapter Summary

System design is defined as those tasks that focus on the specification of a detailed computer-based solution, on the technical or implementation. There are kinds of design defined in this chapter, which are architectural design and user interface design.

Architectural design partitions the system into subsystems and functions. It shows the overall system to be developed. It also provides the beginnings an outline for drawing the data flow diagrams. User interface design is concerned with the interaction medium between a user and the computer. Too complicated user interfaces will prevent users from using the system. Thus, the user interface should be user friendliness. A few screenshot of user interface of Image Edge Detection System are captured in this chapter.

6.1 Introduction

The process of assuring that the information system is operational and then allowing users to take over its operation is called system implementation. System implementation is further defined as the construction of the new system and the delivery of that system into production in a day-today operation. It involves coding step that translates a detailed design representation of software into a program language realization. System implementation implements the various components of the system based on the collected requirements, where the design is translated into a machine-readable form.

During implementation, all functionality planned in design phased is checked. It should be able to process the correct data and produce accurate information to end-users. Any problem or malfunction occurred id revised carefully and fixed accordingly.

6.2 System development

The development environment is crucial for the completeness and successfulness of any computer system. Development environment plays a major role in determining the speed of developing the system. During development, the weaknesses will be noticed and improved; while the errors found will be removed.

Using suitable hardware and software will help to speed up system development. Thus, the hardware and software are carefully considered to facilitate the development of the Image Edge Detection System. System development consists the used of methodology chosen, forms coding, development tools. The details are illustrated as below:

6.2.1 Development Tools

6.2.1.1 Hardware Requirements

The following hardware specifications are required to develop Image Edge Detection System:

- Processor : Pentium II 166 Mhz and above.
- Memory : 56.0 Mbytes of RAM.
- Hard disk : at least 4.75 Gigabytes.
- Input device : Mouse, keyboard, scanner, floppy drive and disk drive.
- Output device : Printer, monitor.

6.2.1.2 Software Requirements

The following software specifications have been used to develop Image Edge Detection system:

- Operating system : Microsoft Windows 2000.
- Development Tools : Visual c++
- Programming Language : C

6.2.2 Methodology

This project is developed using the waterfall approach. The development of this project will consist of five stages, which are requirement, design, coding, testing and operation. The system is design using logical flow and it allows the estimation of the milestones. Each stage must be completed before proceed to the next stage to ensure that the system is built according to the requirements and specifications.

6.2.3 System Coding

System coding is a set of instruction written in order to enable the code to be executed and perform the required functionality. A good and well-managed program coding will enhance the readability of the whole program. In addition, it provides an easy understanding to the program flow especially for those programs with high degree of complexity.

6.2.3.1 Coding Approach

Some of the approaches used in the coding development are listed as below:

6.2.3.1.1 Readability

Code document is important to ease the readability of a system. It begins with the selection of identifier (such as variables and labels) names and continues with the composition and organizing the whole program.

6.2.3.1.2 Naming Technique

This is good and meaningful technique of variables, controls and modules that provide easy identification for the program. The naming convention is created with the consistency and standardization in coding.

6.2.3.1.3 Internal Documentation

This provides a clear guideline to developers and readers about the function of a particular source code in the program. Therefore, comments provide the developer with the means of communication with other readers of the source code. The

statement of the module and descriptive comments are embedded within the body of the source code is used to describe the processing function.

6.2.3.1.4 Modularity

The main purpose of modularity is to reduce complexity of system and to facilitate the developer to implement the system by encouraging parallel development of different parts of the system. With the approach of modularity, developer can implement all modules at the same time and does not have to wait for a particular module to complete before going into another module.

6.2.3.2 Coding Style

Coding style is an important component of the source code and it determines the intelligibility of a program. An easy to read source code makes the system easier to be maintained and enhanced in future. Listed below are some of the coding styles used during the coding phase of this project:

- Selection of meaningful identifier names (variables, forms, labels, images and pictures).
- Description and an appropriate comment written in the source code to make it easier for the readers to understand the source code.
- Indentation of codes will increase the readability of the program and for a neater look.
- Meaningful and understandable function and method declarations.
- Keep all complex statement as simple as possible to avoid confusion.

6.2.4 System Coding Tool – Visual C++ 6.0

Visual C++ is a powerful and complex tool for building 32-bit applications for Window 95 and Windows NT. These applications are much larger and more complex than their predecessors for 16-bit Windows or older programs that did not use a graphical user interface. Yet, as program size and complexity has increased, programmer effort has decreased, at least for programmers who are using the right tools.

Visual C++ is one of the right tools. With its code-generating wizards, it can produce the shell of a working Windows application in seconds. The class library included with Visual C++, the Microsoft Foundation Classes (MFC), has become the industry standard for Windows software development in a variety of C++ compilers. The visual editing tools make layout of menus and dialogs a snap.

Actually, the development tool that was chosen in requirement analysis to implement this system is Visual Basic 6.0 and MATLAB 6.1, but because of some reason, the development tool was changed. For supporting MATLAB to be a stand-alone program, it is easy to used Visual C++ rather than Visual Basic because in Visual C++ there is an add-in function which is simply convert MATLAB files (M-file) to extension program (*.exe). The MATLAB add-in adds a MATLAB Wizard to the New Project dialog box and a toolbar to the Developer Studio user interface. The actions associated with the toolbar buttons are: adding a file to the project; opening the matrix viewer while debugging; packaging a completed program for redistribution; and viewing this help file.

Figure 6.0 : Matlab add-In Toolbar

That is the main reason why the development has been changed. But the big problem with Visual C++ is it cannot support any types of image from M-files. Though the M-files executed successfully in Visual C++, but the compiler failed to read the image. To solve this problem, all the coding was rewrite in C language.

6.2.5 Coding Concept

Generally, the concept of edge detection that used in this system is scan every pixel in the image one by one, then subtract the pixel with the adjacent pixels (right, bottom & bottom-right pixels). The system subtracts it to find the different between the scanned pixel and it's adjacent pixel. Then find which value is the biggest one (the most obvious different value) and put it into the scanned pixel as the edge. Because the scanning moves from left to right and from top to bottom, it only scan the pixel that are in front of it ie:right, bottom & bottom-right pixels. Bellow are the code that execute the comparison process:

```
//get center color(currently scanned pixel)
cc = getPixel1(img, x, y, channel);
//compare center color with right side color
if(x<img->sizeX-1)//check if we are not out of the image boundary
//compare center color with bottom color
if(y<img->sizeY-1)//check if we are not out of the image boundary
```

6.3 Chapter Summary

This chapter describes the implementation of the system being developed. It begins with the introduction to the system implementation. System implementation implements the various components of the system based on the collected requirements, where the design is translated into a machine-readable form.

Then, the chapter describes the development environment of the system. The system development includes of hardware and software requirements, methodology chosen, system coding and development tools and coding concept. A sample code is included in Appendices to show the coding environment.

7.1 Introduction

System testing is a critical element of software quality assurance. It is required to ensure that the system is developed according to its specifications and in line with the users requirements and expectations. Testing is not the first place where faults finding take place but it is focused on finding faults and errors. There are many ways to increase the effectiveness and efficiency of the testing efforts, which will be discussed later in this chapter. Failure of a system can be the results of several reasons:

- The specification may be wrong of have missing requirement and do not state exactly what the customer needs.
- The specification may contain a requirement that is impossible to implement by the given predescribed hardware, software and resources.
- The system design phase may contain fault or error that carried forward to the implementation phase.
- The program code may be wrong. Perhaps the algorithm is implemented improperly.

Faults identification is the process of determining what fault causes the failure of the system. The fault correction or removal is the process of making changes to the system so that the fault can be removed.

7.2 Objective of Testing

The reason and objectives for performing extensive tests during the design and development of the system are as followed:

- Achieve high quality assurance such as completeness, accuracy, reliability and maintainability of the software program and its documentation.
- Ensure that the system can perform its functions as expected.
- Reduce cost in maintaining the system.
- A method for detection and removal errors.

7.3 Testing Technique

The component of a system will be allowed to manipulate the data, and the output will be observed. Thus, a wide range of inputs and conditions are chosen in order to test that particular component. A test point/test case is a particular choice of input data to be used in testing program.

7.3.1 White Box Testing

White box testing is a testing case design method that uses the control structure of the procedural design to derive test cases. By using white box testing methods, the test cases with the following characteristics can be driven:

- Exercise all logical decision on their true or false side.
- Exercise all loops at their boundaries and within their operational bounds.
- Exercise internal data structure to ensure their validity.
- Guarantee that all independent paths within a module have been exercised at least once.

