PENGENALPASTIAN BAKTERIA TIBI

Disertasi ini dikemukakan kepada
UNIVERSITI SAINS MALAYSIA

Sebagai memenuhi sebahagian daripada syarat keperluan
untuk ijazah dengan kepujian

SARJANA MUDA KEJURUTERAAN (KEJURUTERAAN ELEKTRONIK)

Oleh

Nazrul Hilmi bin Mohammad

Pusat Pengajian Kejuruteraan
Elektrik dan Elektronik
Universiti Sains Malaysia

Mac 2005

ABSTRAK

Kaedah konvensional dalam mengenalpasti bakteria *Mycobacterium tuberculosis* daripada sampel air liur adalah menyukarkan dan sangat terdedah kepada ralat kerana pengiraan organisma dilakukan secara manual oleh juruteknologi ketika sesuatu spesimen sedang diperiksa melalui mikroskop. Kaedah ini lazim menyebabkan kelesuan mata yang boleh mengakibatkan banyak ralat. Oleh yang demikian, seseorang juruteknologi dapat memeriksa tidak lebih daripada dua puluh specimen dalam masa sehari untuk mengelakkan kelesuan yang boleh menyebabkan ralat itu. Maka kaedah konvensional ini telah dianggap sebagai tidak produktif. Sebagai suatu usaha untuk mengatasi masalah ini, projek ini membangunkan satu sistem perisian pemprosesan imej digital, yang direkabentuk dan diimplimentasikan dengan menggunakan alatan pembangunan perisian visual Borland C++ Builder. Perisian yang dibangunkan ini secara khususnya disasarkan untuk memudahkan proses pengenalpastian bakteria *Mycobacterium tuberculosis* dengan menggunakan imej-imej digital spesimen air liur daripada pesakit-pesakit yang disyaki. Ia merangkumi tiga peringkat pengimejan utama: peningkatan imej, peruasan imej, dan analisa gugusan. proses peningkatan imej dalam perisian ini melibatkan transformasi jalur kelabu dan penurasan negatif untuk menyediakan data imej bagi dua proses yang seterusnya. Kemudian, proses peruasan imej menyediakan pilihan untuk pengguna sama ada menggunakan teknik pertumbuhan kawasan berasaskan titik benih secara manual atau automatik. Seterusnya, teknik analisa dan pengiraan gugusan ecara automatik mengira bilang organisma yang terdapat dalam imej. Adalah diharapkan sistem perisian ini dapat diterima sebagai aplikasi yang pantas dan tepat untuk mengenalpasti bakteria *Mycobacterium tuberculosis* daripada spesimen air liur pesakit.

ACKNOWLEDGEMENTS

This project has benefited from a number of individuals who gave generously their time and expertise, and the following persons deserve special mentions. My highest thanks goes to Associate Professor Umi Kalthum Ngah, my project supervisor who has done a lot to facilitate me for the whole duration of the project, right from the day I was assigned the project title, next to the implementation of the project, and further to the completion of this dissertation. I would also like to thank Dr. Suraiya Mohd. Noor from the School of Medicine for facilitating me during the data acquisition activity. Also, I must express my appreciation to Dr. Nor Ashidi Mat Isa who has been assigned as the second examiner for my project, for his willingness to examine this dissertation and attend my presentation alongside my project supervisor. Furthermore, my acknowledgement extends to my dearest counterparts, Tina Zahani Zainuddin and Syed Mohamad Hatmi Syed Zainal Abidin, under the supervision of the same supervisor as mine, for their bits of suggestions and constructive criticisms that helped me quite a lot to improve my project. Thanks also to the Final Year Project Coordinator for this term, Associate Professor Dr. Mohd. Yusof Mashor who provided general guidelines on the project development, particularly on the rules and standards in writing this dissertation. Last but not least, my undying thanks to all other individuals whom have not been mentioned here in particular for their own bit of direct and indirect contribution in this project.

TABLE OF CONTENTS

Page

DIAGRAM LIST

PREFACE

This dissertation is written as an official report for the **Identification of Tuberculosis Bacteria** project that I have completed. In other words, this is the written representation of the project that covers every aspect of it, beginning from the background that leads to the initiation of this project, further to the project methodology, and complete with the discussion on the results obtained by using the developed software and recommendation for possible improvement of the system itself. Chapter 1 gives an introduction of the project, which discusses the background issues that lead to the development of the system and the objectives to be achieved by the solution. This chapter shall also describe in brief on the development stages of the software system. Next is the Chapter 2, which covers in brief about Borland C++ Builder, the development tool that is used in designing and implementing the software. In Chapter 3, there lays the critically important part of the project itself. The chapter starts with a brief discussion on digital image processing fundamentals and further extends to discuss image enhancement, segmentation, and clustering algorithms applied in the software programme. Next, Chapter 4 discusses the tests conducted against the software system and the acceptability of the results. Also, the factors of some non-conformity that arise from the tests are also covered. Finally, Chapter 5 gives overall conclusion on the project and discusses possible improvements to be applied on the system in the future.

CHAPTER 1

INTRODUCTION

1.1 Preface

As reiterated by Forero et al., *Mycobacterium tuberculosis* (TB) bacilli are the origin of the pulmonary tuberculosis disease, although these microbes can infect tissues of other organs such as brain, kidneys, bone, and skin. According to Ginsberg (1998), Tuberculosis is the main cause of death resulting from infectious illness whereas the World Health Organisation (WHO) states that 1.722 billion people are carriers of *Mycobacterium tuberculosis*, triggering 10 million cases of active tuberculosis worldwide and approximately 3 million of deaths annually. Cumbersome traditional method for identification of *Mycobacterium tuberculosis* bacteria requires a technologist to count visible bacteria clusters manually as he or she is examining the specimen through a microscope. As this technique causes eye fatigue that leads to reading error, a technologist can only examine a limited number of specimens per day, leaving this traditional method irrelevant and counter-productive.

Image processing techniques provide a good tool for improving the manual screening of specimens. As a mean of solution for the problem, this software is designed to automatically identify *Mycobacterium tuberculosis* clusters from digitised microscopic images of sputum specimens. This software incorporates three major image processing domains; image enhancement, cluster segmentation, and cluster counting. Image enhancement provides negative filtering to prepare the image data for clustering. The cluster segmentation technique involves the application of k-mean clustering technique and seeded region growing based on region mean. Furthermore, cluster counting algorithm performs the quantitative detection of clusters developed from the previous technique.

