GMIT
GALWAY-MAYO INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO

Development of a Semi – Automatic Self Learning Pattern Recognition System for the Control of Electronic Boards

In One Volume

Vlad Teleanca

June 2002

Submitted for the Degree of
Master in Science

Submitted to:	Galway Mayo Institute of Technology, Ireland
Research carried out at:	Galway Mayo Institute of Technology, Ireland
Research Supervisor:	Dr. Patrick DELASSUS

Acknowledgments

I would like to express my gratitude to the following people and organizations for their support and assistance in developing this project:

Patrick, my supervisor, a sincere word of thanks for your assistance, ideas, guidance and constructive criticism. I have enjoyed our time working together.

All my appreciation for Galway – Mayo Institute of Technology and Tyco/Mallinckrodt for giving me the opportunity to realise this project.

Abstract

This project was funded under the Applied Research Grants Scheme administered by Enterprise Ireland. The project was a partnership between Galway – Mayo Institute of Technology and an industrial company, Tyco/Mallinckrodt Galway. The project aimed to develop a semi – automatic, self – learning pattern recognition system capable of detecting defects on the printed circuits boards such as component vacancy, component misalignment, component orientation, component error, and component weld.

The research was conducted in three directions: image acquisition, image filtering/recognition and software development. Image acquisition studied the process of forming and digitizing images and some fundamental aspects regarding the human visual perception. The importance of choosing the right camera and illumination system for a certain type of problem has been highlighted. Probably the most important step towards image recognition is image filtering. The filters are used to correct and enhance images in order to prepare them for recognition. Convolution, histogram equalisation, filters based on Boolean mathematics, noise reduction, edge detection, geometrical filters, cross – correlation filters and image compression are some examples of the filters that have been studied and successfully implemented in the software application. The software application developed during the research is customized in order to meet the requirements of the industrial partner. The application is able to analyze pictures, perform the filtering, build libraries, process images and generate log files. It incorporates most of the filters studied and together with the illumination system and the camera it provides a fully integrated framework able to analyze defects on printed circuit boards.

TABLE OF CONTENTS

1 Introduction

1.1 Thesis motivation

The numerical processing of the images is a field with a fast evolution and the area of application is extending to more and more human activities. Starting with military technology or high security systems and ending with now, the ordinary bar – code reader in a supermarket or software able to perform optical character recognition, the image processing has applications in arts, health sector, industry, science, or agriculture. The fast progress in the past few years in electronics, optics, or software development has made it possible to use digital images as a mean to improve work efficiency. If it is to diagnose, treat or understand the abnormalities in the internal structures of the human body in the medical field; use satellite data to analyze and compare crops or vegetation in agriculture or forestry fields; detect flaws on circuit boards or leaks on fuel tanks in industry; detect and count targets and their orientation in military field; identify persons by retina or fingerprint scan for security systems or just create new images and virtual realities in arts and computer games, image processing is the answer.

Not all the applications targeted by image processing require the same approach, thus the techniques that involve image manipulation can be divided in several categories: image enhancement, segmentation, compression or recognition. For example a medic might be helped to highlight in a radiography image those aspects that are important from his point of view. This can be done by image enhancement using procedures like filtering, histogram equalization, magnification or color enhancement. Large amounts of pictures received from satellites scanning the extraterrestrial space are compressed in order to save space and this way, easing the transmission, and then, regions of interest are extracted for further processing using image segmentation. But one of the most spectacular applications of image processing is, no doubt, the recognition of shapes. Actual progress in this field allows the recognition of a finite number of shapes. Notable achievements have been obtained for the recognition of the written text, classification of fingerprints, autonomic car guidance or target recognition (i.e. planes or tanks for the military field).

The examples presented are far from being complete; the list of institutions and persons involved in this field is continuously growing.

1.2 Aims and objectives of the thesis

Mallinckrodt Galway manufactures ventilatory supply systems for the healthcare sector. The ventilatory systems mainly include a very advanced gas pump system as well as a control and command system. The control and command system includes a relatively complex electronic board. The company has some difficulties in measuring the quality of these electronic boards. At present, the boards, supplied by other companies, are one hundred percent visually controlled by an operator. This control is very slow and is still not one hundred percent reliable.

The objective of the project is to develop a semi – automatic, self – learning pattern recognition system capable of detecting defects such as component vacancy, component misalignment, component orientation, component error, and component weld. Once the system is operational for a given board, the next part of the project was to generalise the pattern recognition system for any type of electronic boards supplied and used in the company.

The development of such a system is strategically important to Mallinckrodt for improving the manufacturing lead time of the unit and reducing the cost due to non – quality.

1.3 Approach to work

The research began with a study of the requirements made by the industrial partner, Mallinckrodt Galway. At that time, the requirements and needs of the research were outlined. The project continued with a literature review and, at the same time, a study and evaluation of different software packages that target this field. As a result, a set of rules have been developed, the software package Matlab from Mathworks was chosen to develop the filters and Visual C++ from Microsoft as a platform to build the user interface of the application.

For the next step, an illumination system was built and a suitable digital camera chosen in order to create an image acquisition system. The longest part of the research was the software development that included decision making programming and system integration. The project was concluded with a testing and validation phase that was divided in three sections: reliability testing, dynamic testing and improvement.

1.4 Thesis structure

The thesis is structured as follows:

- ***Chapter One*** – presents the motivation, the objectives, the approach to work and the layout of the thesis.
- ***Chapter Two*** – describes the filters that form the basis for image processing, for example, noise reduction, image enhancement through histogram equalization or contrast stretching, normalized cross – correlation and smoothing filters.
- ***Chapter Three*** – studies image morphology, a collection of optimized methods of processing and analyzing shapes in binary and grey scale images.
- ***Chapter Four*** – illustrates how the images are formed, how different illumination techniques are able to substantially reduce the processing tasks and it also brings some examples of different software systems on the market.
- ***Chapter Five*** – is a comprehensive description of the image acquisition and processing system built to analyse and diagnose printed circuit boards supplied by the industrial partner. The description of the system includes the digital camera used to acquire images, illumination system to enhance the images and, most important, the software module that processes the images.
- ***Chapter Six*** – summarises and draws conclusions from the work carried out in the thesis and presents recommendations for further development.

Note that chapters two, three and four represent the coverage of literature studied, while chapter five illustrates the results of the research as described above. Figure 1.4 summarises the thesis structure.

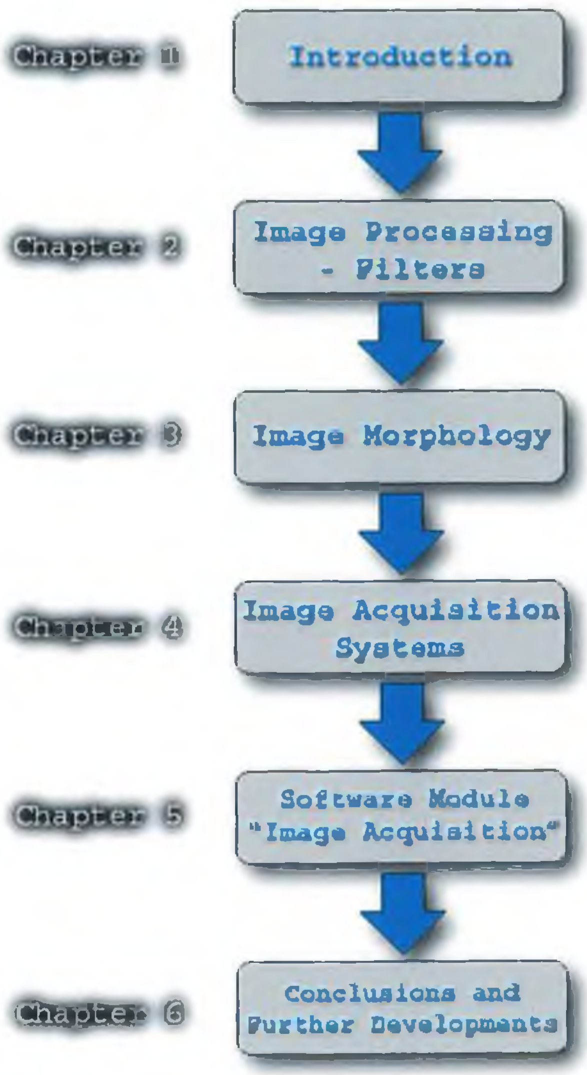

Chapter 1
Introduction
Chapter 2
Image Processing - Filters
Chapter 3
Image Morphology
Chapter 4
Image Acquisition Systems
Chapter 5
Software Module "Image Acquisition"
Chapter 6
Conclusions and Further Developments

Figure 1.4: Thesis structure

2 Image Processing – Filters

2.1 Introduction

The aim of this chapter is to describe filters and operations that are fundamental to digital image processing. These operations can be divided into several categories. The operations based on convolution are very important for the formation of analogue and digital images; the histogram operations are important when one wishes to compare two images acquired in different circumstances; the operations based on mathematics provide a fast and easy way to manipulate images; smoothing filters are important as they provide a convenient way to reduce the noise in images and also prepare them for further processing; the derivative based operations are the background for more advanced filters such as edge detection; cross – correlation filters are used for template matching and are a standard approach to feature detection; the geometric operations allow the pictures to be rotated, scaled or translated. These operations can be described in terms of their implementation as a point operation (filters based on mathematics), a local operation (convolution), or a global operation (histogram equalization).