7.3.2 Black Box Testing

Black box testing focuses on the functionality requirements of the system. It enables the developer to derive sets of inputs condition that will fully exercise all functional requirements for an application. Black box testing was not used as an alternative to white box testing technique rather than this technique is used as a complementary approach that is likely to uncover a different class of errors. Black box testing attempts t find errors in the following categories:

Incorrect or missing functions

- Interface errors
- Errors in data structures or external data access
- Performance access
- Initialization and termination errors.

It also tests the functionality of the system in an ad hoc basis without knowing the logic structure of the code. Input is provided and output is verified manually to check for accuracy.

7.4 Testing Strategy

A strategy to test this system is actually a series of steps that are implemented sequentially. After a program is completely coded, it will be tested under unit testing. Module testing will start when all the programs under a particular module have been completely coded and tested under unit testing. The integration testing is to recover errors associated with interfacing when integrating all the modules.

7.4.1 Unit Testing

Unit testing focuses on verification effort on the smallest component of the system design. Each component is treated as a standalone entity and tested individually to ensure that they operate correctly. The unit test is usually white-box oriented and the step can be conducted in parallel for multiple components.

The test that occurs as part of unit tests is illustrated schematically in Figure 6.1. The module interface is tested to ensure that information properly flows into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. Boundary conditions are tested to ensure that the module operate properly at boundaries established to limit or restrict processing. All independent paths (basis path) through the executed at least once. Finally, all error –handling paths are tested.

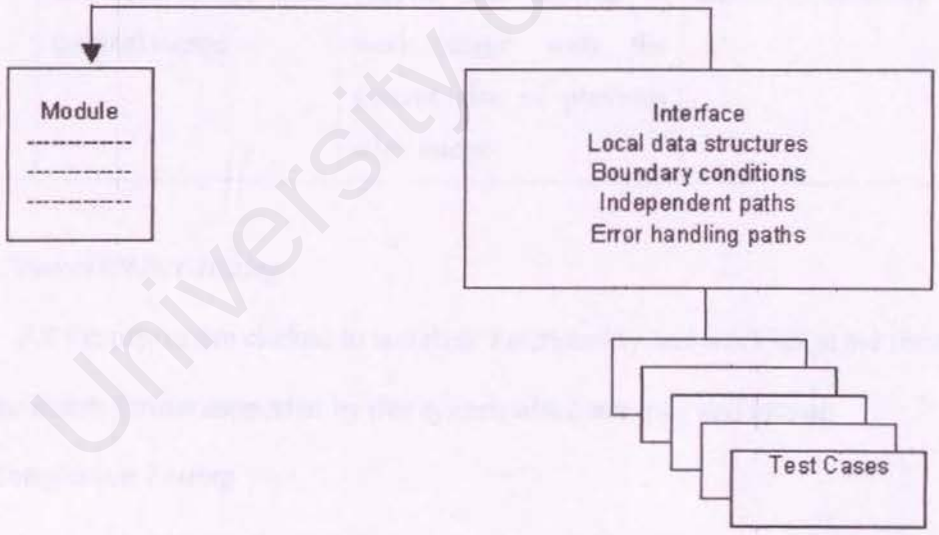


Figure 7.0 : Unit Testing

7.4.1.1 Unit Testing Example

Table below shows the test cases for unit testing on the edge detection program.

Table 7.0 : Unit testing example

Step	Test Procedure	Expected Outcome	Test Result Analyzing
1	Load new image either in jpeg or bitmap format to the system	The image is loaded and showed in work stage	The image displayed successfully
2	Click edge detection function to detect edge	The image is processed and the edge image shown.	The edge image shown successfully
3	Click Reload Image function to get the original image	The original image loaded and showed in work stage with the current size of previous edge image	The original image shown successfully

7.4.2 Control Object Testing

All the menus are clicked to test their functionality and work stage are tested with the image format supported by this system which are jpeg and bitmap.

7.4.3 Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. Testing a specific feature together with other newly developed feature is known as integration testing. In other words, when the individual components are

working correctly and meet the objectives, these components are combined into a working system.

In this system, a bottom-up approach has been used. Bottom-up integration testing begins construction and testing with modules at the lowest levels of the system and then moving upward to the modules at the higher levels of the system. Regression testing is the re-execution of some subset of tests that already been conducted to ensure that changes have not unintended side effects. It is the activity that helps to ensure that changes (due to testing or other reason) do not introduce unintended behavior or additional errors.

7.4.4 System testing

System testing is a series of different tests designed to fully exercise the software system to uncover its limitations and measure its capabilities. The objective is to test an integrated system and verify that it meets specified requirements. Although each test in this system has a difference, all work to verify that the system elements have been properly integrated and perform allocated functions.

7.5 Chapter Summary

This chapter is all about testing. These testing include unit testing, control object testing, integration testing and system testing.

Image edge detection system has been tested and debugged effectively to achieve the objectives of the system. Through all the testing phases, it is easier to ensure the system's qualities and strengths. Debugging and fixing of the program can be done. The limitations of the system's functionalities can be found and improved.

As a conclusion, testing phase is a very important phase in image edge detection system and it must be done repeatedly and carefully to assure good software quality.

8.1 Introduction

In the process of developing a system, various problems have been identified which some have been solved and some of them are yet to be discovered and overcome. These problems were solved through research and reference books. Besides, a lot of system analysis has been done on technological and programming concepts to grasp the concept of Internet programming.

After all the designing and developing as well as implementing of image edge detection system, the end product of the project is brought up for evaluation. Image edge detection system was evaluated to identify the strengths and the limitations of the system. Besides, proposal and recommendations are made for the future enhancements of the system.

8.2 Problems Encountered and solutions

- *Difficulty in choosing a suitable development tools*

  There are too many software tools that are available for developing image edge detection system. It is difficult to choose the most suitable development tools from a wide variety of choices. Choosing a suitable technology and tools was a critical process as all tools possesses their own strengths and weakness. Besides, the availability of a technology, hardware and supporting software to support, its learning curve, compatibility with the existence operating system and technologies are also the major consideration.

In order to solve the problem, seeking advices and views from project supervisor, course-mates and even seniors engaging in similar project were carried out. Furthermore, a great deal of reading and research from many resources like books and Internet regarding the problems helped to solve the problem and choose the suitable tools were done before any decision was made.

- *Lack of knowledge in Visual C++*

  Since there was no prior knowledge of programming in Visual C++ 6.0, there was an uncertainly on how to organize the codes. These new programming languages and concepts were never taught before and to implement such as application requires a fair grasp of the languages. These programming approaches seem to be totally different from the traditional programming languages. Although it really cause a lot of time to learn the new technology, but choosing to program in Visual C++ proved to be a wise move. Most of the problems faced were manageable through browsing the Internet for related materials and referring to the help function provided in the software. Discussion with friends especially course-mate using the same technology was a great help. A more efficient method was through trail and error during the coding phase.

- *Difficulties in defining the flow logic of the system*

  This system is only based on the information gathered from reference books and Internet; as a result, the flow of the system is very hard to define. This system is merely following the flow logic based on my understanding of

the requirements and the important of ease of use. The image processing knowledge that gained from the lecture session also useful to design the flow.

- *No multi-function in the system*

  Based on the system title, which is 'Image Edge Detection System', of course the system will only detect the edge of image. If MATLAB 6.0 were used in building all the codes, there will have a types of detection like Sobell, Prewitt, Robert and Gaussian, unfortunately Visual C++ cannot read image from M-files. This problem solved by rewrite the source code based on new algorithm in C language.

  The effect function was added in this system in order to make it more fun and useful rather than just detect edges. The effects included are blur, blur more negative, line art and diffuse.

8.3 System Strengths

- *Simple, user-friendly and easy to use*

  The design of the interface of this system is based on Visual C++ wizard. It is design to be as user-friendly as this system is relatively easy to learn and use. All the menus used to ease the user explore and try this system by themselves. An action is just a click away and the user just needs minimal knowledge of mouse and keyboard to use this system.

- *Support colour image*

  Although MATLAB has a variety types of edge detection, the detection only process the .tiff image, that is black and white or grayscale image. This

system support two types of image which are jpeg and bitmap format, it does not matter either the image is in two or three dimensional, colourful or black and white.