1.2 Project Development Stages

The development of this project consists of several stages. It consists of academic research on tuberculosis disease and image processing techniques, self-tutorial for familiarisation with Rational Rose and Borland C++ Builder software engineering tools, software requirement analysis, software design, data acquisition activity, software implementation, units integration, and software testing. Sequential development of this project is depicted in the following flow chart:

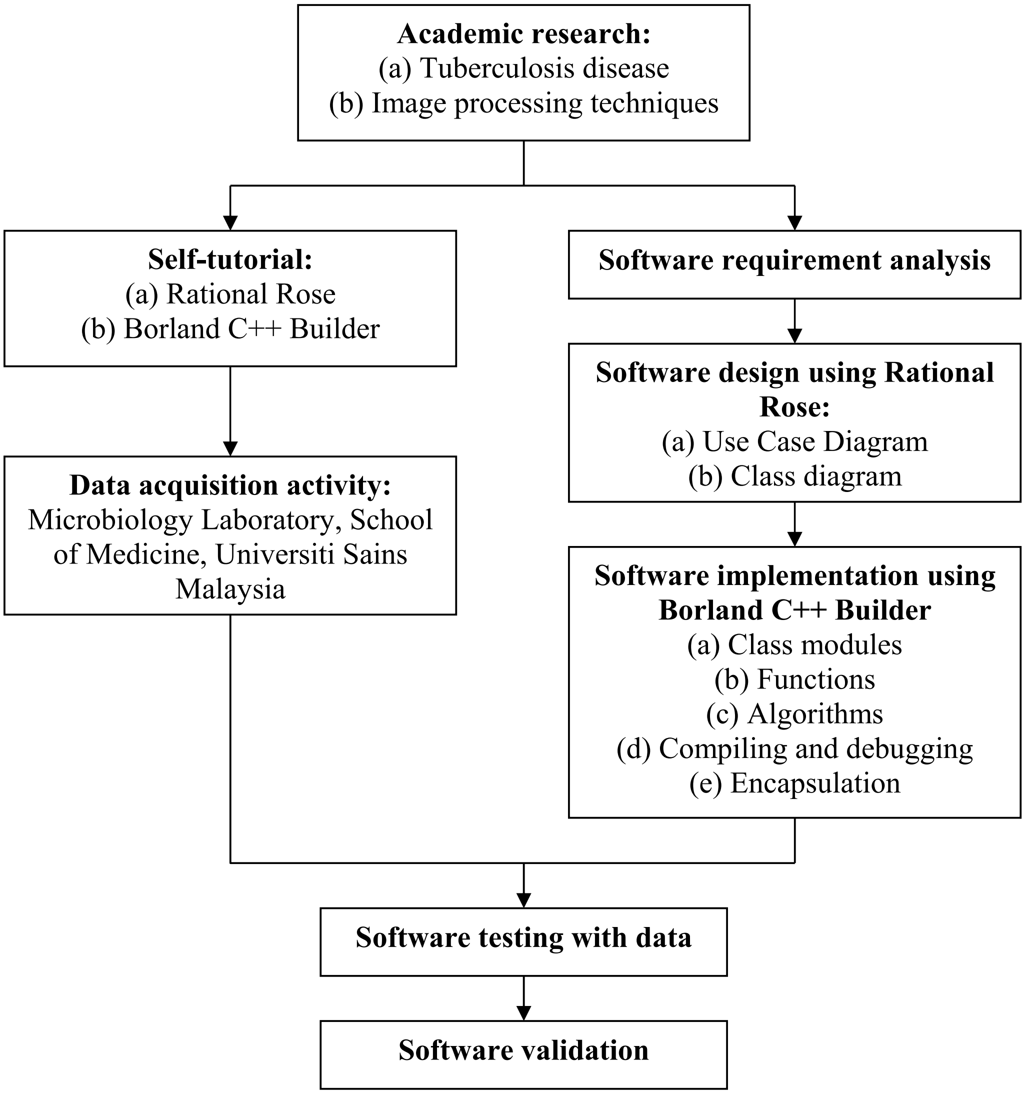

Academic research:
(a) Tuberculosis disease
(b) Image processing techniques
Self-tutorial:
(a) Rational Rose
(b) Borland C++ Builder
Software requirement analysis
Software design using Rational Rose:
(a) Use Case Diagram
(b) Class diagram
Data acquisition activity:
Microbiology Laboratory, School of Medicine, Universiti Sains Malaysia
Software implementation using Borland C++ Builder
(a) Class modules
(b) Functions
(c) Algorithms
(d) Compiling and debugging
(e) Encapsulation
Software testing with data
Software validation

Figure 1.1: Project Development Stages

1.3 Project Objective

The software, in particular, is designed and implemented to fulfil the following objectives:

1. Enhance digital microscope images of sputum specimens in overall through negative filtering to prepare it for cluster segmentation.
2. Developing clusters of Mycobacterium tuberculosis organisms in the image from colour and shape using either k-mean clustering or seeded region growing based on region mean.
3. Analyse and count Mycobacterium tuberculosis clusters in the image.

1.4 Introduction to Digital Image Processing

An image could be defined as a two-dimensional function f(x,y), where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the intensity or gray level of the image at that point. When x, y, and the amplitude values of f are all finite, discrete quantities, the image is called a digital image. Digital image processing refers to processing digital images by means of a computer. A digital image consists of a finite number of elements, each of which has a particular location and value. These elements are referred to as picture elements, image elements, and pixels. Pixel is the most frequently used term to denote the elements of a digital image. Digital image processing techniques implemented in this project will be discussed in Chapter 3.

1.5 Report Outline

Chapter 2 provides a brief description on the development tools used in implementing this project, followed by Chapter 3, which discusses digital image processing techniques in detail. Next, Chapter 4 discusses the test results and factors that affected the results. Finally, Chapter 5 concludes this project in overall and suggests some recommendations for improvements in the future.

CHAPTER 2

DEVELOPMENT TOOL: BORLAND C++ BUILDER

2.1 Preface

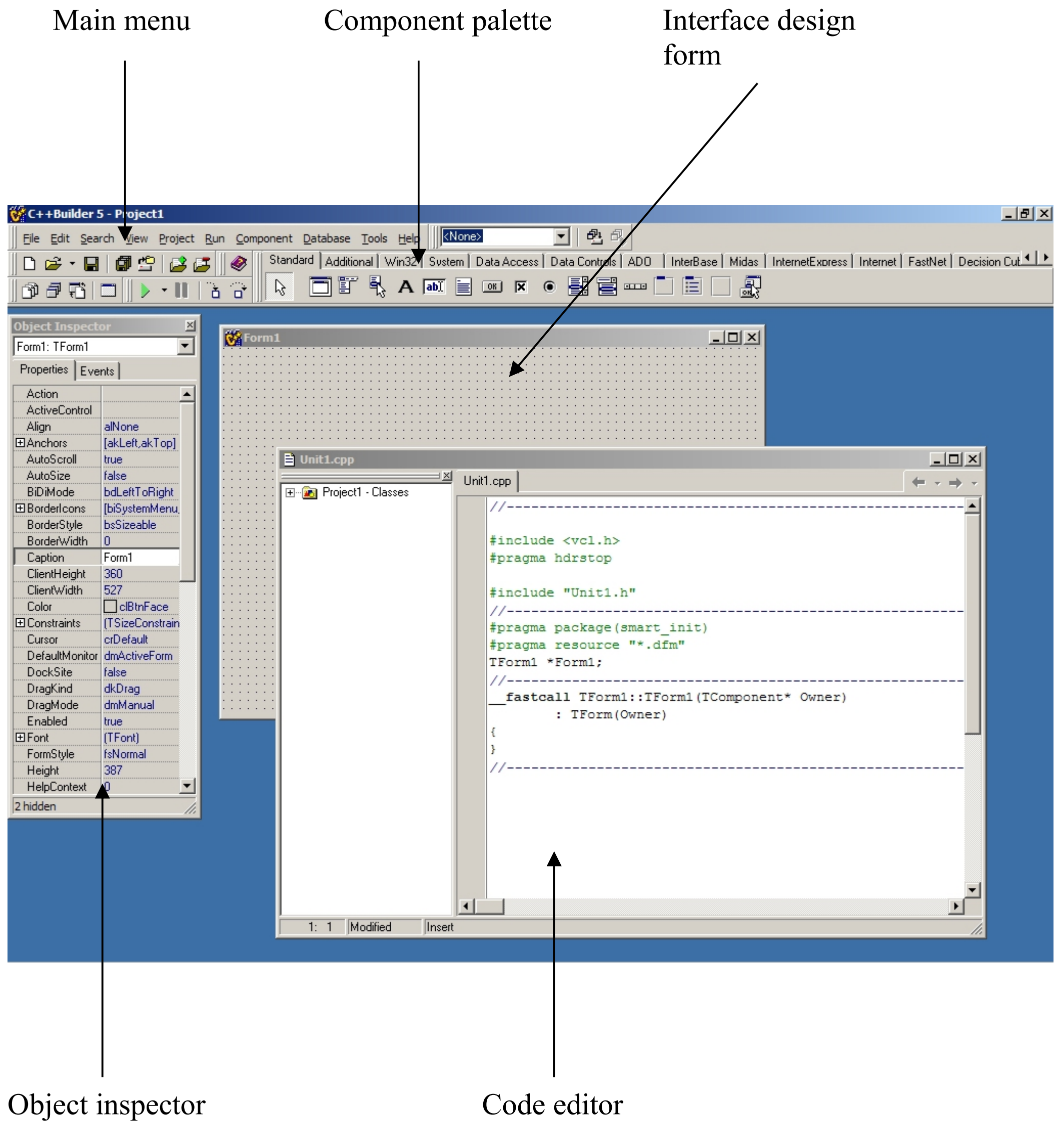
Computer users will always prefer software solutions that are easy to use and can be run on any computers. Thus, software engineers and programmers in designing and implementing their solutions always prefer visual programming tools. In this project, Borland C++ Builder is chosen as the development tool as it incorporates the powerful C++ programming language together with visual component development.