Note that all the examples presented in this chapter have been realized using the Image Processing Toolbox contained in Matlab.

2.2 *Operations based on convolution*

The convolution is considered a local operation because an element of the output picture (a pixel) is the result of the combination of those input elements localized in an area of the image that has small dimensions in comparison with the dimension of the whole image. The basic idea is that a window of some finite size and shape – the support – is scanned across the image. The output pixel value is the weighted sum of the input pixels within the window where the weights are the values of the filter assigned to every pixel of the window itself. The window with its weights is called the convolution kernel. There are many mathematical interpretations of this process, some notable contributions that highlight the importance of convolution in image processing have been presented by Young [Young, 1995], Brigham [Brigham 1988] or Oppenheim [Oppenheim 1975]. Considering the filter h[j, k] is zero outside the (rectangular) window {j = 0,1,...,J–1; k=0,1,...,K–1}, then, the convolution can be written as the following finite sum:

$$c[m,n] = a[m,n] \otimes h[m,n] = \sum_{j=0}^{J-1}\sum_{k=0}^{K-1} h[j,k]a[m-j,n-k]$$

This equation can be viewed as more than just a reliable mechanism for smoothing or sharpening an image. Further, the equation illustrates the local character of this operation and suggests that the operation can be implemented through the use of the Fourier domain which requires a global operation – the Fourier transform. In a variety of image forming systems an appropriate model for the transformation of the physical signal a(x, y) into an electronic signal c(x, y) is the convolution of the input signal with the impulse response of the sensor system. This system might consist of both an optical as well as an electrical system. If each of these systems can be treated as a **L**inear, **S**hift – **I**nvariant (LSI) system then the convolution model is appropriate. The definitions of these two possible system properties are given below:

i) If $a_1 \rightarrow c_1$ and $a_2 \rightarrow c_2$, *linearity* can be defined as:

$$w_1 \bullet a_1 + w_2 \bullet a_2 \rightarrow w_1 \bullet c_1 + w_2 \bullet c_2$$

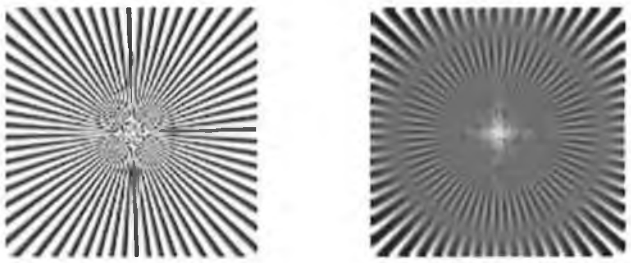
ii) If $a(x,y) \to c(x,y)$, *shift – invariance* can be defined as:

$$a(x-x_0, y-y_0) \to c(x-x_0, y-y_0)$$

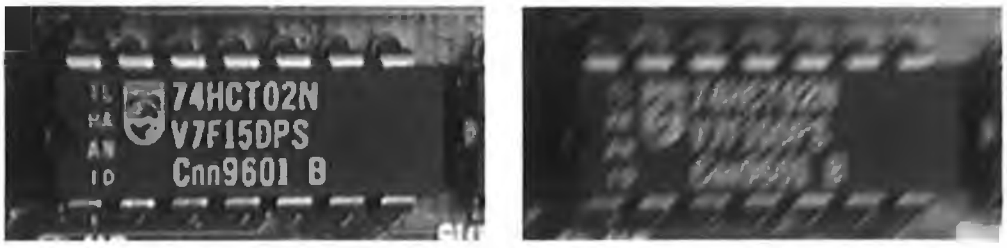
Where w_1 and w_2 are arbitrary complex constants and x_0 and y_0 are coordinates corresponding to arbitrary spatial translations. Two remarks are appropriate at this point. First, linearity implies (by choosing $w_1 = w_2 = 0$) that "zero input" gives "zero output". The offset (x_0, y_0) described in equation means that this kind of camera signals are not the output of a linear system and thus the convolution result is not applicable. Fortunately, it is straightforward to correct for this non – linear effect. Second, optical lenses with a magnification, M, other than *1x* are not shift invariant; a translation of one unit in the input image *a(x, y)* produces a translation of M units in the output image *c(x, y)* but, this case can still be handled by linear system theory.

If an impulse point of light *d(x, y)* is imaged through an LSI system then the impulse response of that system is called the **P**oint **S**pread **F**unction (PSF). The output image then becomes the convolution of the input image with the PSF. The Fourier transform of the PSF is called the **O**ptical **T**ransfer **F**unction (OTF). For optical systems that are circularly – symmetric, aberration – free and diffraction–limited the PSF is given by the Airy disk. The Airy disk is the image of a bright point object as focused by a lens system. With monochromatic light, it consists of a central point of maximum intensity surrounded by alternate circles of light and darkness caused by the reinforcement and interference of diffracted rays. The light areas are called maxima and the dark areas minima. The distribution of light from the center to the outer areas of the figure was first investigated mathematically by Sir George Airy. The diffraction disk forms a basis for determining the resolving power of an ideal lens system. The diameter of the disk depends largely on the aperture of the lens. The diffraction of light causing the Airy disk is a factor limiting the resolution of a well corrected optical system [Chapman, 1991].

If the convolution window is not the diffraction–limited PSF of the lens but rather the effect of defocusing a lens then an appropriate model for h(x, y) is a pill box. The effect on a test pattern is illustrated in Figure 2.2.



a) Test pattern b) Defocused image (radius of 4.5 pixels)



c) Test pattern d) Defocused image (radius of 7.5 pixels)

Figure 2.2: Convolution of some test patterns

The effect of the defocusing is more than just simple blurring or smoothing. The almost periodic negative lobes in the transfer function produce a 180 degrees phase shift in which black turns to white and vice – versa. The phase shift is clearly visible in Figure 2.2b. The reverse of this technique can be used to enhance blurred images.

2.3 Operations based on histogram

An important class of point operations is based upon the manipulation of an image histogram or a region histogram. The most important examples are described below.

2.3.1 Contrast stretching

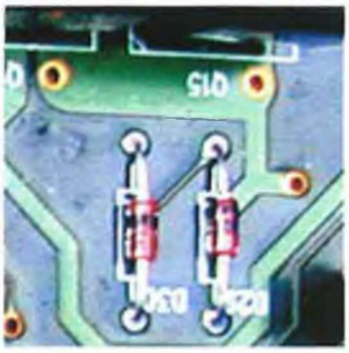
Frequently, an image is scanned in such a way that the resulting brightness values do not make full use of the available dynamic range. This can be easily observed in the histogram of the brightness values. By stretching the histogram over the available dynamic range attempt to correct this situation [Sahoo, 1998]. If the image is intended to go from brightness *0* to brightness 2^B-1, then one generally maps the *0%* value (or minimum) to the value *0* and the *100%* value (or maximum) to the value 2^B-1. The appropriate transformation is given by:

$$b[m,n] = \left(2^B-1\right)\bullet\frac{a[m,n]-\min}{\max-\min}$$

However, this formula can be somewhat sensitive to external influences and a less sensitive and more general version is given by:

$$b[m,n]=\begin{cases} 0 & a[m,n]\le p_{low}\% \\ \left(2^B-1\right)\bullet\dfrac{a[m,n]-p_{low}\%}{p_{high}\%-p_{low}\%} & p_{low}\%<a[m,n]\le p_{high}\% \\ \left(2^B-1\right) & a[m,n]\le p_{high}\% \end{cases}$$

In this second version someone might choose the *1%* and *99%* values for $p_{low}\%$ and $p_{high}\%$, respectively, instead of the *0%* and *100%* values represented by first equation. It is also possible to apply the contrast – stretching operation on a regional basis using the histogram from a region to determine the appropriate limits for the algorithm. Note that in equations it is possible to suppress the term 2^B-1 and simply normalize the brightness range to $0\le b[m,n]\le 1$. This means representing the final pixel brightness as real instead of integers but modern CPU's speeds and memory capacities make this quite feasible. Figure 2.3.1 shows a sample picture (2.3.1a) which was contrast stretched (2.3.1b) using the equation above:


a) sample picture

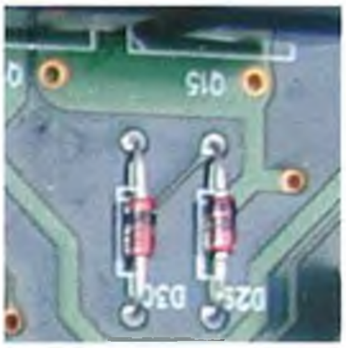

b) contrast stretched

Figure 2.3.1: Contrast stretching

The values used in equation are:

- a[m,n] – sample picture
- b[m,n] – contrast stretched picture
- m = 176; n = 178 (pixels)
- $2^B-1 = 0.45$ (contrast stretching factor)

2.3.2 *Equalization*

When one wishes to compare two or more images on a specific basis, such as texture, it is common to first normalize their histograms to a "standard" histogram. This can be especially useful when the images have been acquired under different circumstances. The most common histogram normalization technique is histogram equalization where one attempts to change the histogram through the use of a function $b = f(a)$ into a histogram that is constant for all brightness values. This would correspond to a brightness distribution where all values are equally probable. Unfortunately, for an arbitrary image, the result can only be approximated. Remarkable descriptions of various equalization techniques have been done by Otsu [Otsu, 1979], Kitter [Kitter, 1986], Cho [Cho, 1989] or Pratt [Pratt, 1991].