8.4 System Constraints

- *No print function*

  No print function in this system. The user needs to save the image processed first in the IMG file that created in this system before they transfer the IMG file to their local disk and print the image.

- *No report generation*

  In this system, only the processed image will be shown. No other reports or information about the threshold value, edge intensity or other related information.

- *Not so effective*

  The original image should be display beside image processed so that the user can compare the result with the original version. The image processed must be save in IMG format that only readable in the system, that's mean the image cannot be display any platform except the system.

8.5 Future Enhancements

System development is a dynamic process and changes must be expected. Due to the limited resources that the author have, especially time, this cause the author miss or overlooks certain aspect of the system. However, after the development system has been completed and valuable advices and suggestions from my project supervisor and moderator, the author have identified certain important

aspects that can add on for future enhancement. The additional features that can be implementing in future are as followed:

- *Report Generation*

  In current edge detection system, there is no report about the edge. This is certainly not enough for the user especially for those who used this system in learning image processing. Thus, in the future, more reports about the edge detected should be generated. These reports might include the edge intensity, the detection persistence and other information related.

- *Support variety of image format*

  Since there are many types of image now, the system should be able to support all the types to make it widely use.

- *Add more detection types*

  To make this system more useful in the future, a number of detection types should be included. The user can choose any method they want and compare the result or the difference between each type.

- *Add more useful function*

The current system does not allowed user to save image processed in other format besides IMG. In the future, the system should add a function that allow user to save their image in any format they want and a print function to make their image available in hard copy version.

8.6 Knowledge and Experience Gained

Towards the accomplishment of the Image Edge Detection System, from the beginning to the end of the development and final documentation, a number of problems and difficulties are encountered. However, the solutions to these problems and difficulties have brought numerous valuable knowledge and experience. The benefits and knowledge gained are as followed:

- *The importance of all phase in SDLC*

  System analysis is an important phase in the System Development Life Cycle (SDLC). This phase is capturing user requirements and the goal of the system. If this phase is wrong defined, it will cause faulty to the system development and later progress. With a complete and thorough system analysis, the system that is developed will fulfill all the requirements and achieve it goals.

  System testing is also an important and critical phase in SDLC. There is no application that is free of error in this world. However, with the procedures in the system-testing phase, errors and faults in the system can be minimized.

- *Development tools knowledge*

  This project is developed using Microsoft Visual C++ 6.0 (VC++). VC++ is a very suitable development tools for developing Windows environment application. It is easy to use and provides the simple layout and many examples to follow and also improved the author knowledge in C language.

8.7 Chapter Summary

Evaluation of system is indeed to ensure its objectives and intended functions have been achieved. This chapter covers all the aspects of the evaluating application software.

The successful development of the system at the present is the first step towards the future expansion of the system. The problem encountered and experience gained during the development phases should be helpful in future efforts.

Besides, this chapter also summarizes the system strengths, system constraints and future enhancements that can be added. The future enhancements will equip the system towards more capabilities of doing its daily operations and activities.

Image Edge Detection System is one of the images processing system to detect edge of digital image. Besides, it also provides some types of effect that allowed user to apply on their image. However, the system will become more complete and capable of performing more tasks when the enhancement and the new features are added on in the near future.

In the process of developing this system, invaluable insight was gained into complexities and intricacies of programming. The application of Software Engineering principles, fundamentals and additional knowledge in programming languages, skills coding writing and others all added up to contribute to the success of developing this system. Adhere to development schedule is crucial in determining that a system will be completed in time. The experience gathered in this project will definitely provide a solid foundation in the system development in the future.

With target goals and objectives in mind even before the development takes place, makes the development process more systematic. Sometimes, conflicts in real world situation and programming tools capabilities make the programming difficult. However, as an overall review, this project has achieved and fulfilled the objectives though its not meets the requirements determined during the analysis phase entirely.