2.2 Introduction to Borland C++ Builder

Borland C++ Builder is a Rapid Application Development (RAD) package introduced by Inprise Corporation for software engineers to develop user-friendly applications based on the highly versatile and powerful C++ programming language. The tool utilises object-oriented-programming (OOP) in building a software package.

Figure 2.1 shows the basic interface of Borland C++ Builder. The interface features several essential parts:

- Main Menu
- Toolbar
- Component Palette
- Interface Design Form
- Object Inspector
- Project Manager
- Code Editor


Main menu
Component palette
Interface design form
C++Builder 5 - Project1
File Edit Search View Project Run Component Database Tools Help
<None>
Standard | Additional | Win32 | System | Data Access | Data Controls | ADO | InterBase | Midas | InternetExpress | Internet | FastNet | Decision Cube
Form1
Unit1.cpp
Project1 - Classes
Object inspector
Code editor

Object Inspector

Form1: TForm1

Properties | Events

Action	
ActiveControl	
Align	alNone
⊞Anchors	[akLeft,akTop]
AutoScroll	true
AutoSize	false
BiDiMode	bdLeftToRight
⊞BorderIcons	[biSystemMenu,
BorderStyle	bsSizeable
BorderWidth	0
Caption	Form1
ClientHeight	360
ClientWidth	527
Color	□ clBtnFace
⊞Constraints	(TSizeConstrain
Cursor	crDefault
DefaultMonitor	dmActiveForm
DockSite	false
DragKind	dkDrag
DragMode	dmManual
Enabled	true
⊞Font	(TFont)
FormStyle	fsNormal
Height	387
HelpContext	0

2 hidden

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

1: 1 | Modified | Insert

Figure 2.1: Borland C++ Builder interface

The main interface has toolbar on the left side and component palette on the right side. The component palette contains functional tools for programmers to build interface display on the form such as images, labels, buttons, scroll bar, and many others. The object inspector contains the properties for the components selected by the programmer. As a unit is being built, the code editor enables the programmer to write desired programmes to get the components work.

Object-oriented Programming (OOP) is a discipline in programming that is not associated to only one programming language. Theoretically, most programming languages from high-level languages and above can be applied with OOP, but practically, programmers only use the programming languages that are meant for OOP.

2.3 Borland C++ Builder Debugging Utility

The integrated debugger in Borland C++ Builder facilitates debugging and validation processes in developing software. C++ Builder debugger automatically launches when the “Run” button is pressed. Capability to control the variables in debugging is important as it predicts whether a programme has been implemented correctly.

2.4 Conclusion

The software package for this project is built by using Borland C++ Builder. As Borland C++ Builder facilitates in building windows-based application, it is renowned as a highly coveted software development tool.

CHAPTER 3

DIGITAL IMAGE PROCESSING TECHNIQUES

3.1 Preface

As described earlier in Section 1.4, digital image processing is a scientific discipline that is related to the manipulation of images and its data, digitally. It can also be defined as a proper technique in two-dimensional digital image processing which is aimed to gather information from the manipulated image. Lindley (1991) stated that digital image processing could be applied for images and its datum when any of the following conditions is fulfilled:

1. The images have to be enhanced and modified to improve quality or to establish a certain aspect from the information available in images.
2. The images have elements that can be classified, categorised, matched, or measured.
3. The images have parts or elements that have to be merged or rearranged.

This project applies digital image processing techniques in concern to conditions 1 and 2. It implies that digital microscope images of sputum specimens have to be enhanced to obtain a better image quality (condition 1) and that clusters of *Mycobacterium tuberculosis* organism need to be extracted from the images (condition 2).

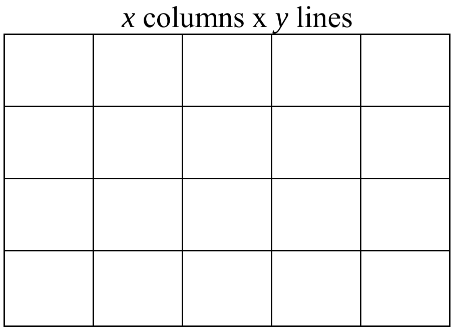
This chapter will first discuss on fundamental concepts in digital image processing. Furthermore, digital image processing techniques applied in this project will be given emphasis. The techniques that will be covered are gray level transformations, region growing, and cluster counting.

3.2 Fundamental Concepts in Digital Image Processing

In digital image processing, fundamental concepts that should be given emphasis include digital image definition, types of digital image processing, gray level, histogram, and convolution. These concepts are given detailed description as follows.

3.2.1 Digital Image Definition

As being described by Gonzalez and Woods (2002), a digital image contains a basic element known as pixel. A digital image consists of a matrices array of pixels. Each individual pixel holds a certain characteristic on its position in the image. Figure 3.1 shows the representation of a digital image by its pixels in matrices form. Shown in Figure 3.1 (a) is an array of pixels represented in grid with *x* columns and *y* lines in two dimensions, while Figure 3.1 (b) shows the image represented in matrices form.



$$\begin{pmatrix} a_{0,0} & \cdots & a_{x-1,0} \\ \vdots & \ddots & \vdots \\ a_{0,y-1} & \cdots & a_{x-1,y-1} \end{pmatrix}$$

(a) An array of pixels with x columns and y lines

(b) Matrices representation for the pixel array in (a)

Figure 3.1: Matrices representation for an image

A pixel P at a coordinate (x, y) has two adjacent neighbours of same column and two adjacent neighbours of the same line with the following coordinates: (x, y-1), (x, y+1), (x-1, y), and (x+1, y). All eight adjacent pixels for the pixel P are shown in Figure 3.2.