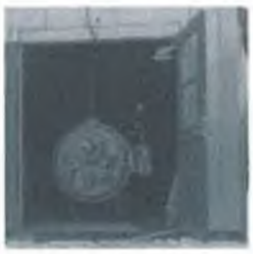
For a "suitable" function *f(*)* the relation between the input probability density function, the output probability density function, and the function *f(*)* is given by [Pratt, 1991]:

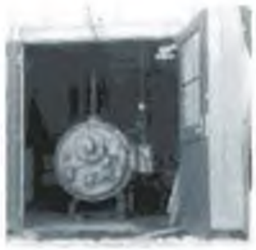
$$p_b(b)db = p_a(a)da \Rightarrow df = \frac{p_a(a)da}{p_b(b)}$$

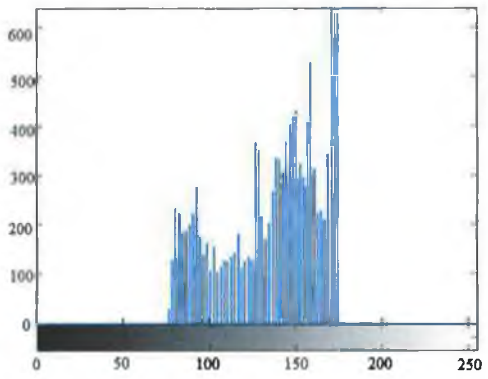
From equation it can be seen that "suitable" means that *f (*)* is differentiable and that $\frac{df}{da} \geq 0$. For histogram equalization is desirable that $p_b(b)$ = *constant* and this means that:

$$f(a) = (2^B - 1) \bullet P(a)$$

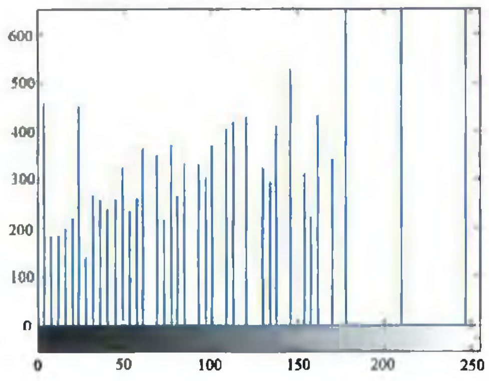
Where *P(a)* is the probability distribution function. In other words, the quantized probability distribution function normalized from 0 to 2^B–1 is the look–up table required for histogram equalization. Figure 2.3.2 illustrates the effect of histogram equalization on a standard image. The histogram equalization procedure can also be applied on a regional basis [Otsu, 1979].


a) Original


b) Histogram Equalized



c) Light distribution in original image;



d) Light distribution in equalized image

Figure 2.3.2: Histogram equalization

In figure 2.3.2c it can be observed that the light has an uneven distribution across the picture; it is concentrated on in a small area. The effect of equalization (figure 2.3.2d) is that the light is constant distributed an all areas of the picture.

2.3.3 Other histogram–based operations

The histogram derived from a local region can also be used to drive local filters that are to be applied to that region. Examples include minimum filtering, median filtering, and maximum filtering.

2.4 Operations based on mathematics

A distinction has to be made in this section between binary arithmetic and ordinary arithmetic. In the binary case there are two brightness values "0" and "1". In the ordinary case we begin with 2^B brightness values or levels but the processing of the image can easily generate many more levels. For this reason many *software* systems provide 16 or 32 bit representations for pixel brightness in order to avoid problems with arithmetic overflow.

2.4.1 Binary operations

Operations based on binary (Boolean) arithmetic form the basis for a powerful set of tools that will be described here and in the next chapter. The operations described below are point operations and thus admit a variety of efficient implementations including simple look–up tables. The standard notation for the basic set of binary operations is:

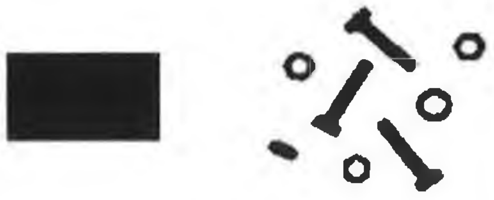
$$\begin{array}{ll}
NOT & c = \overline{a} \\
OR & c = a + b \\
AND & c = a \bullet b \\
XOR & c = a \oplus b = a \bullet \overline{b} + \overline{a} \bullet b \\
SUB & c = a \setminus b = a - b = a \bullet \overline{b}
\end{array}$$

The implication is that each operation is applied on a pixel–by–pixel basis, for example, $c[m,n] = a[m,n] \bullet \overline{b}[m,n]$. The definition of each operation is described in table 2.4.1:

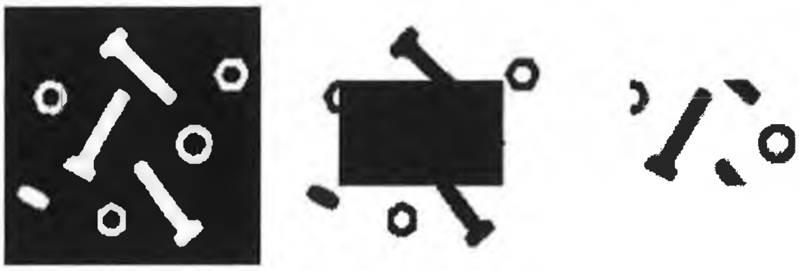
Operation	Input		Output
	a	*b*	
NOT	0	–	1
	1	–	0
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0
AND	1	1	1
	1	0	0
	0	1	0
	0	0	0
XOR	1	1	0
	1	0	1
	0	1	1
	0	0	0
SUB	1	1	0
	1	0	1
	0	1	0
	0	0	0

Table 2.4.1: Binary operations

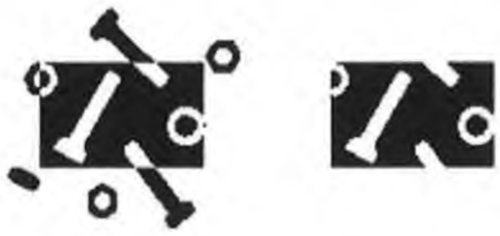
These operations are illustrated in Figure 2.4.1 where the binary value "1" is shown in black and the value "0" in white.



a) Image *a* **b)** Image *b*



c) NOT $(b) = \bar{b}$ **d)** OR $(a, b) = a + b$ **e)** AND $(a, b) = a \bullet b$



f) $XOR(a,b) = a \oplus b$ **g)** SUB $(a, b) = a \setminus b$

Figure 2.4.1: Examples of the various binary point operations.

The SUB (\) operation can be particularly useful when the image *a* represents a region–of–interest that has to be analyzed and the image *b* represents objects that, having been analyzed, can now be discarded from the region.

2.4.2 *Arithmetic–based operations*

The grey – value operations, very popular in image processing, are based on ordinary mathematics and include:

Operation	Definition	Preferred Data Type
ADD	$c = a + b$	*integer*
SUB	$c = a - b$	*integer*
MUL	$c = a * b$	*integer or floating point*
DIV	$c = a / b$	*floating point*
LOG	$c = log(a)$	*floating point*
EXP	$c = exp(a)$	*floating point*
SQRT	$c = sqrt(a)$	*floating point*
TRIG.	$c = sin(a), cos(a)$	*floating point*
INVERT	$c = (2^B - 1) - a$	*integer*

Table 2.4.2: Mathematical operations

2.5 *Operations based on smoothing*

These algorithms are applied in order to reduce noise and (or) to prepare images for further processing such as segmentation. A distinction has been made between linear and non – linear algorithms where the linear filters are open to Fourier analysis while the non – linear are not. Also, a distinction has been made between rectangular and circular shaped filters.

2.5.1 Linear Filters

This section will present several linear filtering algorithms together with some sample images.

a. ***Uniform filter*** – The output image is based on a local averaging of the input filter where all of the values within the filter support have the same weight. For the discrete spatial domain [*m,n*] the filter values are the samples of the continuous domain case. Examples for the rectangular case (*j = k = 5*) and the circular case (*r* = 2.5) are shown in Figure 2.5.1.1.

$$h_{rect}[j,k]=\frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix} \qquad h_{circ}[j,k]=\frac{1}{21}\begin{bmatrix}0&1&1&1&0\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\0&1&1&1&0\end{bmatrix}$$

a) Rectangular filter (*j = k = 5*) b) Circular filter (*r = 2.5*)

Figure 2.5.1.1: Image smoothing – uniform filters

Note that in both cases the filter is normalized so that $\Sigma h[j, k] = 1$. It has been done this way so that, if the input *a*[*m*, *n*] is a constant, then the output image *c*[*m*, *n*] is the same constant. Attention has to be paid as both of these filters have transfer functions that have negative lobes and therefore can lead to phase reversal as seen in Figure 2.2. The effect of the filter on a sample image is presented in figure 2.5.1.2.