New User Interface

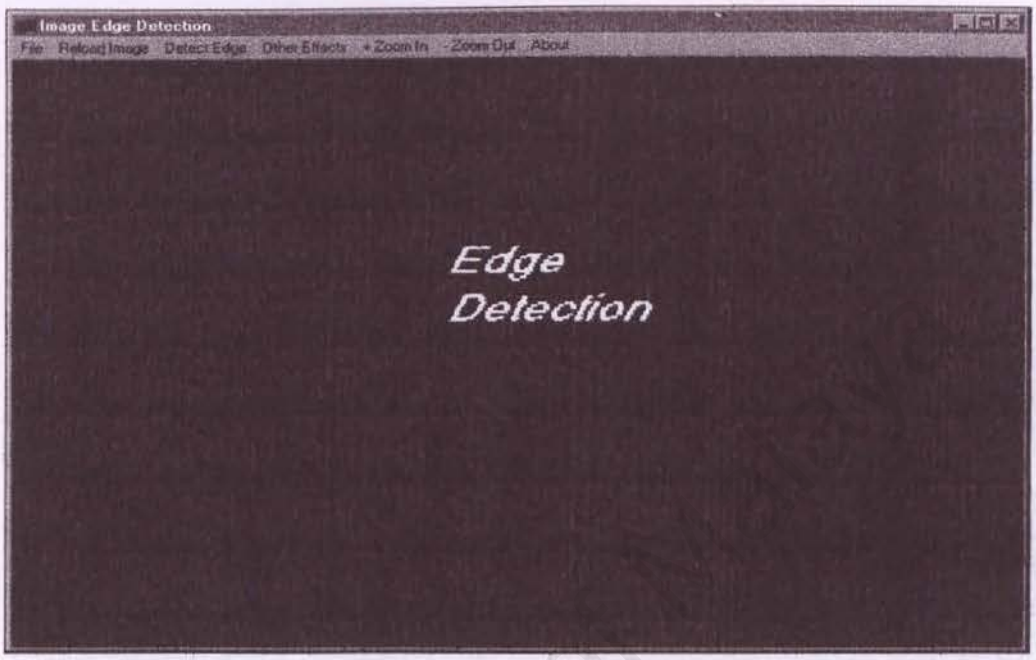


Figure 1.0 : Main display

Figure 2.0 : The list of main menu

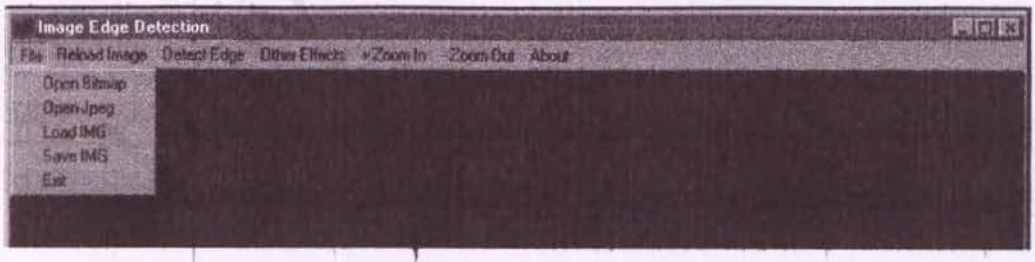


Figure 3.0 : The list of function in File menu

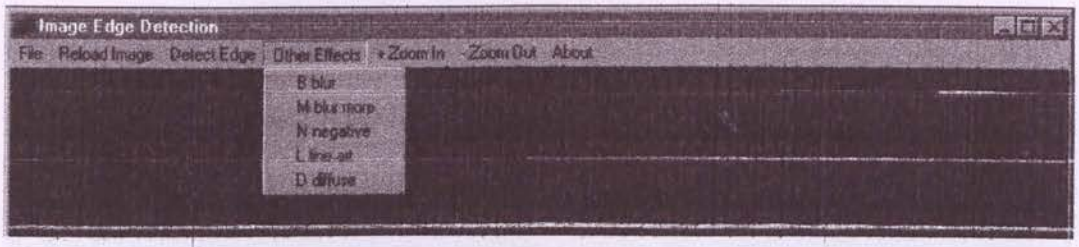


Figure 4.0 : The list of function in Other Effect menu

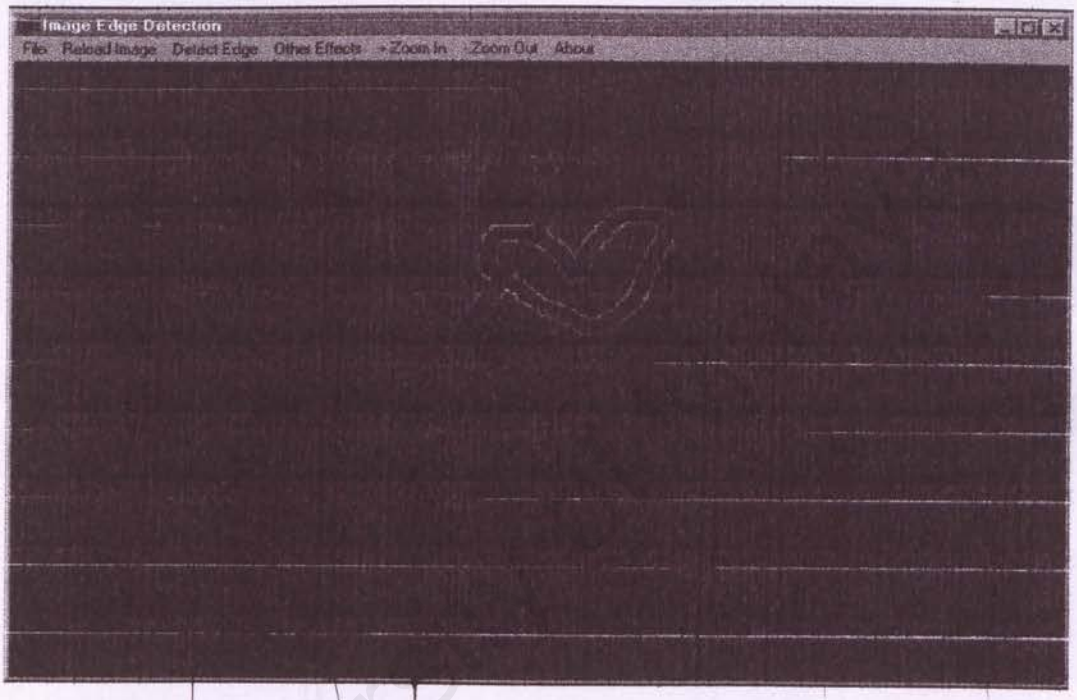
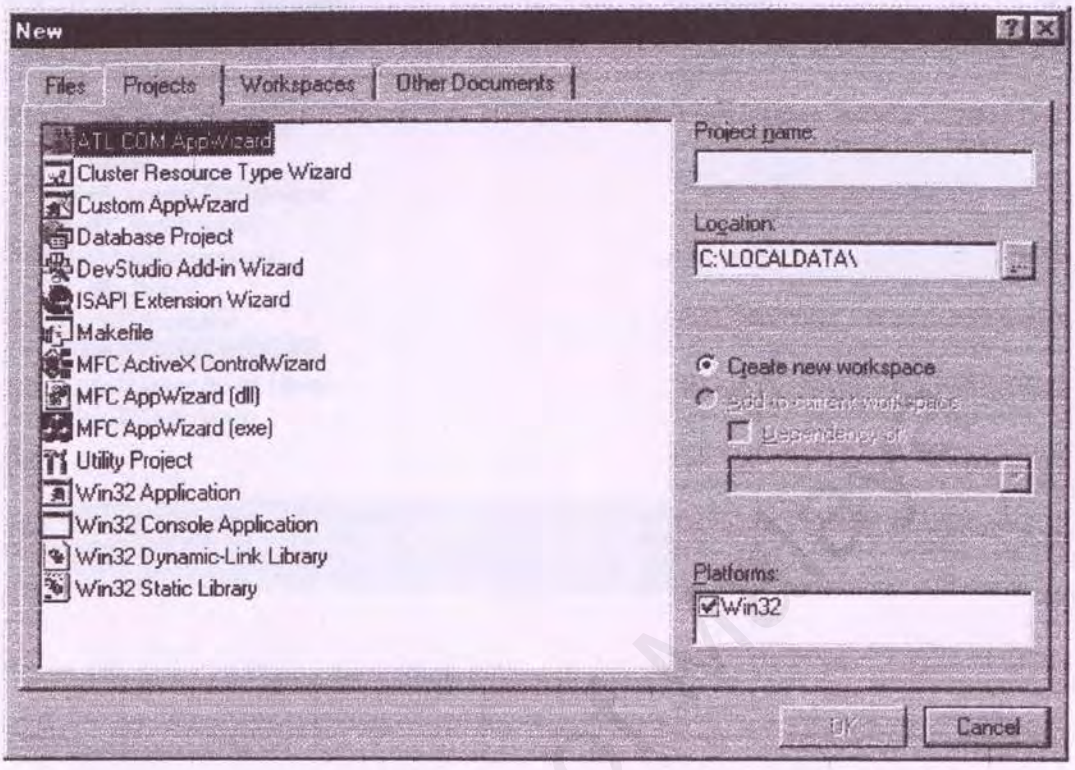
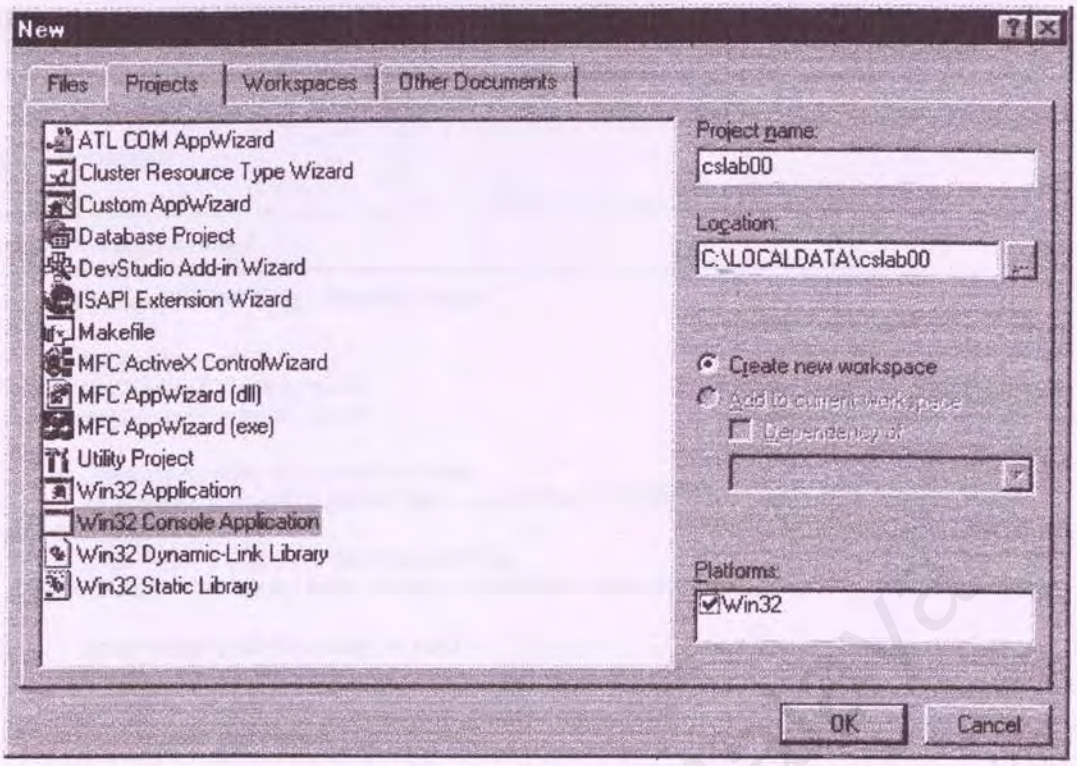


Figure 5.0 Example of edge image

Select File|New from the menu. The "New" dialog box will appear.



From the list at the left, select "Win32 Console Application." In the Location text box on the right, type the name of the directory you wish to create this project's directory in. Alternately, you may select the base directory from a dialog box by selecting the "..." button to the right of the text box. Type in a name for the project in the textbox "Project Name". As you type, notice that the project's name is added to the directory in the directory text box. VC++ will automatically create a new directory with the same name as your project in the base directory.



Select OK to set up the project. Click OK on the next two screens.

Creating source file

Visual C++ will return to the startup screen, but will add two tabs ("Classview" and "FileView") to the leftmost window. You will need to add any source files needed for your project. Select "Project|Add to Project|New" from the menu. This will bring up the "New" dialog box again, only this time the "Files" tab will be selected. Select "C++ Source File" from the list at the left. Enter a name for the file in the "File name" text box. Press OK to create the blank file. Now you can enter the code for a your program either in C language or C++.