(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	P (x, y)	(x+1, y)
(x-1, y+1)	(x, y+1)	(x+1, y+1)

Figure 3.2: Adjacent pixels for pixel P

3.2.2 Types of Digital Image Processing

Lindley (1991) has suggested four different types of image processing as the following: point processing, spatial processing, frame processing, and geometrical processing. As frame processing is not applied in this project, only the remaining three processing types are discussed here.

Point processing has become the basis for most digital image processing operations. The value of a pixel in an image would be altered solely based on the value of the pixel itself. The pixel value would be replaced by a new value based on the original value of the pixel. Spatial relationship in the image remains intact after a point processing operation. Thus, information in the image also remains unchanged. Histogram representation is one of the techniques used in this project that based on point processing.

Spatial processing uses a group of pixels in an image to obtain information about the image. Neighbouring pixels in this group are adjacent to the desired pixel, where its neighbours surround the desired pixel. Calculation processes that involve spatial processing include mean, variance, and convolution. Spatial processing could function using convolution as spatial filter and edge detector.

Geometrical processing would change the array or position of pixels in an image. Pixels array and position could be altered through image enlargement and image rotation techniques. Despite of being marginally applied in this project, the process remains significant as an image could sometimes provide information in greater detail through geometrical conversion.

3.2.3 Gray Level

Pixels are the smallest unit that represents a digital image. Each of the pixels that made up the image contains different gray level value. Different combination of gray level values dictates the brightness and intensity of the image. In this project, sputum specimen images in colour are transformed into grayscale images with 256 gray level values in which the value 0 represents the darkest up to 255 for the brightest pixel. Other pixels might have different brightness and intensity levels between these two values. All the images used in this project are of bitmap (BMP) format.

3.2.4 Histogram

Histogram for a digital image is essentially a discrete function in the form of bar chart that displays the quantity of pixels with certain gray level values. Alang Ahmad (1999) defines that histogram is a tabulation with input for every possible gray level value of pixels, which provides the number of pixels with certain gray level value.

Histogram closely relates to the probability density function, which tells the probability of a pixel having a certain gray level value, for a group of pixels in the image. Histogram is fundamentally defined as a set of M numbers as given in Equation 3.1.

$$h_i = \frac{n_i}{n_t} \ , \quad \text{for i = 0,1,2…M-1} \tag{3.1}$$

where M = the total of gray levels in an image

i = index number representing the i-th gray level

n_i = number of pixels in the image with i-th gray level

n_t = total number of pixels in the image

Histogram for an image is usually displayed in the form of graph, in which the number of pixels versus gray level values, as shown in Figure 3.3. It is common that certain gray level values have a lot of pixels while other gray level values have too little pixels or no pixel at all. Brightness of the image predicts the curve of the histogram graph. A dark image would have high curves on the left part of its histogram, depicting a large number of pixels with low gray value levels. Instead, a bright image would have high curves on the right part of its histogram, depicting a large number of pixels with high gray value levels.

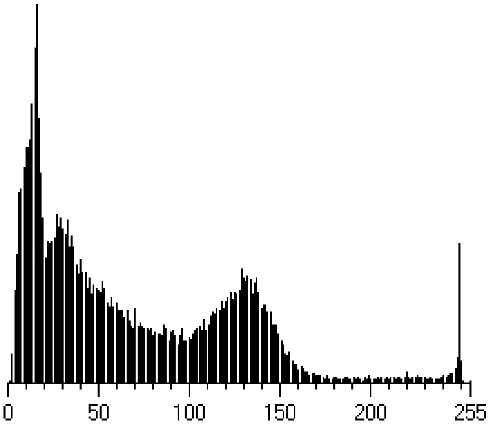


Figure 3.3: Histogram

Despite of a number of information could be obtained from a histogram; it does not provide any information on the contents of the image itself. Position of the histogram peak represents the image intensity while the peak width, also known as dynamic range, indicates its contrast. A small dynamic range represents low contrast, and vice versa. Understanding histogram shapes is useful since it predicts the probability for contrast enhancement of an image. Besides contrast enhancement, a histogram is also viable in predicting a threshold value, whose position is between two histogram peaks, for image segmentation processes. A correct or suitable threshold value would enable a clear differentiation of two objects.

3.2.5 Convolution

Convolution, which is easy to understand and apply, is a renowned technique particularly in digital signal processing. As suggested by Alang Ahmad (1999), convolution could be applied to implement various spatial processing transformations. Filtering and detection techniques largely involve convolution technique.

Chuah (1999) defines that image convolution is a cross product between N x N image data and M x M mask function. This concept could be illustrated using a 3 x 3 image data with values from X_1 to X_9 and a 3 x 3 mask function with values from Y_1 to Y_9. Representation for both the image data and mask function are shown in Figure 3.4.

X_1	X_2	X_3
X_4	X_5	X_6
X_7	X_8	X_9

(a) Image data

Y_1	Y_2	Y_3
Y_4	Y_5	Y_6
Y_7	Y_8	Y_9

(b) Mask function

Figure 3.4: 3 x 3 representations for image data and mask function

Convolution of image data with mask function produces the result, which is described in Equation 3.2.

$$[X] * [Y] = X_1Y_1 + X_2Y_2 + X_3Y_3 + \ldots + X_9Y_9 \qquad \textbf{(3.2)}$$

This result would replace the position of X5 for the image once the processing is complete. However, the original value of the position cannot change, as the data in that particular pixel is required for the processing of its neighbours.

3.3 Gray Level Transformation

As described by Gonzalez and Woods (2002), gray level transformations are among the easiest of all image enhancement techniques. The values of pixels before and after processing could be denoted as r and s, respectively. These values are related by an expression, which is shown in Equation 3.3,

$$s = T(r) \tag{3.3}$$

where T is a transformation that maps pixel value r into pixel value s. As digital image processing involves digital quantities, values of the transformation function are stored in a one-dimensional array and the mappings from r to s are implemented via table lookups, in which an 8-bit lookup table will have 256 entries ranging from 0 up to 255, in representative of 256 gray level values.

Figure 3.5 shows three basic types of functions widely used for image enhancement: linear (negative and identity transformations), logarithmic (log and inverse-log transformations), and power-law (n-th power and n-th root transformations).

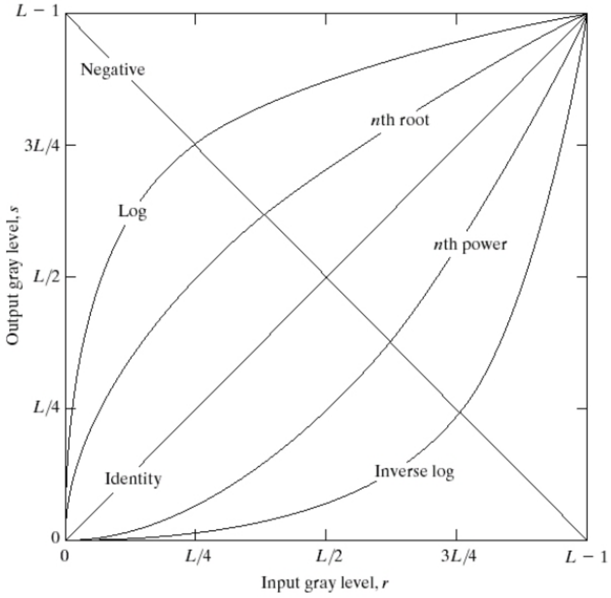


Figure 3.5: Basic gray level transformations used for image enhancement

3.3.1 Negative Transformation

The negative of an image with gray level values in the range from 0 to L-1 is obtained by using negative transformation shown in Figure 3.5, which is given by the expression in Equation 3.4,

$$s = L - 1 - r \qquad \textbf{(3.4)}$$

where s = new gray level value
r = original gray level value
L = number of gray levels

*For 8-bit gray level images, it is $L = 2^8 = 256$.