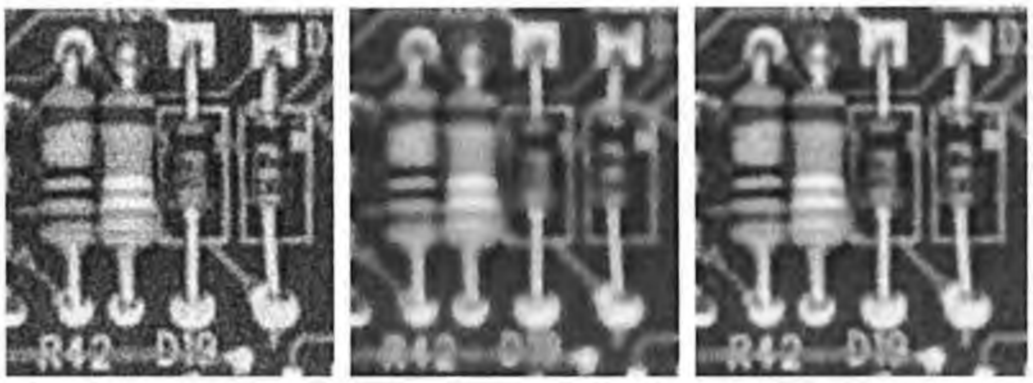

a) Noisy image b) Rectangular filter c) Circular Filter

Figure 2.5.1.2: Uniform filters samples

b. ***Triangular filter*** – The output image is based on a local averaging of the input filter where the values within the filter support have differing weights. In general, the filter can be seen as the convolution of two (identical) uniform filters either rectangular or circular and this has direct consequences for the computational complexity. The transfer functions of these filters do not have negative lobes and thus do not imply phase reversal.

Examples for the rectangular (pyramidal) support case (*j* = *k* = *5*) and the circular (conical) support case (*r* = *2.5*) are shown in Figure 2.5.1.3. The filter is again normalized so that $\Sigma h[j, k]=1$.

$$h_{circ}[j,k]=\frac{1}{25}\begin{bmatrix}0&0&1&0&0\\0&2&2&2&0\\1&2&5&2&1\\0&2&2&2&0\\0&0&1&0&0\end{bmatrix} \qquad h_{rect}[j,k]=\frac{1}{81}\begin{bmatrix}1&2&3&2&1\\2&4&6&4&2\\3&6&9&6&3\\2&4&6&4&2\\1&2&3&2&1\end{bmatrix}$$

a) Pyramidal filter (*j* = *k* =*5*) b) Conical filter (*r* = *2.5*)

Figure 2.5.1.3: Triangular filters for image smoothing

An implementation of the triangular filter is shown in figure 2.5.1.4.

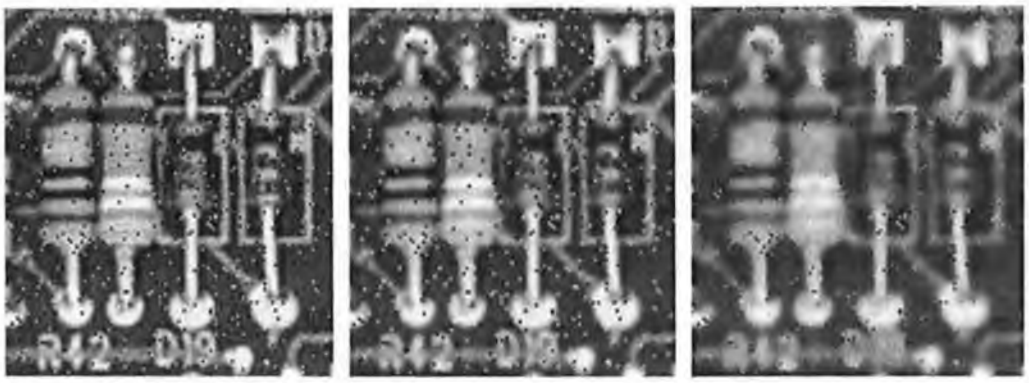

a) Noisy image b) Pyramidal filter c) Conical Filter

Figure 2.5.1.4: Triangular filters for image smoothing

c. ***Gaussian filter*** – In image processing, the use of the Gaussian kernel is a popular method for smoothing. This is because of certain properties of the Gaussian (i.e. the central limit theorem) as well as several application areas such as edge detection. The result of the Gaussian filter on a sample image is shown in figure 2.5.1.5.

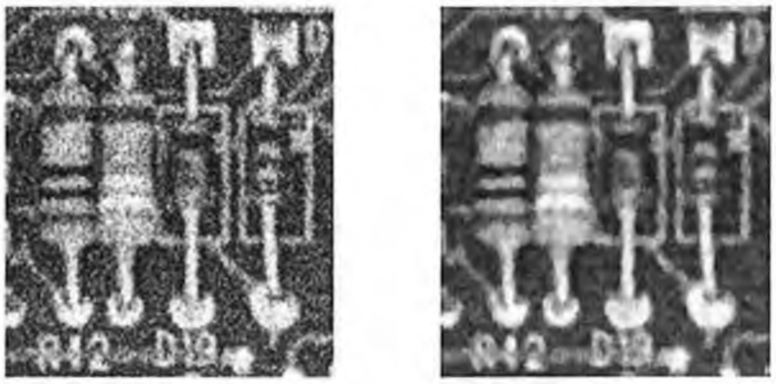

a) Noisy image b) Gaussian filter

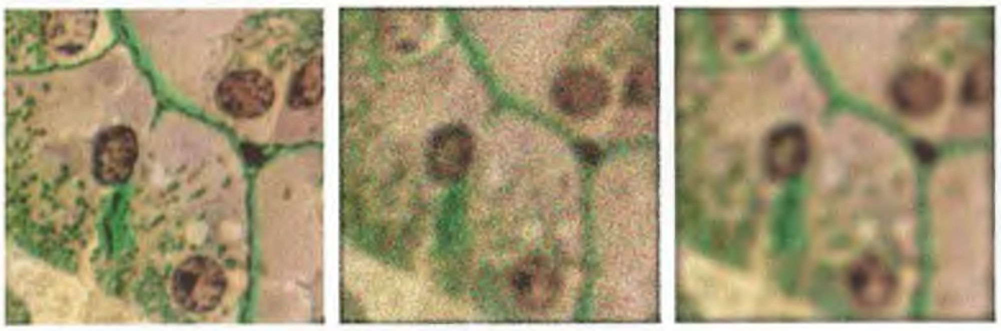
Figure 2.5.1.5: Gaussian filter

2.5.2 Non – linear filters

A large variety of smoothing filters have been developed that are not linear. While they can't, generally, be submitted to Fourier analysis, their properties and domains of application have been studied extensively [Lindeberg, 1994].

a. ***Median filter*** – A median filter is based upon moving a window over an image (as previously discussed, in a convolution) and computing the output pixel as the median value of the brightness within the input window. If the window is $j \bullet k$ in size, we can order the $j \bullet k$ pixels in brightness value from smallest to largest. If $j \bullet k$ is odd then the median will be the $(j \bullet k+1)/2$ entry in the list of ordered brightness. Note that the value selected will be exactly equal to one of the existing brightness so that no round–off error will be involved if we want to work exclusively with integer brightness values. The algorithm as it is described above has a generic complexity per pixel of $O(j \bullet k \bullet \log(j \bullet k))$. Fortunately, there is a fast algorithm that reduces the complexity to $O(K)$ assuming $j \geq k$.

A useful variation on the theme of the median filter is the *percentile filter*. Here the center pixel in the window is replaced not by the 50% (median) brightness value but rather by the *p*% brightness value where *p*% ranges from 0% (the *minimum filter*) to 100% (the *maximum filter*). Values other then *p*=50% do not, in general, correspond to smoothing filters.



a) Original image b) Noise added c) Restored image

Figure 2.5.2.1 Median Filter

b. ***Kuwahara filter*** – Edges play an important role in our perception of images as well as in the analysis of images. As such it is important to be able to smooth images without disturbing the sharpness and, if possible, the position of edges. A filter that accomplishes this goal is termed an *edge – preserving filter* and one particular example is the Kuwahara filter. Although this filter can be implemented for a variety of different window shapes, the algorithm will be described for a square window of size $J = K = 4L + 1$ where L is an integer. The window is partitioned into four regions as shown in Figure 2.5.2.2.

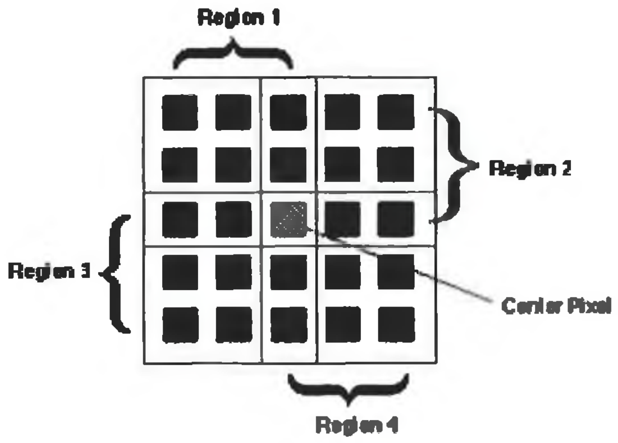
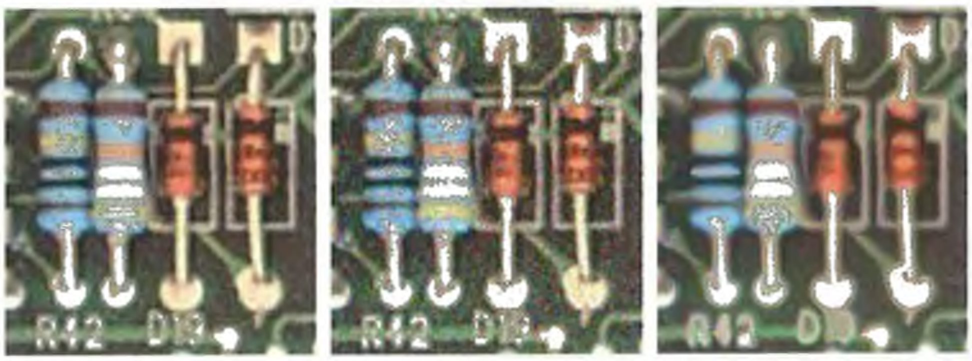


Figure 2.5.2.2: Four square regions defined for the Kuwahara filter.