Sample Code : Image processing

```
#include "image.h"

//=========================================================================
//copy from img1 to img 2
//=========================================================================
bool copyImage(IMAGE *img1, IMAGE *img2)
{
	//copy image size
	img2->sizeX = img1->sizeX;
	img2->sizeY = img1->sizeY;

	//allocate memory for destination image
	int sizeimg = sizeof(GLubyte)*img1->sizeX*img1->sizeY*3;//<- times by 3 because image
has 3 channel (RGB)
	img2->data = (GLubyte*)malloc(sizeimg);
	if(!img2->data)return false; //if unable to allocate memory, return

	//copy image pixel from img1 to img2
	for(int i=0; i<sizeimg; i++)
	{
		img2->data[i] = img1->data[i];
	}

	return true;
}
//=========================================================================
//free memory of image
//=========================================================================
void destroyImage(IMAGE *img)
{
	if(img->data)
	free(img->data);
}
//=========================================================================
//set color of entire pixel in an image
//=========================================================================
void setColor(IMAGE *img, GLubyte R, GLubyte G, GLubyte B)
{
	//set every pixel to R, G & B
	for(int x=0; x<img->sizeX; x++)
		for(int y=0; y<img->sizeY; y++)
		{
			putPixel1(img, x, y, _RED,	R);
			putPixel1(img, x, y, _GREEN, G);
			putPixel1(img, x, y, _BLUE,	B);
		}
}
//=========================================================================
//process images to find edges
//=========================================================================
/*
what this code is doing is scan every pixel in the image one by one,
then subtract the pixel with the adjacent pixels(right, bottom & bottom-right
pixels). We subtract it to find the different between the scanned pixel and
it's adjacent pixel. Then we find which value is the biggest one(the most obvious
```

different value) and put it into the scanned pixel as the edge.

Because the scanning moves from left to right and from top to bottom, it only scan the pixel that are infront of it ie:right, bottom & bottom-right pixels.
*/
//this macro will return the biggest number between x, y & z
#define biggest(x,y,z) x>y? (x>z? x:z):(y>z? y:z)

```
void Edge(IMAGE *img)
{
        int             x, y,
                        xplus1, yplus1,
                        channel;

        GLubyte cc, rc, bc, rbc;

        //check every pixel in the image
        for(x=0; x<img->sizeX; x++)//scan in x axis
        {
                for(y=0; y<img->sizeY; y++)//scan in y axis
                {
                        //process all 3 channels(RGB) one by one
                        for(channel=0; channel<3; channel++)
                        {

                                xplus1 = x+1; //to get the right side pixel
                                yplus1 = y+1; //to get the bottom side pixel

                                //get center color(currently scanned pixel)
                                cc = getPixel1(img, x, y, channel);

                                //compare center color with right side color
                                if(x<img->sizeX-1)//check if we are not out of the image
boundary

                                        //get cc minus rightside color
                                        rc = abs(cc - getPixel1(img, xplus1, y, channel));

                                //if we are out of boundary, set it to 0
                                else rc = 0;

                                //compare center color with bottom color
                                if(y<img->sizeY-1)//check if we are not out of the image
boundary

                                        //get cc minus bottom side color
                                        bc = abs(cc - getPixel1(img, x, yplus1, channel));

                                //if we are out of boundary, set it to 0
                                else bc = 0;

                                //compare center color with right-bottom color
                                if(x<img->sizeX-1 && y<img->sizeY-1)//check if we are not out
of the image boundary

                                        //get cc minus bottom-right side color
                                        rbc = abs(cc - getPixel1(img, xplus1, yplus1, channel));

                                //if we are out of boundary, set it to 0
```

```
                                else rbc = 0;

                                //find the the biggest value between rc, bc & rbc(the most
obvious edge)
                                cc = biggest(rc, bc, rbc);

                                {
                                        //set the pixel as edge
                                        putPixel1(img, x, y, channel, cc);
                                }
                        }//channel
                }//y
        }//x
}
//=============================================================
// blur an image - take a very long time if too much value
//=============================================================
/*
To do blur effect, scan every pixel in the image one by one,
then add the value with all pixels adjacent to it. Then divide
the value by the number of pixel that we add and put it into
the scanned pixel back.
in other word we get the average color of a pixel with it's surrounding pixels.
*/
void Blur(IMAGE *img, int value)
{
        IMAGE tempIMG; //temporary image for processing
        int             x, y, xx, yy, xplusxx, yplusyy;
        GLubyte ccR, ccG, ccB;
        int             totalColorR=0, totalColorG=0, totalColorB=0;
        int             div = (value*2+1)*(value*2+1); //total number of pixel the we add

        if(value<=0) return;

        //copy the image into the temporary image
        copyImage(img, &tempIMG);

        //scan all pixel
        for(x=0; x<img->sizeX; x++)
        {
                for(y=0; y<img->sizeY; y++)
                {
                        //get center color
                        ccR = getPixel2(tempIMG, x, y, _RED);
                        ccG = getPixel2(tempIMG, x, y, _GREEN);
                        ccB = getPixel2(tempIMG, x, y, _BLUE);

                        //get color surrounding center color
                        for(xx=-value; xx<value+1; xx++)
                        {
                                for(yy=-value; yy<value+1; yy++)
                                {
                                        xplusxx = x + xx;
                                        yplusyy = y + yy;

                                        //sum up all color
                                        if(xplusxx>=0 && xplusxx<img->sizeX
                                                &&      yplusyy>=0      &&      yplusyy<img-
>sizeY)//check the we are not out of the image boundary
                                        {
```

```
                                        totalColorR += getPixel2(tempIMG, xplusxx,
yplusyy, _RED);
                                        totalColorG += getPixel2(tempIMG, xplusxx,
yplusyy, _GREEN);
                                        totalColorB += getPixel2(tempIMG, xplusxx,
yplusyy, _BLUE);
                                }
                                else
                                {
                                        totalColorR += ccR;
                                        totalColorG += ccG;
                                        totalColorB += ccB;
                                }
                        }//yy
                }//xx

                //get average color and put in image
                totalColorR /= div;
                totalColorG /= div;
                totalColorB /= div;

                //put back the average color into the scanned pixel
                putPixel1(img, x, y, _RED,   totalColorR);
                putPixel1(img, x, y, _GREEN, totalColorG);
                putPixel1(img, x, y, _BLUE,  totalColorB);

                //reset the total color for the next pixel
                totalColorR = 0;
                totalColorG = 0;
                totalColorB = 0;

        }//y
    }//x

    //free memory
    destroyImage(&tempIMG);

}
//=========================================================
//set an image to negative color
//=========================================================
/*
Every channel in a pixel has a value range from 0 to 255 (1 byte)
to get the negative value, the formula is 255-value.
*/

void Negative(IMAGE *img)
{
    int     x, y;

    for(x=0; x<img->sizeX; x++)
    {
        for(y=0; y<img->sizeY; y++)
        {
            //put in the pixel the negative value of it own
            putPixel1(img, x, y, _RED,   255-getPixel1(img, x, y, _RED));
            putPixel1(img, x, y, _GREEN, 255-getPixel1(img, x, y, _GREEN));
            putPixel1(img, x, y, _BLUE,  255-getPixel1(img, x, y, _BLUE));
        }
    }
```

```
}
//=============================================================================
//diffuse an image pixels
//=============================================================================
void Diffuse(IMAGE *img, int value)
/*
we get a pixel, and put it at a random position.
value is the random limit so that the pixel will not go too far
*/

{
	int		x, y, xplusrx, yplusry;

	if(value<0)value = 0;

	//scan every pixel one by one
	for(x=0; x<img->sizeX; x++)
	{
		for(y=0; y<img->sizeY; y++)
		{
			//get the random position for the pixel
			xplusrx = x + (rand()%value)-value/2;
			yplusry = y + (rand()%value)-value/2;

			//check that we do not get out of the image boundary
			if(xplusrx>=0 && xplusrx<img->sizeX-1
				&& yplusry>=0 && yplusry<img->sizeY-1)
			{
				//put the pixel at the random position
				putPixel1(img, xplusrx, yplusry, _RED,	getPixel1(img, x, y,
_RED));
				putPixel1(img, xplusrx, yplusry, _GREEN, getPixel1(img, x, y,
_GREEN));
				putPixel1(img, xplusrx, yplusry, _BLUE,	getPixel1(img, x, y,
_BLUE));
			}
		}
	}
}
//=============================================================================
//LineArt effect - like a photostat quality
//value from 0 to 100
//=============================================================================
/*
scan every pixel one by one, then check wether it is larger or
smaller then the value. if it is larger, put back 255(white color) else
put back 0(black color)
*/

void LineArt(IMAGE *img, int value)
{
	int		x, y;
	int		brightness;

	if(value<0)value = 0;
	if(value>100)value = 100;

	value = (value*255*3)/100;

	for(x=0; x<img->sizeX; x++)
```

```
	{
		for(y=0; y<img->sizeY; y++)
		{
			brightness =
				getPixel1(img, x, y, _RED) +
				getPixel1(img, x, y, _GREEN) +
				getPixel1(img, x, y, _BLUE);

			if(brightness>value)
			{
				putPixel1(img, x, y, _RED,   255);
				putPixel1(img, x, y, _GREEN, 255);
				putPixel1(img, x, y, _BLUE,  255);
			}
			else
			{
				putPixel1(img, x, y, _RED,   0);
				putPixel1(img, x, y, _GREEN, 0);
				putPixel1(img, x, y, _BLUE,  0);
			}
		}
	}

}
//=============================================================
//=============================================================
/*
This will be our own image file format when reading/saving the IMG file
	first 3 letters will be the file identification
	char		'I'
	char		'M'
	char		'G'

	next 2 integers are the width and heigth
	int			sizeX
	int			sizeY

	then all the pixel data
	unsigned char	data[sizeX * sizeY * 3channels]
*/
//=============================================================
//Load our own image format
//=============================================================
bool LoadIMG(char *filename, IMAGE *img,
		int *actualWidth, int *actualHeigth)
{
	FILE	*file;
	IMAGE tempIMG;

	char	ID[3] = {'I', 'M', 'G'},
		id;

	//try to open the file
	file = fopen(filename, "rb");
	if(!file)return false;

	//check for correct IMG file opened
	for(int i=0; i<3; i++)
	{
		fscanf(file, "%c", &id);
```

```
            if(id != ID[i])
            {
                fclose(file);
                MessageBox(NULL, "Not an IMG file", "Error", MB_OK);
                return false;
            }
        }

        fscanf(file, "%i ", &tempIMG.sizeX);
        fscanf(file, "%i", &tempIMG.sizeY);

        int sizeimg = sizeof(GLubyte)*tempIMG.sizeX*tempIMG.sizeY*3;
        tempIMG.data = (GLubyte*)malloc(sizeimg);

        GLubyte R, G, B;
        for(int y=0; y<tempIMG.sizeY; y++)
        {
            for(int x=0; x<tempIMG.sizeX; x++)
            {
                fscanf(file, "%c%c%c", &R, &G, &B);

                putPixel2(tempIMG, x, y, _RED,   R);
                putPixel2(tempIMG, x, y, _GREEN,G);
                putPixel2(tempIMG, x, y, _BLUE, B);
            }
        }

        fclose(file);

        //get compatible openGL image size
        //because openGL texture width and height can only be in
        //be these number ie: 8, 16, 32, 64, 128, 256, 512, 1024 only
#define compatibleSize(x) x<=8? 8:x<=16? 16:x<=32? 32:x<=64? 64:x<=128? 128:x<=256?
256:x<=512? 512:x<=1024? 1024:0

        //save img size into temporary variables
        int tempSizeX = img->sizeX;
        int tempSizeY = img->sizeY;

        img->sizeX = compatibleSize(tempIMG.sizeX);
        img->sizeY = compatibleSize(tempIMG.sizeY);

        //if size is valid continue
        if(img->sizeX!=0 && img->sizeY!=0)
        {
            //free img memory from previous data first
            destroyImage(img);

            //reallocate image memory
            sizeimg = sizeof(GLubyte)*img->sizeX*img->sizeY*3;
            img->data = (GLubyte*)malloc(sizeimg);

            //copy from tempIMG to img
            GLubyte lastColor[3];
            for(int x=0; x<img->sizeX; x++)
            {
                for(int y=0; y<img->sizeY; y++)
                {
                    for(int channel=0; channel<3; channel++)
                    {
```

```
                                        if(x<tempIMG.sizeX && y<tempIMG.sizeY)
                                        {
                                                lastColor[channel] = getPixel2(tempIMG, x, y,
channel);
                                                putPixel1(img,      x,      y,      channel,
lastColor[channel]);
                                        }
                                        else
                                                putPixel1(img,      x,      y,      channel,
lastColor[channel]);
                                }
                        }
                }
                *actualWidth    = tempIMG.sizeX;
                *actualHeigth   = tempIMG.sizeY;
        }
        else
        {
                //if size is not valid(too big) get back the old size
                MessageBox(NULL, "Bitmap file too large to display", "ERROR", MB_OK);
                img->sizeX = tempSizeX;
                img->sizeY = tempSizeY;
        }

        destroyImage(&tempIMG);

        return true;
}
//=============================================================
//save IMG file format
//=============================================================
bool SaveIMG(char *filename, IMAGE *img, int actualWidth, int actualHeigth)
{
        FILE *file;

        file = fopen(filename, "wb");
        if(!file)return false;

        fprintf(file, "I");
        fprintf(file, "M");
        fprintf(file, "G");
        fprintf(file, "%i ", actualWidth);
        fprintf(file, "%i", actualHeigth);

        for(int y=0; y<actualHeigth; y++)
        {
                for(int x=0; x<actualWidth; x++)
                {
                        fprintf(file, "%c", getPixel1(img, x, y, _RED));
                        fprintf(file, "%c", getPixel1(img, x, y, _GREEN));
                        fprintf(file, "%c", getPixel1(img, x, y, _BLUE));
                }
        }

        fclose(file);

        return true;
}
```