Reversing the intensity levels of an image in this way produces the equivalent of a photographic negative.

3.3.2 Log Transformation

Equation 3.5 represents the general form of log transformation.

$$s = c \log (1 + r) \qquad \textbf{(3.5)}$$

where c is a constant and it is assumed that $r \geq 0$. The shape of the log curve in Figure 3.5 shows that this transformation maps a narrow range of gray level values in the input image into a wider range of output levels. Log transformation could be applied to expand the values of dark pixels in an image while compressing the higher-level values. The opposite for this transformation is inverse-log transformation, where the values of bright pixels are expanded while dark values are compressed.

3.3.3 Gray Level Transformation in This Project

In this project, a specimen image is supposed to be transformed from colour into gray level prior to enhancement and clustering process. The gray level transformation algorithm in this software consecutively performs colour depth conversion and negative filtering with a single button click. Gray level transformation is considered to suit the image data to the developed grayscale image processing algorithm.

Processing an image in colour domain would require the algorithm to perform extensive considerations and manipulations for red, green, and blue (RGB) channel levels for each pixel. Manipulating three colour levels together for every pixel needs for more advanced and complex algorithms. Besides a cumbersome development of algorithms for colour image enhancement and segmentation, clustering process in colour domain is very resource-intensive and would stretch the processing capability of the computer to its limit. Hence, gray level transformation with negative filtering is considered a viable technique to adapt colour specimen images into grayscale image processing algorithm.

Gray level transformation algorithm in the software applies a simple negative transformation technique. The negative gray level value of a pixel is the deduction of its original gray level value from 255. Gray level transformation process implemented in the software is described by the following flow chart in Figure 3.6. Initially, the software reads each pixel in the colour image in terms of brightness and appoints the brightness value as gray level value. The algorithm scans the pixels beginning with pixel (0,0) and performs negative transformation by deducting the original gray level value from 255. The x-coordinate iterates as the algorithm transforms the next pixel. Once all the pixels of (x,0) coordinates have been negative-transformed, the y-coordinate iterates. The process is performed recursively until all pixels in the image have been negative-transformed.

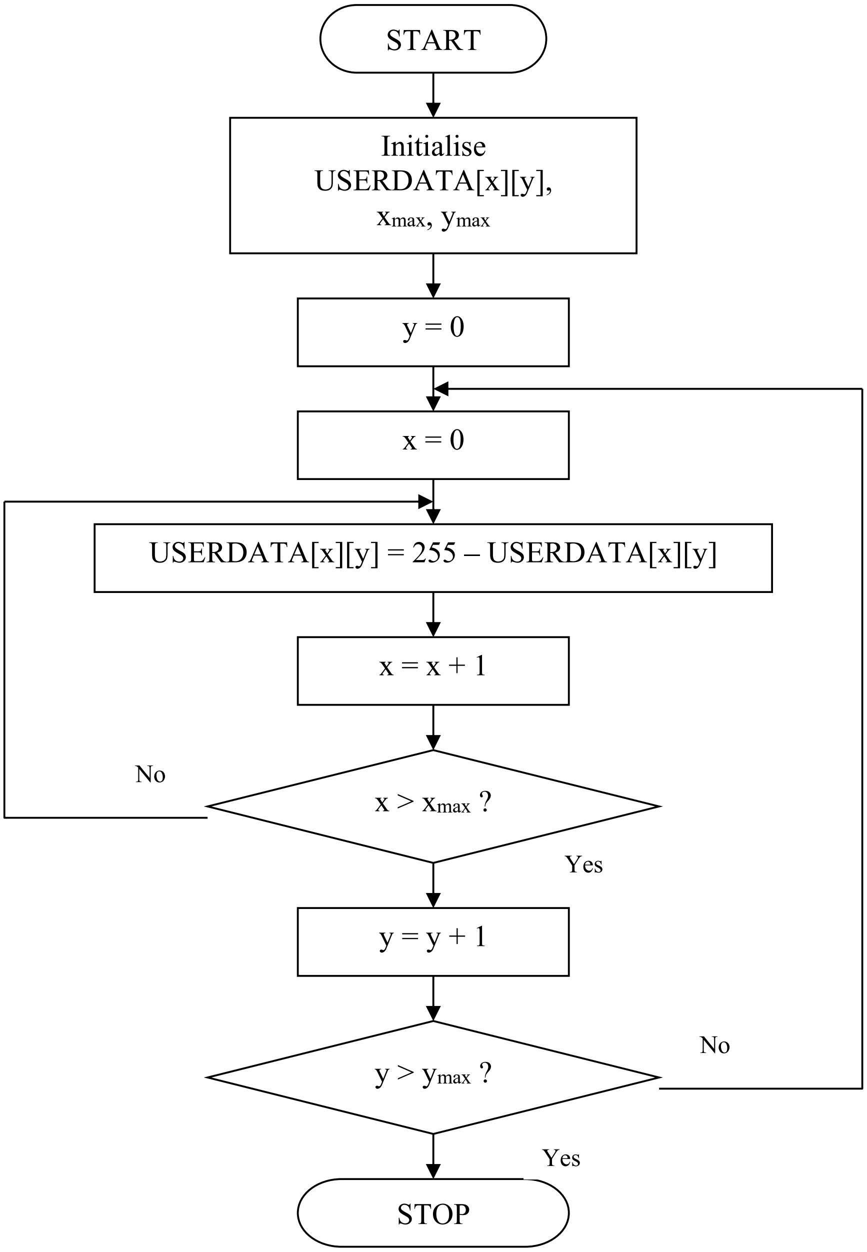
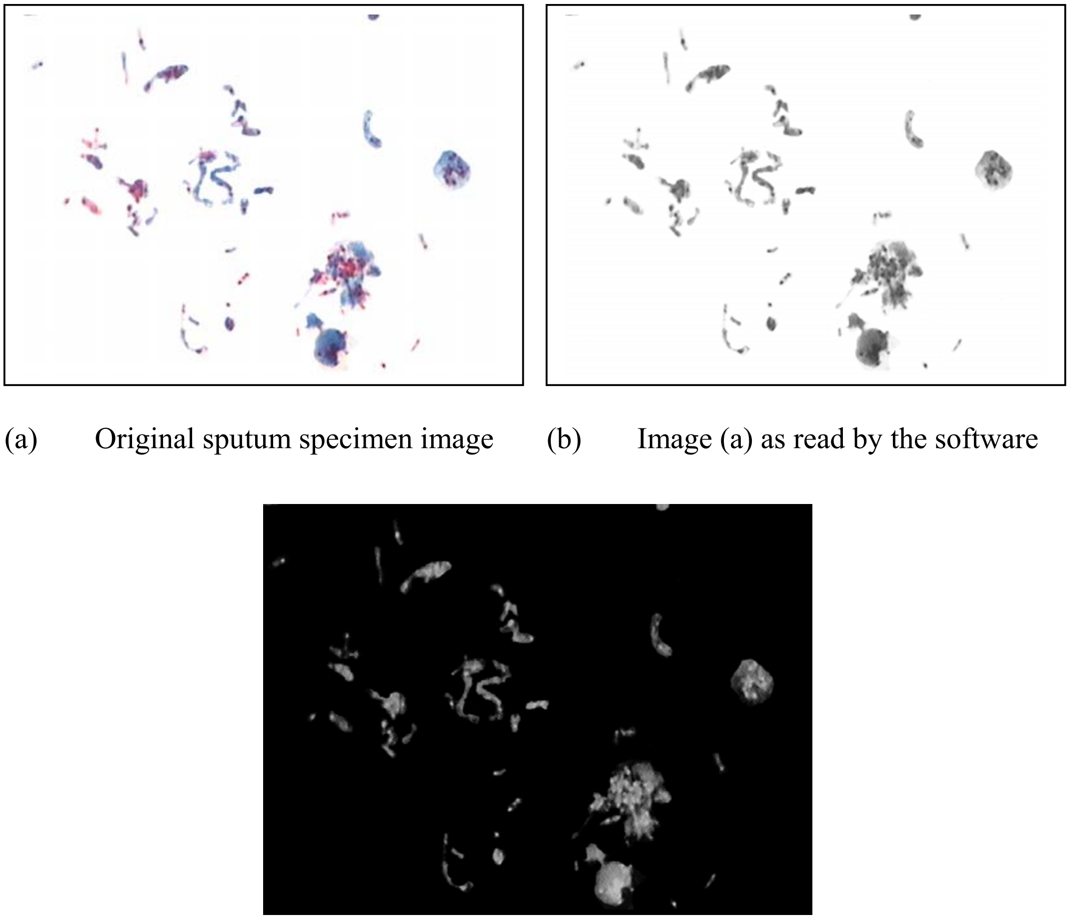