In this example $L = 1$ and thus $j = k = 5$. Each region is $[(j+1)/2] \bullet [(k+1)/2]$. In each of the four regions ($i = 1, 2, 3, 4$), the mean brightness, m_i, and the variance, s_i^2, are measured. The output value of the centre pixel in the window is the mean value of that region that has the smallest variance.



a) Original image b) Noise added c) Restored image

Figure 2.5.2.3: Kuwahara Filter ($j = k = 5$)

2.6 *Operations based on derivative*

Another important operation in image processing is the ability to take one or more spatial derivatives of an image. An important implementation of the operations based on derivative is the detection of the edges. The filters based on the first derivate work perfect on images where the noise is not present and where the edges are steep (step shaped). In case the edges are smoother, more rounded (i.e. as a result of a linear filter or just the nature of the image), the second derivate is the answer for the edge detection process. The basic problem is that, according to the mathematical definition of a derivative, this is impossible to be done. A digitized image is not a continuous function $f(x, y)$ of spatial variables but, rather a discrete function $f[m, n]$ of integer spatial coordinates. As a result, the algorithms that will be presented can only be seen as approximations to the true spatial derivatives of the original spatially continuous image. This means that high frequency noise will be emphasized in the resulting image. The general solution to this problem is to combine the derivative operation with one that suppresses high frequency noise, in short, smoothing in combination with the desired derivative operation.

2.6.1 *First Derivatives*

As an image is a function of two (or more) variables it is necessary to define the direction in which the derivative is taken. For the two–dimensional case we have the horizontal direction, the vertical direction, or an arbitrary direction which can be considered as a combination of the two. If we use $\mathbf{h}_x$ to denote a horizontal derivative filter (matrix), $\mathbf{h}_y$ to denote a vertical derivative filter (matrix), and $\mathbf{h}_\theta$ to denote the arbitrary angle derivative filter (matrix), then:

$$[h_\theta] = \cos\theta \bullet [h_x] + \sin\theta \bullet [h_y]$$

a. ***Gradient filters*** – It is also possible to generate a vector derivative description as the *gradient,* $\nabla a[m,n]$, of an image:

$$\nabla a = \frac{\partial a}{\partial x}\vec{i}_x + \frac{\partial a}{\partial y}\vec{i}_y = (h_x \otimes a)\vec{i}_x + (h_y \otimes a)\vec{i}_y$$

Where $\vec{i}_x$ and $\vec{i}_y$ are unit vectors in the horizontal and vertical direction, respectively. This leads to two descriptions:

- ***Gradient magnitude:*** $|\nabla a| = \sqrt{(h_x \otimes a)^2 + (h_y \otimes a)^2}$

and

- ***Gradient direction:*** $\psi(\nabla a) = \arctan\left\{ {(h_y \otimes a)} / {(h_x \otimes a)} \right\}$

The gradient magnitude can be approximated by:

- ***Approximate Gradient magnitude:*** $|\nabla a| \approx |h_x \otimes a| + |h_y \otimes a|$

The final results of these calculations depend strongly on the choices of **h**$_x$ and **h**$_y$. A number of possible choices for (**h**$_x$, **h**$_y$) will be described now.

b. ***Basic derivative filters*** – These filters are specified by:

i) $[h_x] = [h_y]^t = [1 \quad -1]$

ii) $[h_x] = [h_y]^t = [1 \quad 0 \quad -1]$

Where "t" denotes matrix transpose. These two filters differ significantly in their Fourier magnitude and Fourier phase characteristics. For the frequency range $0 \le \Omega \le \pi$, these are given by:

$$i)[h]=[1 \quad -1] \quad \overset{F}{\leftrightarrow} \quad |H(\Omega)|=2|\sin(\Omega/2)|; \qquad \varphi(\Omega)=(\pi-\Omega)/2$$
$$ii)[h]=[1 \quad 0 \quad -1] \quad \overset{F}{\leftrightarrow} \quad |H(\Omega)|=2|\sin(\Omega)|; \qquad \varphi(\Omega)=\pi/2$$

The second form (*ii*) gives suppression of high frequency terms ($\Omega \sim \pi$) while the first form (*i*) does not. The first form leads to a phase shift; the second form does not.

c. ***Prewitt gradient filters*** – These filters are specified by:

$$[h_x]=\frac{1}{3}\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix}=\frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix}\bullet[1 \quad 0 \quad -1]$$

$$[h_y]=\frac{1}{3}\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix}=\frac{1}{3}\begin{bmatrix}1\\0\\-1\end{bmatrix}\bullet[1 \quad 1 \quad 1]$$

Both $\mathbf{h}_x$ and $\mathbf{h}_y$ are separable. Beyond the computational implications are the implications for the analysis of the filter. Each filter takes the derivative in one direction using equation *(ii)* and smoothes in the orthogonal direction using a one dimensional version of a *uniform* filter. The following pictures illustrate edge detection using Prewitt gradient filters:

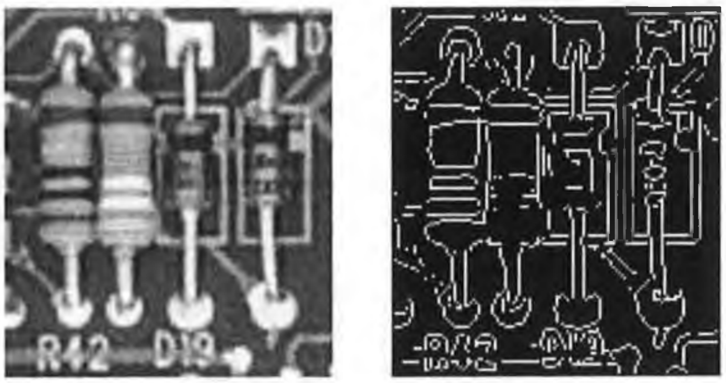

a) Original picture b) Edge detection

Figure 2.6.1.1: Prewitt gradient filter

d. ***Sobel gradient filters*** – These filters are specified by:

$$[h_x]=\frac{1}{4}\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}=\frac{1}{4}\begin{bmatrix}1\\ 2\\ 1\end{bmatrix}\bullet[1 \quad 0 \quad -1]$$

$$[h_y]=\frac{1}{4}\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}=\frac{1}{4}\begin{bmatrix}1\\ 0\\ -1\end{bmatrix}\bullet[1 \quad 2 \quad 1]$$

Again, $\mathbf{h}_x$ and $\mathbf{h}_y$ are separable. Each filter takes the derivative in one direction using equation *(ii)* and smoothes in the orthogonal direction using a one dimensional version of a *triangular* filter. An example of the Sobel gradient filter in presented in figure 2.6.1.2

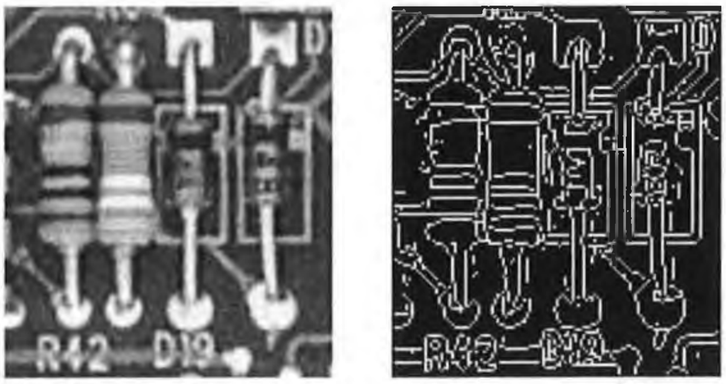
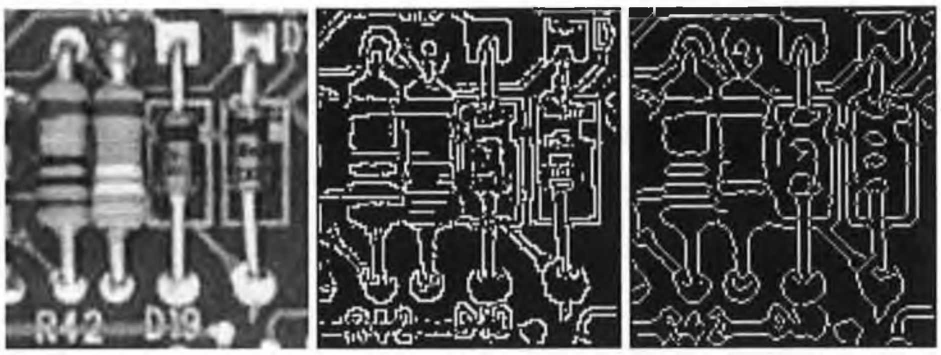

a) Original picture b) Edge detection

Figure 2.6.1.2: Sobel gradient filter

e. ***Alternative gradient filters*** – The variety of techniques available from one dimensional signal processing for the design of digital filters offers us powerful tools for designing one dimensional versions of $\mathbf{h}_x$ and $\mathbf{h}_y$. Using the Parks – McClellan filter design algorithm [Selesnick, 1997], for example, we can choose the frequency bands where we want the derivative to be taken and the frequency bands where we want the noise to be suppressed. The algorithm will then produce a real, odd filter with a minimum length that meets the specifications. Figure 2.6.1.3 presents the effects of Canny and Roberts gradient filters on a sample image.