User Manual

Image Edge Detection is one of the images processing software that is not a server-based architecture. It is a stand-alone application that able to run in any version of Windows platform. There is no installation and configuration needed before this system available to use. All you have to do is just grab the software and run the system directly through the compact disk. The hardware and software requirements for Image Edge Detection System are as followed:

1.1 Hardware requirements

The hardware specifications are:

Processor : Pentium II 166 Mhz and above.

Memory : 56.0 Mbytes of RAM.

Hard disk : at least 4.75 Gigabytes.

1.2 Software Requirements

The software specifications are:

Operating system platform: Microsoft Windows 95, Windows 98, Windows Millennium Edition (ME), Windows NT 4.0, or Windows 2000.

To start using the system, double click the *edge detection* program icon and the main display will pop-up as below:

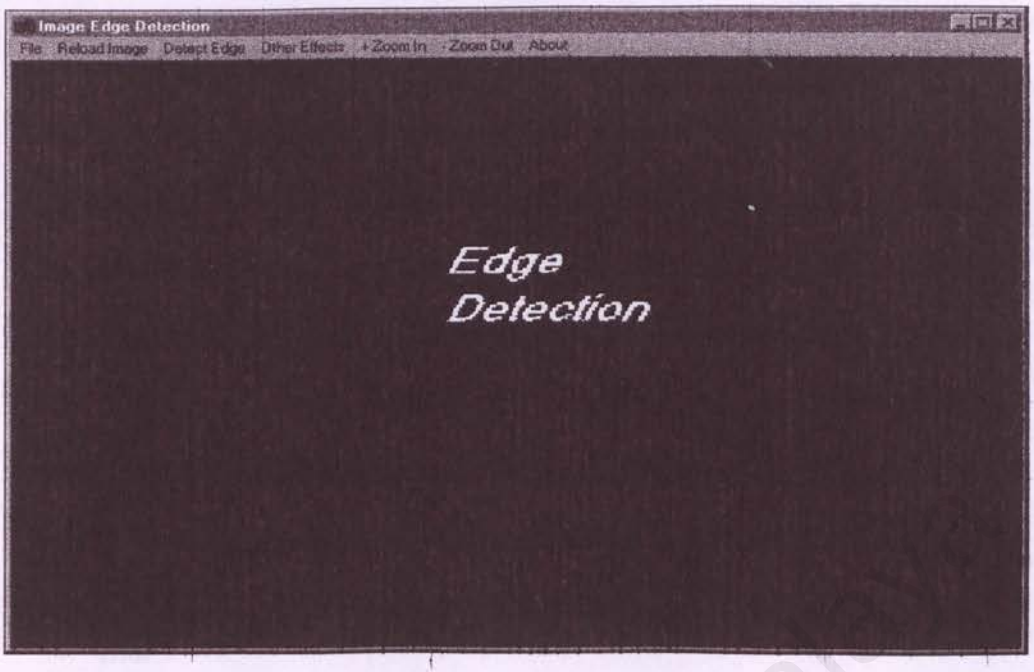


Figure 1.0 : Main display

Before you start loading your own image, you can try all the function first on the default image displayed. For example, by clicking the *Detect Edge* menu, the edge image will be show automatically right after the menu clicked.