START
Initialise USERDATA[x][y], x_{max}, y_{max}
y = 0
x = 0
USERDATA[x][y] = 255 – USERDATA[x][y]
x = x + 1
No
x > x_{max} ?
Yes
y = y + 1
No
y > y_{max} ?
Yes
STOP

Figure 3.6: Negative transformation algorithm flow chart

Figure 3.7 shows an example of gray level transformation performed by the software. Image (a) is the original image while image (b) is image (a) as read by the software in terms of pixel brightness, where the brightness level is assigned as gray level. Image (c) is the fully transformed result of image (a).



(a) Original sputum specimen image (b) Image (a) as read by the software

(c) Negative-transformed sputum specimen image

Figure 3.7: Sputum specimen image (a) that undergoes gray level transformation (b) followed by negative filtering (c)

3.4 Seeded Region Growing

According to Ooi (2000), seeded region growing involves a seed pixel from which a region is grown with its pixels within a certain range. Firstly, seed pixel is selected at which it is surrounded by neighbouring pixels with a matrices size of P x P, where P is any odd integer. The seed pixel lies right in the centre amongst the neighbouring pixels, as shown in Figure 3.8.

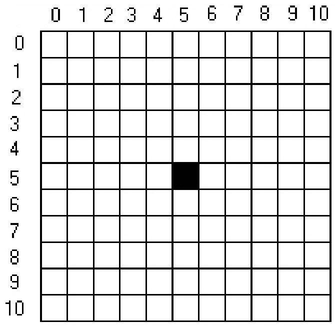
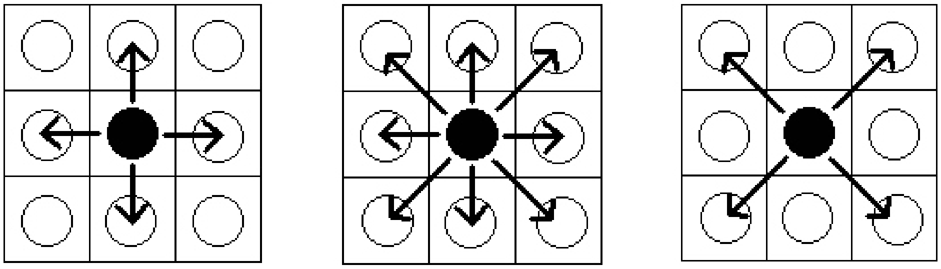


Figure 3.8: Position of a seed pixel surrounded by its neighbours in an 11 x 11 matrices.

Secondly, statistical parameter (such as mean and variance) for the region is computed based on gray level values for all pixels in the matrices. Then, seed point could be grown in the desired direction. The algorithm provides three options of growing directions: four-connected, eight-connected, and four-diagonal as shown in the following Figure 3.9.



(a) four-connected (b) eight-connected (c) four-diagonal

Figure 3.9: Three options for seeded region growing direction

For every growing streak from the seed pixel to its adjacent pixels, a lookup computation is carried out based on gray level value of the adjacent pixels and previously calculated region parameter to decide whether the adjacent pixels will be included into the region or not. If the computation result fulfils a previously set growing condition, then the adjacent pixels are included into the region. Otherwise, the adjacent pixels will be left out of the region.

The process is implemented recursively where newly included pixels will act as new seed pixels. New parameter for the region is recomputed with respect to those newly included pixels. This new parameter value is then used for the next lookup process. The region growing process stops once all pixels have been looked up.

3.4.1 Seeded Region Growing Based on Region Mean

This is the most common approach applied in seeded region growing technique. As the P x P matrices for the initial region is set, mean of the region is then computed using Equation 3.6,

$$\text{Region mean, } \sigma = \frac{\sum_{i=1}^{n} g_i}{n_t} \qquad \textbf{(3.6)}$$

where g_i = gray level value of each individual pixel

n_t = total number of pixel

Next, the seed pixel is grown out towards its neighbours in one of the three directions provided as options. For each growing streak, only absolute values are obtained for the difference between gray level value of neighbouring pixels and the region mean computed previously. As long as the absolute value of difference is less than a previously set threshold value, then the adjacent pixels are included into the region and the process runs recursively against the next pixels until all the pixels have been looked up.

As new pixels are being included into the region, the new mean for the grown region is computed using the updated values of g_i and n_t. The new mean is then used for the next lookup process. The gained region is either marked or circumstanced by a boundary line, depending on the option previously set by the user. Figure 3.10 shows the general algorithm flow chart for seeded region growing.