a) Original picture b) Roberts filter c) Canny filter

Figure 2.6.1.3: Alternative gradient filter

2.6.2 *Second Derivatives*

Although it is possible to compute higher order derivatives of functions of two variables, in image processing, the second derivatives or Laplacian play an important role. The Laplacian is defined as:

$$\nabla^2 a = \frac{\partial^2 a}{\partial x^2} + \frac{\partial^2 a}{\partial y^2} = (h_{2x} \otimes a) + (h_{2y} \otimes a)$$

Where $\mathbf{h_{2x}}$ and $\mathbf{h_{2y}}$ are second derivative filters. The following section will show some implementation of the Laplacian.

a. ***Basic second derivative filter*** – This filter is specified by:

$$[h_{2x}] = [h_{2y}] = [1 \quad -2 \quad 1]$$

And the frequency spectrum of this filter, in each direction, is given by:

$$H(\Omega) = F\{1 \quad -2 \quad 1\} = -2(1 - \cos\Omega)$$

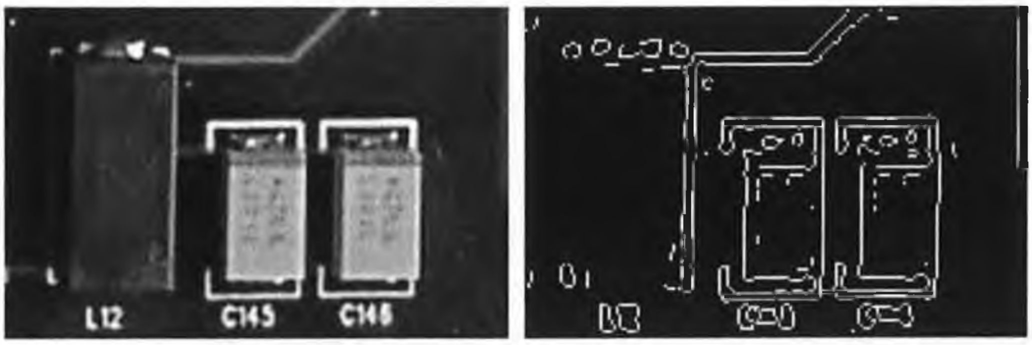
Over the frequency range $-\pi \le \Omega \le \pi$. The two, one dimensional filters can be used in the manner suggested by equation or combined into one, two–dimensional filter as follows:

$$[h] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

b. ***Frequency domain Laplacian*** – This filter is the implementation of the general solution given in equation and for the Laplacian filter takes the form:

$$c[m,n] = F^{-1}\left\{-\left(\Omega^2 + \Psi^2\right)A(\Omega,\Psi)\right\}$$

Figure 2.6.2.1 shows the effect of the filter on a sample image:



a) Sample image b) Laplacian

Figure 2.6.2.1: Laplacian filter

c. ***SDGD filter*** – A filter that is used especially in edge finding and object measurement is the *Second–Derivative–in–the–Gradient–Direction (SDGD)* filter. This filter uses five partial derivatives:

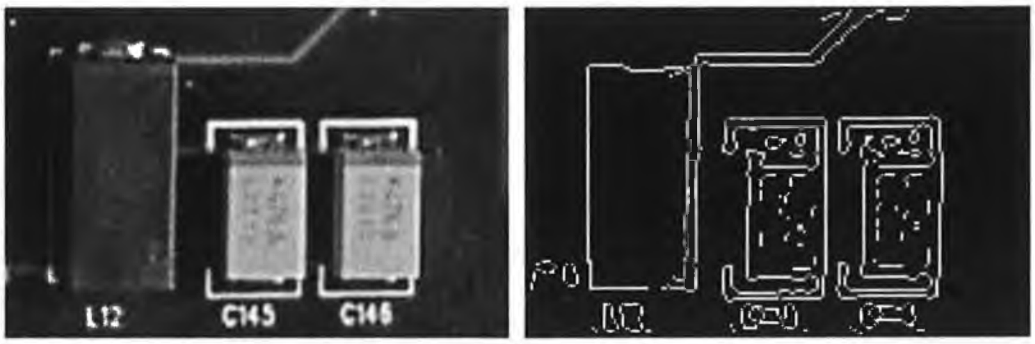
$$A_{xx} = \frac{\partial^2 a}{\partial x^2} \quad A_{xy} = \frac{\partial^2 a}{\partial x \partial y} \quad A_x = \frac{\partial a}{\partial x}$$
$$A_{yx} = \frac{\partial^2 a}{\partial x \partial y} \quad A_{yy} = \frac{\partial^2 a}{\partial y^2} \quad A_y = \frac{\partial a}{\partial y}$$

It is worth to note that $A_{xy} = A_{yx}$, which accounts for the five derivatives.

This *SDGD* combines the different partial derivatives as follows:

$$SDGD(a) = \frac{A_{xx}A_x^2 + 2A_{xy}A_xA_y + A_{yy}A_y^2}{A_x^2 + A_y^2}$$

As it might be expected, the large number of derivatives involved in this filter implies that noise suppression is important and that Gaussian derivative filters – both first and second order – are highly recommended if not required. It is also necessary that the first and second derivative filters have essentially the same passbands and stopbands. This means that if the first derivative filter h_{1x} is given by [1 0 –1] (equation ***ii***) then the second derivative filter should be given by $h_{1x} \otimes h_{1x} = h_{2x} =$ [1 0 –2 0 1]. An implementation of this filter is presented in figure 2.6.2.2.



a) Sample image b) SDGD filter

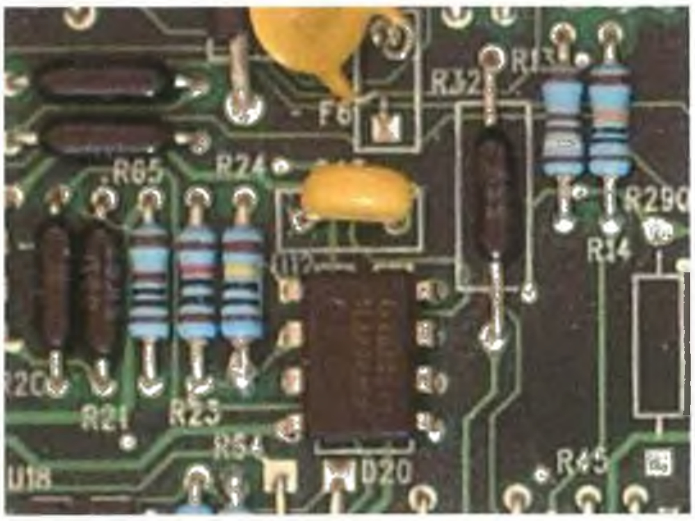
Figure 2.6.2.2: SDGD filter

2.7 *Operations based on normalized cross – correlation*

The cross – correlation based algorithms has been successfully used to detect patterns on a search window (template matching). The correlation between two signals (cross correlation) is a standard approach to feature detection as well as a component of more sophisticated techniques [Duda, 1973]. However, it is quite clear that cross – correlation is not the ideal approach to feature tracking since it is not invariant with respect to imaging scale, rotation, and perspective distortions [Gonzalez, 1992]. The normalized cross – correlation filter is based on the formula:

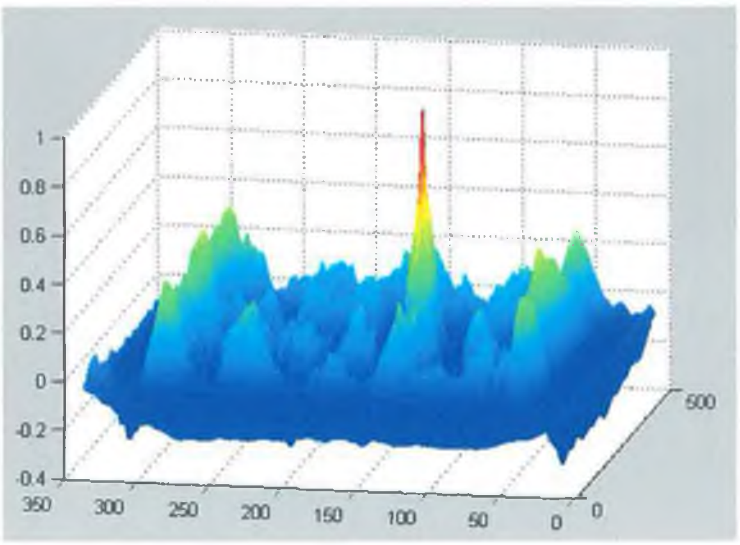
$$C = \frac{\sum_m \sum_n \left(A_{m,n} - \overline{A}\right)\left(B_{m,n} - \overline{B}\right)}{\sqrt{\left(\sum_m \sum_n \left(A_{m,n} - \overline{A}\right)^2\right)\left(\sum_m \sum_n \left(B_{m,n} - \overline{B}\right)^2\right)}}$$

where A is the main picture, B is the pattern picture, C result image, m and n are the dimensions of the pictures, $\overline{A}$ is the mean value of A and $\overline{B}$ is the mean value of B. Some sample pictures tested in Matlab show the efficiency of the algorithm:



a) Main Picture

b) Test Pattern

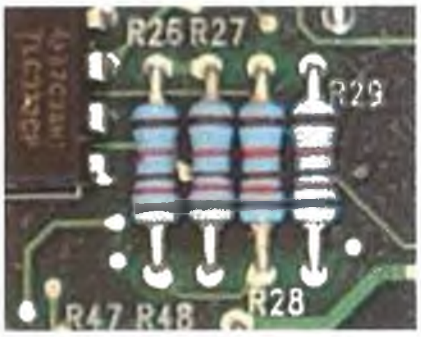


c) Filter result

Figure 2.7.1: Normalized cross – correlation

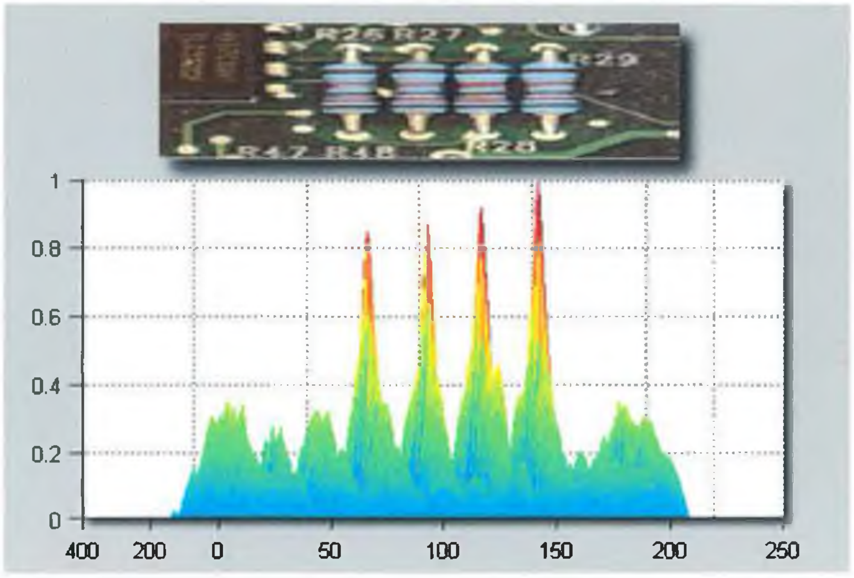
In figure 2.7.1c, on *x* axis is the width of the picture (*m* as in equation), on y is the height of the picture (*n* as in equation) and on *z* axis on a scale from –1 to 1 is pointed where the pattern was found. The peak has a value of 0.98, while the rest (considered noise) reach values of up to 0.2.

Another example is shown in figure 2.7.2, where multiple patterns are found using the normalized cross – correlation algorithm:



a) Main Picture

b) Test Pattern



c) Filter result

Figure 2.7.2: Normalized cross – correlation for multiple patterns

Figure 2.7.2a was stretched on top of the diagram in figure 2.7.2c for visualization purposes. The pattern (resistor) was found four times and the peaks have values in range 0.84 – 0.99 while the noise doesn't reach higher values than 0.37.

2.8 Operations based on geometry

In many imaging systems, detected images are subject to geometric distortion introduced by perspective irregularities given by the position of the device that acquires the images (i.e. camera). Applying a geometric transformation to a uniformly distorted image can correct for a range of perspective distortions by transforming the measurements from the ideal coordinates to those that are actually used.

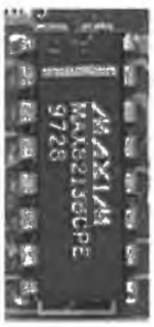
2.8.1 Rotation

The rotation operator performs a geometric transform which maps the position (x_1, y_1) of a pixel element in an input image onto a position (x_2, y_2) in an output image by rotating it through a user specified angle θ about an origin O. In most implementations, output locations (x_2, y_2) which are outside the boundary of the image are ignored. Rotation is most commonly used to improve the visual appearance of an image, although it can be useful as a preprocessor in applications where directional operators are involved [Horn, 1986]. For example, some filters (i.e. edge detection or dilation) may be applied only along a limited set of directions (0, 45, 90… etc) so the rotation may be used to correct this problem (i.e. rotate – apply filter – rotate back).

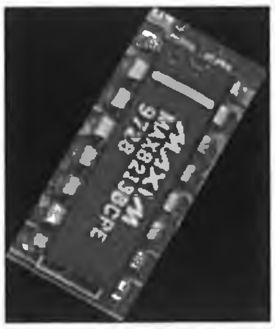
The output pixels (x_2, y_2) of a rotated image have the following relation with the input pixels (x_1, y_1):

$$x_2 = \cos(\theta) \bullet (x_1 - x_0) - \sin(\theta) \bullet (y_1 - y_0) + x_0$$
$$y_2 = \sin(\theta) \bullet (x_1 - x_0) - \cos(\theta) \bullet (y_1 - y_0) + y_0$$

where (x_0, y_0) are the coordinates of the center of rotation, in the input image, and θ is the angle of rotation. In most cases, the rotation operation produces output locations which do not fit within the boundaries of the image (figure 2.8.1), as defined by the dimensions of the original input image. In such cases, destination elements which have been mapped outside the image are ignored by most implementations. The extra pixels resulted from rotation are usually filled in with black pixels.



a) Sample picture



b) Rotated image ($\theta = 30^\circ$)

Figure 2.8.1: Rotation

The rotation algorithm can produce coordinates which are not integers. In order to generate the intensity of the pixels at each integer position, different re-sampling techniques may be employed [Ballard, 1982]. For example:

- ***Nearest neighbor*** – Allows the intensity level at each integer pixel position to assume the value of the nearest non – integer neighbor.
- ***Weighted average*** – Calculate the intensity level at each integer pixel position based on a weighted average of the *n* nearest non – integer values. The weighting is proportional to the distance or pixel overlap of the nearby projections.

Note that the second technique might substantially increase the computation time of the algorithm, but also produces better results.

2.8.2 Scaling

The scale operator performs a geometric transformation which can be used to *shrink* or *zoom* the size of an image (or part of an image). Image reduction, commonly known as *sub – sampling,* is performed by replacement of a group of pixel values by one arbitrarily chosen pixel value from within this group or by *interpolating* between pixel values in a local neighborhoods. Image zooming is achieved by *pixel replication* or by extrapolation. Scaling is used to change the visual appearance of an image or to alter the quantity of information stored in a scene representation.

- ***Shrink*** – The following pictures illustrates two methods of sub – sampling. In the first, one pixel value within a local neighborhood is chosen to be representative of its surroundings. This method is computationally simple, but can lead to poor results if the sampling neighborhoods are too large. The second method interpolates between pixel values within a neighborhood by taking the mean value of the local intensity pixels.

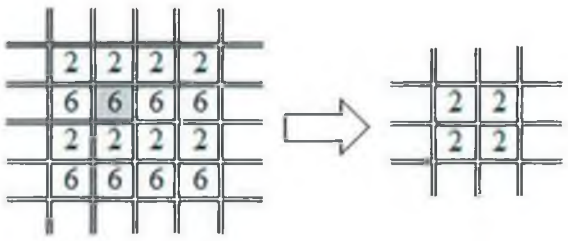


Figure 2.8.2.1: Shrinking by replacement

In the figure above, the highlighted pixel is replaced by the upper – left pixel.

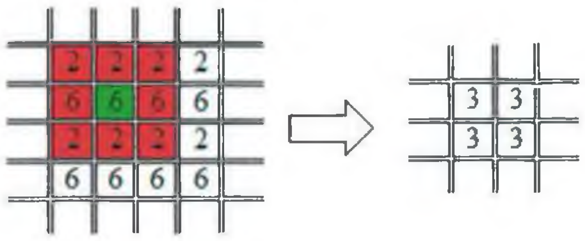


Figure 2.8.2.2: Shrinking by interpolation using mean values

This time the highlighted pixel (green) is interpolated with pixel values in a local neighborhood (red).

- ***Zoom*** – An image, or regions of an image, can be zoomed either through pixel replication or extrapolation. Next figures present how pixel replication simply replaces each original image pixel by a group of pixels with the same value. The group size is determined by the scaling factor. Alternatively, extrapolation of the values of neighboring pixels in the original image can be performed in order to replace each pixel with an expanded group of pixels. Most implementations offer the option of increasing the actual dimensions of the original image, or retaining them and simply zooming a portion of the image within the old image boundaries.

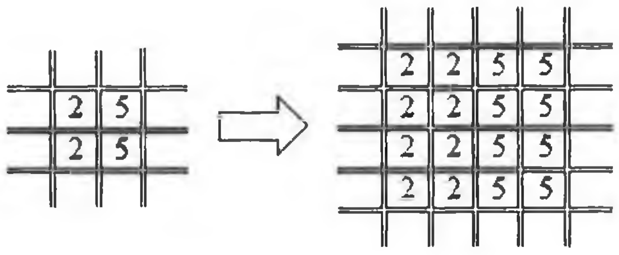


Figure 2.8.2.3: Zooming by replication

As it can bee seen in figure above, zooming by replication simply copies the pixels values in the new figure.