Figure 2.0 : Edge image

If you want to get back to the original image, click the *Reload Image* menu. This menu works like undo function but it cannot turn back to the previous image process. For example, if you apply another effect to the edge image, when the *Reload Image* clicked, it will show the original image, not the edge image.

The *Zoom In* and *Zoom Out* menu is for enlarge or decrease the image size. All the function under the *Other Effects* menu is just a peripheral function that is included in this system. The alphabetical beside the function is a short key that you can press to apply the effect on image. This is the alternative and easy way to apply the function rather than click each function manually. The results of each function are as below:

- Blur

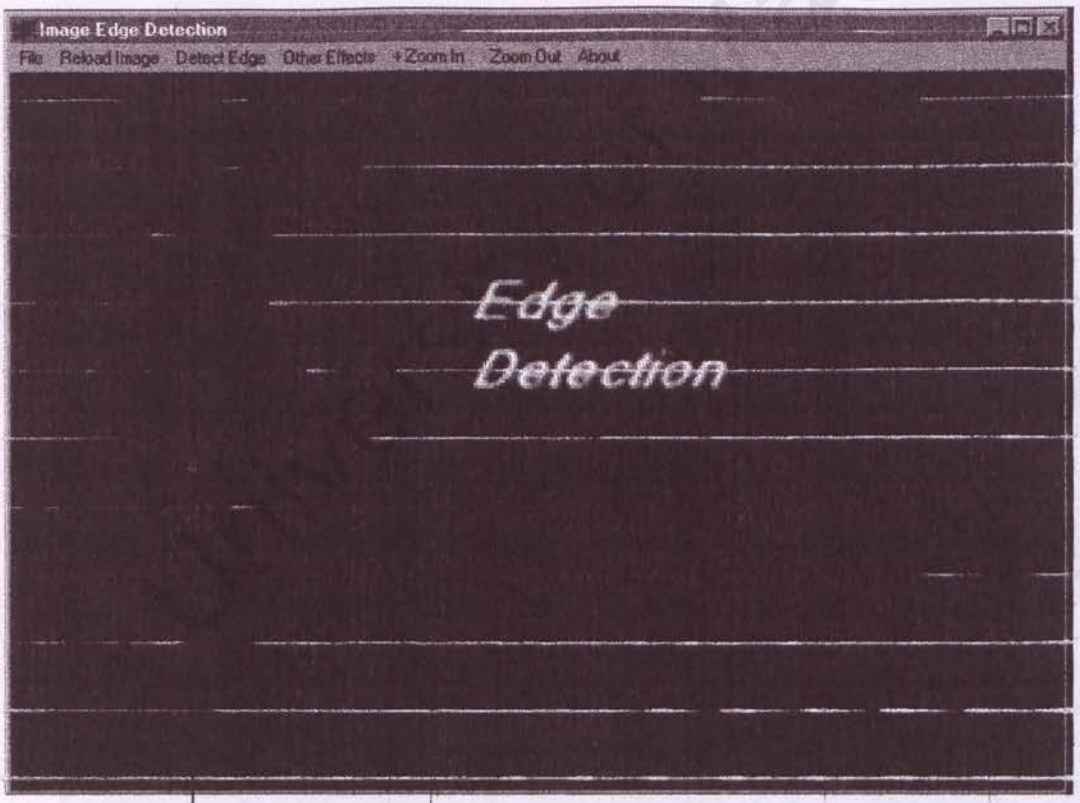


Figure 3.0 Blur image

- Blur more

Figure 4.0 : Blur more image

- Negative

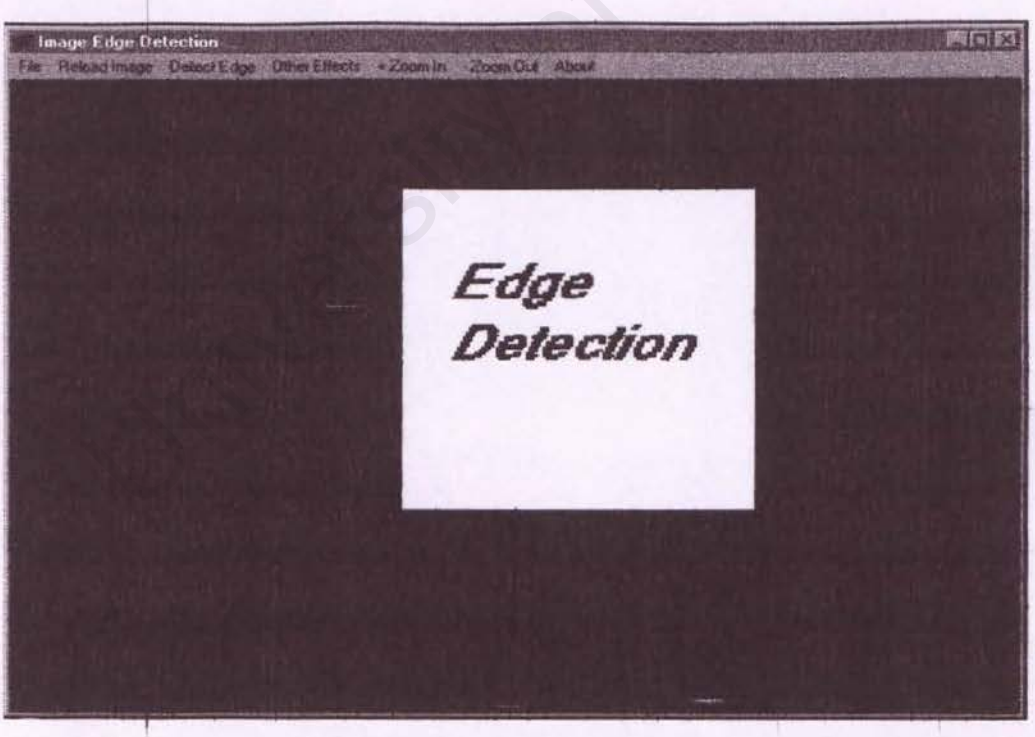


Figure 5.0 : Negative image

- Line art

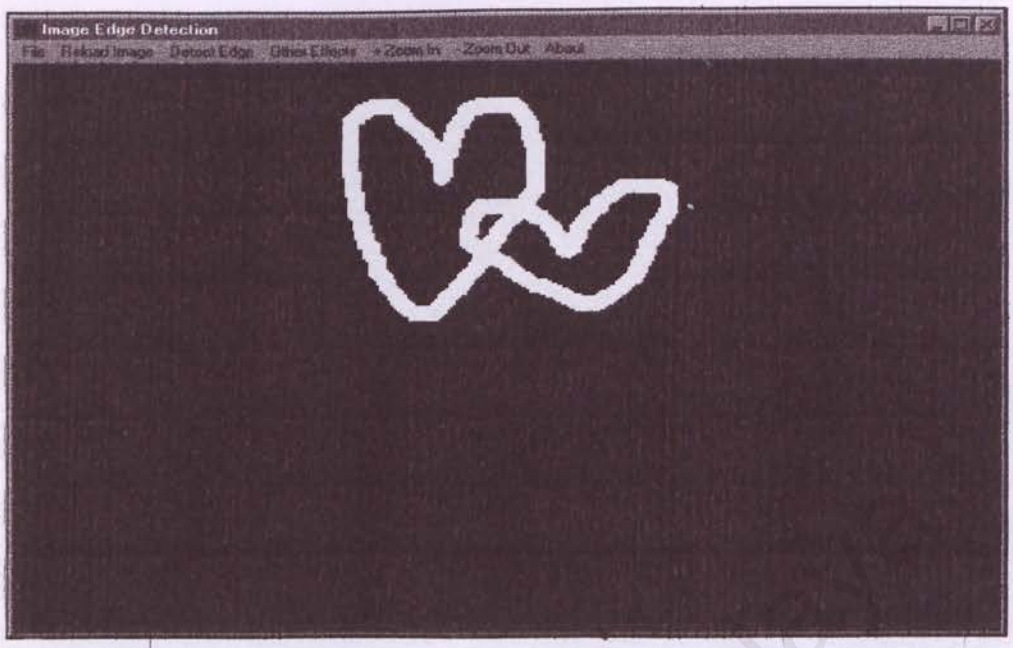


Figure 6.0 : Line art image

Note : This function is to convert the image to black and white. If you apply the line art function on the default image, *Edge Detection* there will no effect on the image because the image is already in black and white.