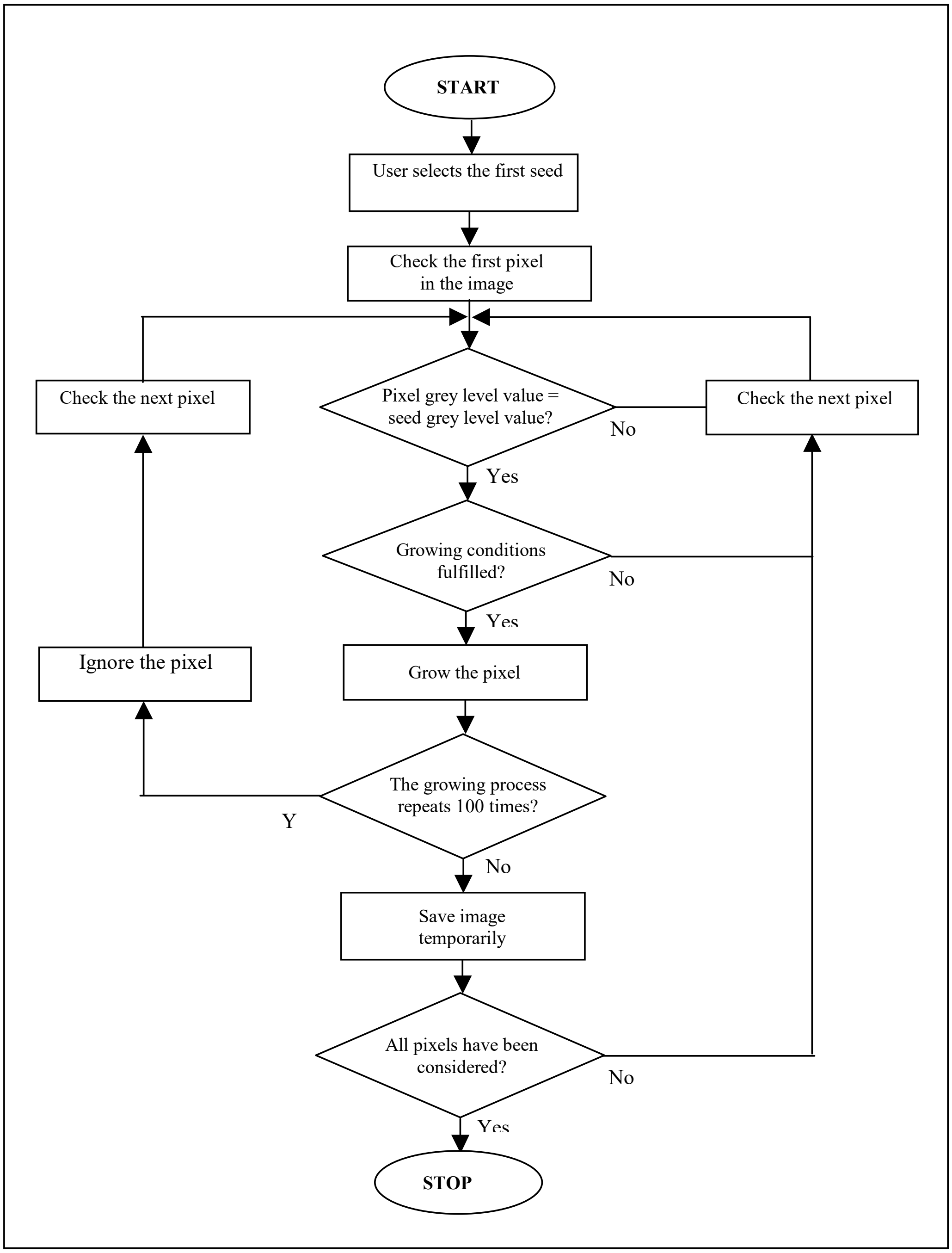
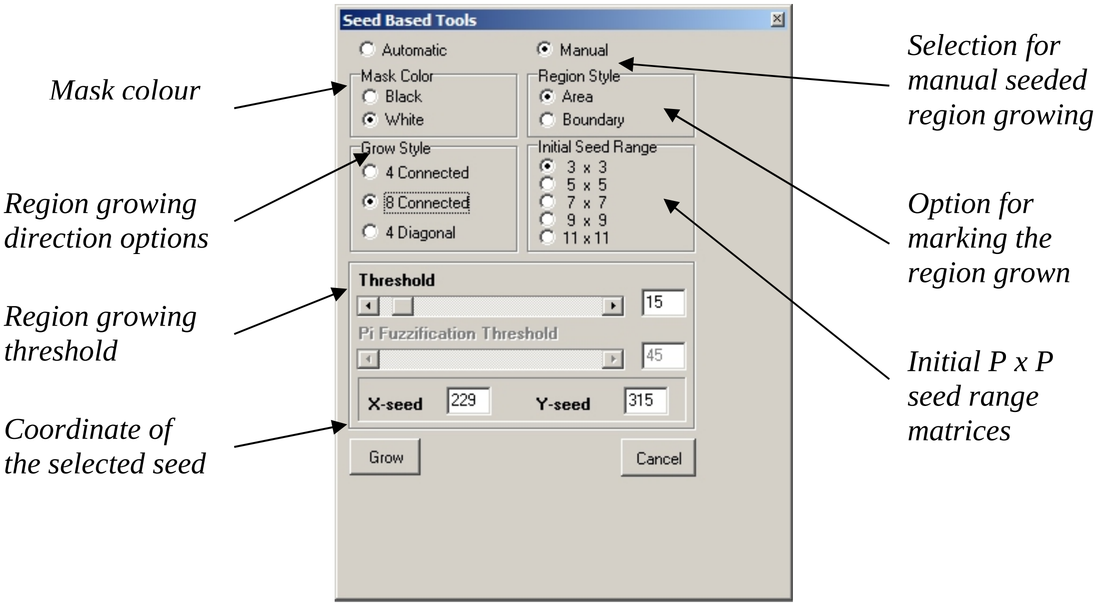

START
User selects the first seed
Check the first pixel in the image
Check the next pixel
Pixel grey level value = seed grey level value?
No
Check the next pixel
Yes
Growing conditions fulfilled?
No
Yes
Ignore the pixel
Grow the pixel
The growing process repeats 100 times?
Y
No
Save image temporarily
All pixels have been considered?
No
Yes
STOP