- Diffuse

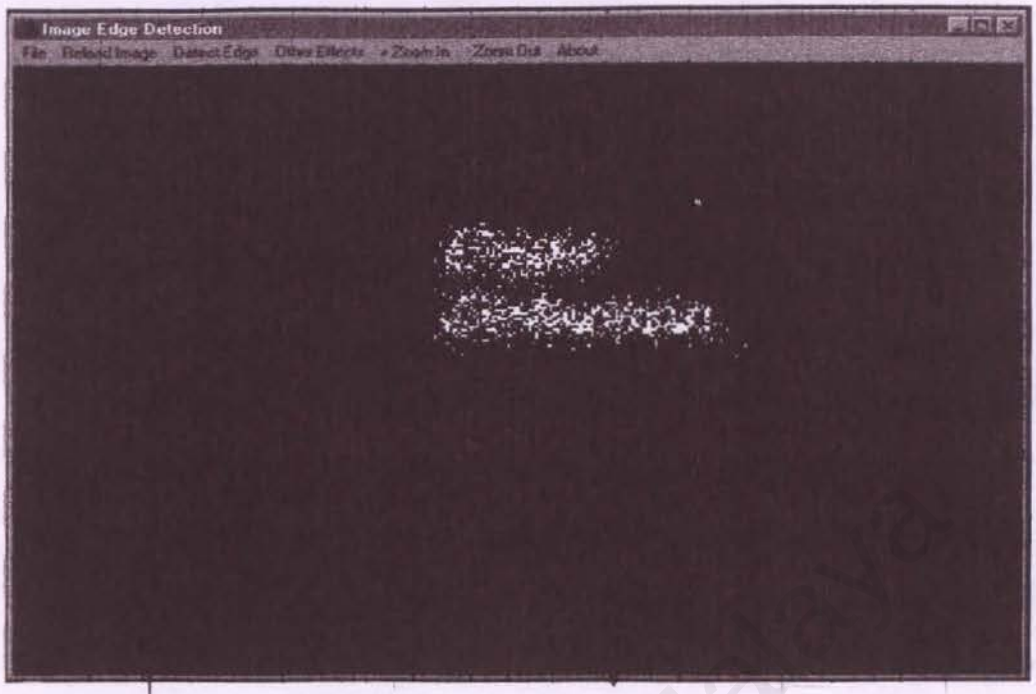


Figure 7.0 : Diffuse image

Finally, the most right menu is all about the developer.

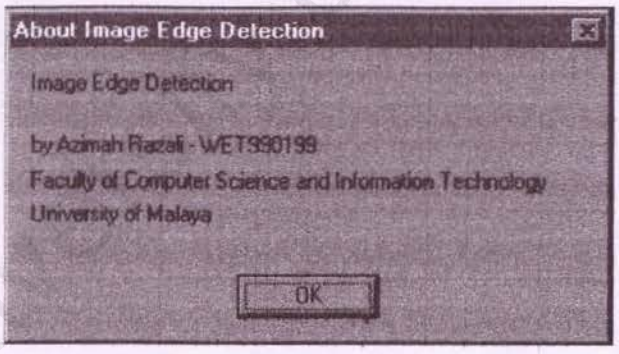


Figure 8.0 The developer information

After the trial session, now you can try to load your own image but remember that this system only supports two kind of image, that is jpeg and bitmap. Click the *Open Bitmap* to load the .bmp image format and Open Jpeg to load the .jpg format. The *Load IMG* function is for retrieve the processed image that has been save in .img format. To save the processed image, click *Save IMG* function.

Please note that all the processed image that you save are automatically formatted to .img, which mean the image saved can only be display in this system, not your computer. Your computer cannot read this format because this is not a standard format like jpeg and bitmap. To exit this system just clicks *Exit* function or the 'x' button on the right side corner.

1. Pfleeger, Shari Lawrence. (2001). *Software Engineering Theory and Practice*. 2nd ed. Prentice Hall Inc.
2. Sommerville, Ian. (2001). Software Engineering. 6th ed. Addison-Wesley.
3. Lee Yew Fei. (2000/2001). E-courier (Package Tracking System). Bach. Thesis. University of Malaya.
4. Christina Shanti. (1999/2000). MR Image 3D Reconstruction and Volume Visualization. Bach. Thesis. University of Malaya.
5. MATLAB Application Program Interface Guide, The Mathwork Inc, 1998.
6. Computer Vision & Image Processing a practical approach using CVIP tools, Scott E. Umbaugh, Prentice-Hall, Inc 1998
7. Introductory remote sensing : digital image processing & applications, Paul J. Gibson & Clare H. Power, 2000, St. Edmundsbury Press, Bury St. Edmundssuffolk
8. Angel, E. *Interactive Computer Graphics: A Top-Down Approach with OpenGL*. Reading, MA: Addison-Wesley, 2000.
9. Davies, A., and P. Fennessy. *Digital Imaging for Photographers*. Boston: Focal Press, 1998.
10. Foley, J., A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Reading, MA: Addison-Wesley, 1990.
11. Gonzalez, R. C., and P. Wintz. *Digital Image Processing*. Reading, MA: Addison-Wesley, 1977.
12. Hall, E. L. *Computer Image Processing and Recognition*. New York: Academic Press, 1979.
13. Hill, F. S. *Computer Graphics*. New York: Macmillan, 1990.

14. Holzmann, G. J. *Beyond Photography: The Digital Darkroom*. Englewood Cliffs, NJ: Prentice Hall, 1988.

15. Hough, T., ed. *The Joy of Photography*. Reading, MA: Addison-Wesley, 1991.

16. Kruglinski, D. J., G. Shepherd, and S. Wingo. *Programming Microsoft Visual C++ Fifth Edition*. Redmond, WA: Microsoft Press, 1998.

17. Lindley, C. A. *Practical Image Processing in C*. New York: John Wiley & Sons, Inc., 1991.

18. Lyon, D. A. *Image Processing in Java*. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

19. Martinez, B. and J. Block. *Visual Forces: An Introduction to Design*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1995.

20. Parsons, T. W. *Introduction to Algorithms in Pascal*. New York: John Wiley & Sons, Inc., 1995.

21. Pitas, I. *Digital Image Processing Algorithms and Applications*. New York: John Wiley & Sons, Inc., 2000.

22. Seul, M., L. O'Gorman, and M. Sammon. *Practical Algorithms for Image Analysis: Description, Examples, and Code*. Cambridge: University Press, 2000.

23. Sphar, C. *Learn Microsoft Visual C++ 6.0 Now*. Redmond, WA: Microsoft Press, 1999.

24. Teuber, J. *Digital Image Processing*. New York: Prentice Hall, 1993.

25. http://www.mathworks.com

26. http://hwr.nici.kun.nl

27. http://peipa.cssex.ac.uk

28. http://www.cs.cmn.edu/afs/cs/project/cil/ftp/html/vision.html

29. http://www.sci.lib.uci.edu/HSG/MedicalImage.html

30. http://www.rz.go.dlr.de:8081/softarch.html

31. http://www.eecs.wsu.edu/Ipdb/title.html

32. http://www-isis.ecs.soton.ac.uk/research/visinfo/rgroup.html

33. http://george.lbl.gov/computer_vision.html

34. http://george.lbl.gov/ITG.html

35. http://www.video.eecs.berkeley.edu/

36. http://www.cg.tuwien.ac.at/studentwork/CESCG97/boros/

37. http://www.ping.be/~ping1339/polar.htm

38. http://www-sop.inria.fr/chir/personnel/devernay/publis/distcalib/

39. http://www.pcigeomatics.com/cgi-bin/pcihlp/IHS

40. http://www.webmonkey.com/programming/

41. http://www.nada.kth.se/~tony/ceru-review/ceru-html/node11.html

42. http://www.www.math.mtu.edu/~msgocken/intro/intro.html