Figure 3.10: General algorithm flow chart for seeded region growing



Seed Based Tools

○ Automatic ● Manual

Mask Color: ○ Black ● White

Region Style: ● Area ○ Boundary

Grow Style: ○ 4 Connected ● 8 Connected ○ 4 Diagonal

Initial Seed Range: ● 3 x 3 ○ 5 x 5 ○ 7 x 7 ○ 9 x 9 ○ 11 x 11

Threshold: 15

Pi Fuzzification Threshold: 45

X-seed: 229 Y-seed: 315

Mask colour

Region growing direction options

Region growing threshold

Coordinate of the selected seed

Selection for manual seeded region growing

Option for marking the region grown

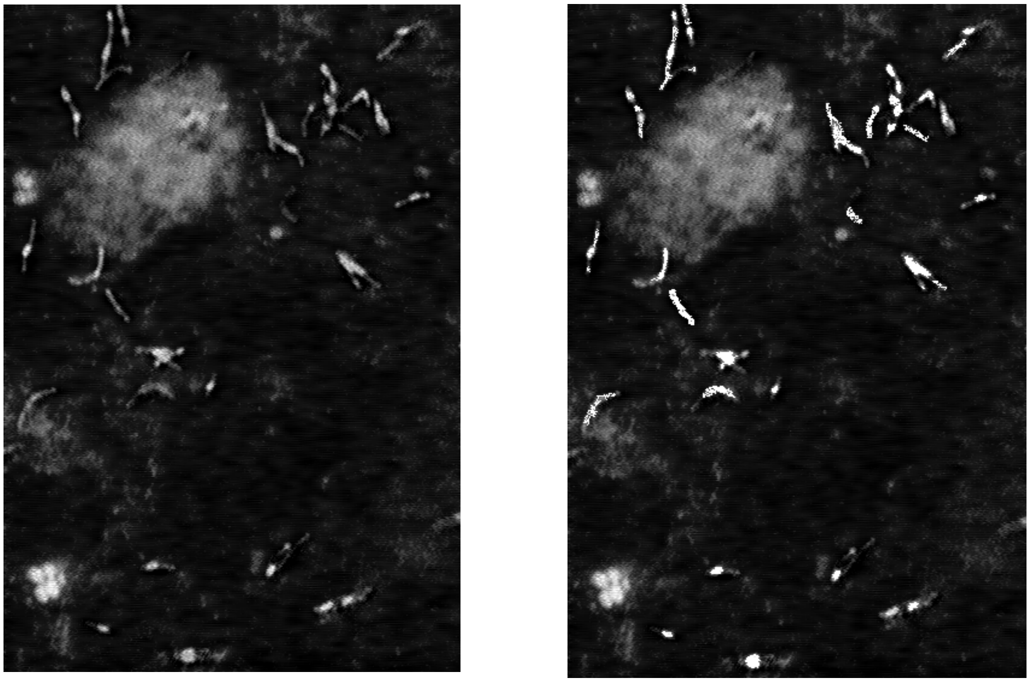
Initial P x P seed range matrices

Figure 3.11: Seeded region growing settings

The panel shown in Figure 3.11 above is the setting for manual seeded region growing. Options are provided for ***mask colour***, ***region marking style***, ***grow direction***, and ***initial seed pixel range matrices***. The ***region growing threshold*** option needs to be adjusted depending on the brightness of the selected seed pixel. A higher threshold value should be selected for brighter seed pixels while a lower threshold value is suitable for darker seed pixels.

Initial seed pixel range refers to the array range, which the seed pixel would be grown. The range 3 x 3 implies that each growing streak shall consider eight adjacent pixels to the current seed pixel. The same range shall be considered for the next growing streaks from new seed pixels. Meanwhile, growing using the range 5 x 5 will consider 24 adjacent pixels to the current seed pixels in every growing streak and the same range is considered for every new seed pixels. Iteratively, selecting the range 7 x 7 shall have 48 adjacent pixels to be considered in every growing streak. The same rule applies for the larger initial seed ranges where the use of 9 x 9 range shall consider 80 adjacent pixels to the seed for every growing streak and the range 11 x 11 shall consider 120 adjacent pixels.

An example of manual seeded region growing is shown in Figure 3.12. Image (a) is a negative-transformed specimen image. Seeded region growing is performed manually against each visible *Mycobacterium tuberculosis* organisms and as the result, clearer *Mycobacterium tuberculosis* clusters are developed as shown in image (b). Hence, seeded region growing is proven as an efficient technique to enhance clusters in poor quality specimen images prior to cluster analysis and calculation.



(a) Negative image with unclear clusters (b) Clearer clusters developed

Figure 3.12: Performance of manual seeded region growing for developing clearer organism clusters prior to counting.

3.4.2 Automatic Region Growing and Cluster Analysis

In addition to conventional seeded region growing, the software features an option for automatic region growing and cluster analysis. Instead of having to select seed pixels manually, the algorithm automatically detects for suitable seed pixels. As quoted by Ng (2001), this automatic algorithm features fuzzy theory set for “on-the-fly” image enhancement. Besides, Marr-Hildreth filter is also applied to improve the image contrast. Sequentially, the algorithm checks for suitable seed pixels. Then, using region mean-based seeded region growing, the algorithm grows the identified seed pixels.

The automatic region growing and cluster analysis process flow is described with the process flow chart shown in Figure 3.13.

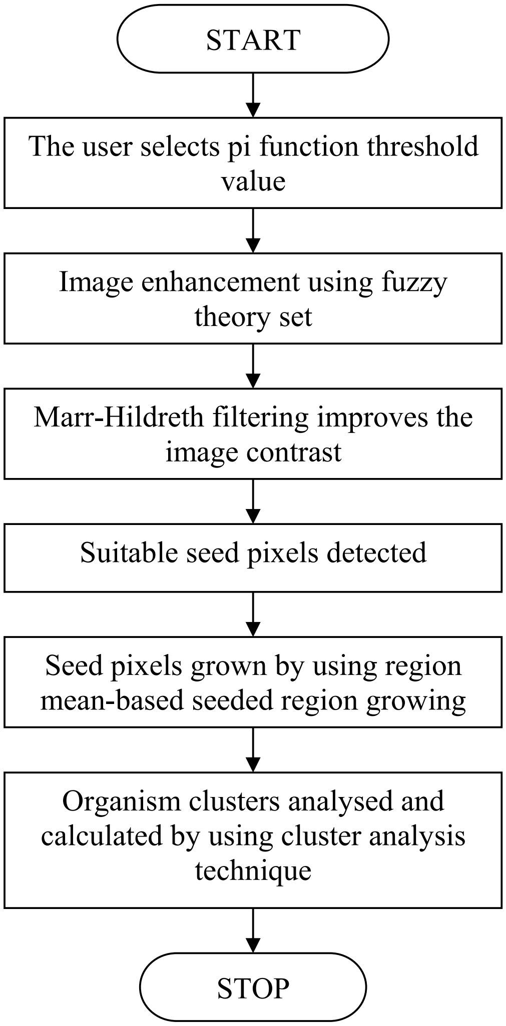
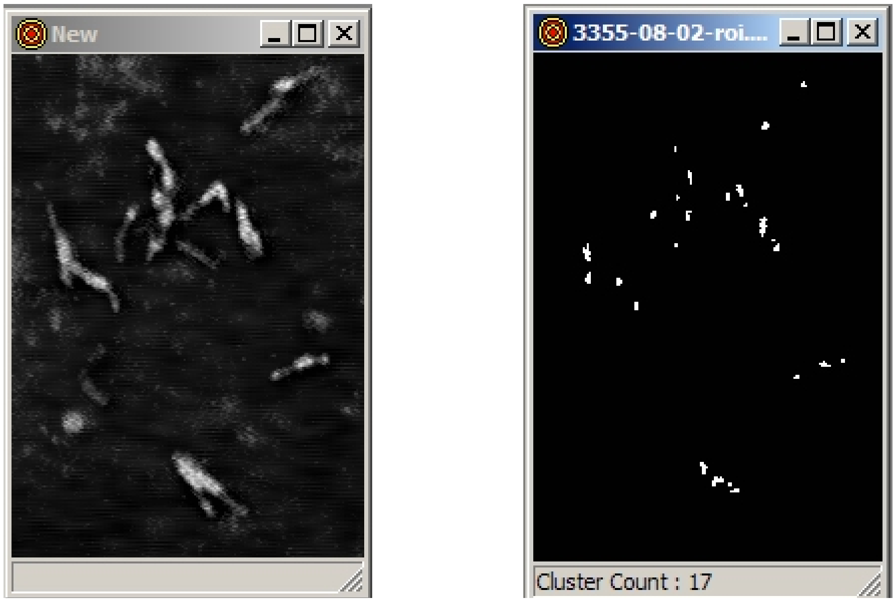

START
The user selects pi function threshold value
Image enhancement using fuzzy theory set
Marr-Hildreth filtering improves the image contrast
Suitable seed pixels detected
Seed pixels grown by using region mean-based seeded region growing
Organism clusters analysed and calculated by using cluster analysis technique
STOP

Figure 3.13: Process flow chart of automatic region growing and cluster analysis

The following Figure 3.14 shows an example of automatic region growing and cluster analysis. Image (a) is a region of interest selected from a specimen image. With a certain adjusted pi function threshold value and one single click, the software automatically enhance and filters the image, identifies seed pixel, performs region growing, and analyses the developed clusters. Image (b) is the result of automatic region growing with the pi function threshold set to 80.



Cluster Count : 17

(a) Original image

(b) Clusters developed with automatic region growing

Figure 3.14: Automatic region growing and cluster analysis