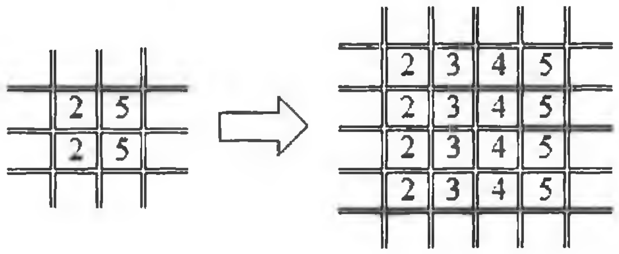


Figure 2.8.2.4: Zooming by extrapolation

Zooming by extrapolation adds pixels that have intermediate values of the original pixels.

Image sampling is a fundamental operation of any image processing system. In a digital computer, images are represented as arrays of finite size numbers. The size of the image array is determined by the number of sampling points measured. The information at each sampling position in the array represents the average irradiance over the small sampling area and is *quantized* into a finite number of bits. The *resolution* of the image increases as we increase the number of sampling and quantization levels. However, large values for these parameters also increase the image storage space and processing requirements. Therefore, the decision as to how to set these parameters must involve striking a compromise between these competing objectives [Jain 1986].

2.8.3 Mirroring

The *mirror* (or reflection) operator geometrically transforms an image such that image elements (pixel values), located at position (x_1, y_1) in an original image are reflected about a user – specified image *axis* or image *point* into a new position (x_2, y_2) in a corresponding output image.

Some examples are given below:

- Reflection about a vertical axis of offset x_0 in the input image (flip horizontal):

$$x_2 = -x_1 + 2 \bullet x_0$$
$$y_2 = y_1$$

- Reflection about a horizontal axis of ordinate y_0 (flip vertical):

$$x_2 = x_1$$
$$y_2 = -y_1 + 2 \bullet y_0$$

- Reflection about an axis oriented in any arbitrary direction θ, and passing through (x_0, y_0):

$$x_2 = x_1 + 2 \bullet \Delta \bullet (-\sin(\theta))$$
$$y_2 = y_1 + 2 \bullet \Delta \bullet \cos(\theta)$$

where $\Delta = (x_1 - x_0) \bullet \sin(\theta) - (y_1 - y_0) \bullet \cos(\theta)$

Note that if (x_0, y_0) are not in the center of the input image, part of the image will be reflected out of the visible range of the image. Most implementations fill in image areas out of which pixels have been reflected with black pixels.

- The horizontal and vertical reflection about a point (x_0, y_0) in the input image is given by (flip vertical and horizontal):

$$x_2 = -x_1 + 2 \bullet x_0$$

$$y_2 = -y_1 + 2 \bullet y_0$$

2.9 Conclusions

The purpose of all the image enhancement techniques is to accentuate some characteristics that are important (such as contrast or edges). In the same time the filters try to attenuate the irrelevant information in the pictures (noise) from the point of view of the designer. The image enhancement techniques don't necessary make the image more accurate but easier to interpret. The interpretation can be visual (by humans) or automatic (by a computer). In the second case, the process of enhancement is followed by other operations (i.e. recognition). The main topics discussed in this chapter and used by most developers are not classified on a scientific base (there isn't a certain rule to tell which one to choose for a given situation). Usually, when it comes to choose a certain algorithm or filter, the developer relies mainly based on intuition and experience. The biggest difficulty comes when the developer has to build a mathematical model of the problem. The user often raises the problem in terms that may be considered vague or ambiguous even for a formal discussion. Regardless of all these considerations, all the techniques described are based on many ingenious ideas and it is hard to think on a system where they can be completely eliminated.

3 Image Morphology

3.1 Introduction

In biology the word morphology is related to the shape and structure of the plants and animals. In Greek language “morphos” means shape. In mathematics, the shapes are represented by sets. In a binary image, the set of pixels with the logical value “1” define the shapes. Every pixel is identified within the set by its coordinates. In the 2D pictures, the coordinates (x, y) of a pixel belong to a subset from Z^2. For 3D pictures (volumes in medical terminology or sequences of images in a movie) the coordinates are from Z^3 ($(x,y) \in Z^3$). The mathematical morphology gives highly optimized methods of processing and analyzing shapes in binary images. The morphological processing of the color or grey scale images became in the recent time a powerful instrument for the applications that use segmentation, filtering or image compression. A good description of the morphological operations has been done by Serra [Serra, 1982] and Giardina [Giardina, 1998]. The aim of this chapter is to describe the base notions of the morphological field.

3.2 *Definitions*

An alternative definition of an image can be based on the notion that an image consists of a set (or collection) of either continuous or discrete coordinates. In a sense the set corresponds to the points or pixels that belong to the objects in the image. This is illustrated in Figure 3.2.1 which contains two objects or sets ***A*** and ***B***. Note that the coordinate system is required. For the moment we will consider the pixel values to be binary. Further we shall restrict our discussion to discrete space (Z^2).

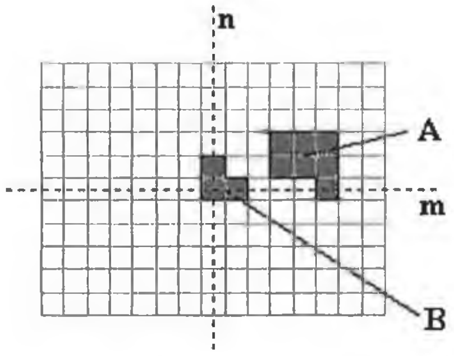


Figure 3.2.1: A binary image containing two object sets *A* and *B*.

The object ***A*** consists of those pixels that share some common property:

- ***Object:*** $A = \{\alpha \mid \alpha = TRUE\}$

As an example, object **B** in Figure 3.2.1 consists of {[0,0], [1,0], [0,1]}.
The background of **A** is given by $\mathbf{A}^c$ (the complement of **A**) which is defined as those elements that are not in **A**:

- ***Background:*** $A^c = \{\alpha \mid \alpha \notin A\}$

It can be now observed that if an object *A* is defined on the basis of *C* – connectivity (*C*=4, 6, or 8) then the background A^c has a connectivity given by 12 – *C*. The necessity for this is illustrated for the Cartesian grid in Figure 3.2.2.

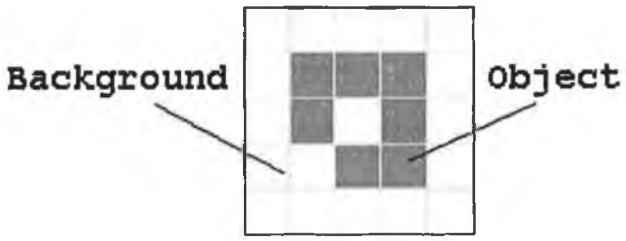


Figure 3.2.2: A binary image requiring careful definition of object and background connectivity.

The basic operations associated with an object are the standard set operations *union*, *intersection*, and *complement* $\{ \cup, \cap, {}^{c} \}$ plus *translation*. Given a vector **x** and a set ***A***, the translation, ***A*** + **x**, is defined as:

- ***Translation:*** $A + x = \{\alpha + x \mid \alpha \in A\}$

Note that, since we are dealing with a digital image composed of pixels at integer coordinate positions (Z^2), this implies restrictions on the acceptable translation vectors **x**. The basic *Minkowski set operations* [Minkowski, 1903], addition and subtraction, can now be defined [Strasser, 1994]. First we note that the individual elements that include ***B*** are not only pixels but also *vectors* as they have a clear coordinate position with respect to [0, 0]. Given two sets ***A*** and ***B***:

- ***Minkowski addition:*** $A \oplus B = \bigcup_{\beta \in B} (A + \beta)$

- ***Minkowski subtraction:*** $A - B = \bigcap_{\beta \in B} (A + \beta)$

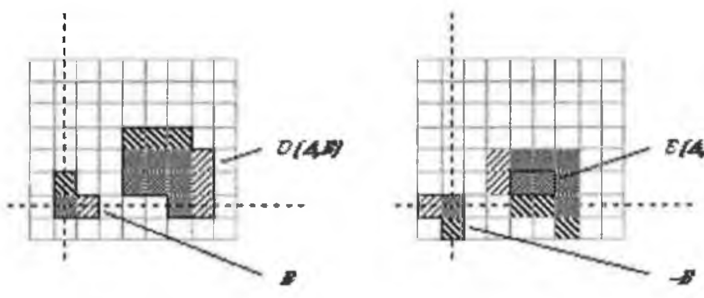
3.3 Dilation and erosion

From the two Minkowski operations defined above, the most important mathematical morphology operations *dilation* and *erosion* can be identified:

- ***Dilation:*** $D(A,B) = A \oplus B = \bigcup_{\beta \in B} (A + \beta)$

- ***Erosion:*** $E(A,B) = A - (-B) = \bigcap_{\beta \in B} (A - \beta)$

Where: $-B = \{-\beta \mid \beta \in B\}$. These two operations are illustrated in Figure 3.3.1 for the objects defined in Figure 3.2.2.

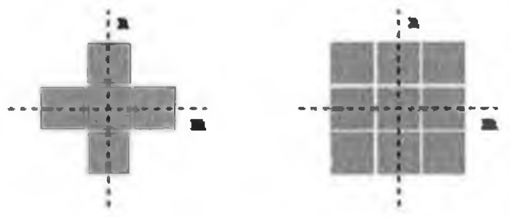


(a) Dilation D(A,B) (b) Erosion E(A,B)

Figure 3.3.1: A binary image containing two object sets A and B. The three pixels in B are "color – coded" as is their effect in the result.

While either set ***A*** or ***B*** can be thought of as an "image", ***A*** is usually considered as the image and ***B*** is called a *structuring element.* The structuring element is to mathematical morphology what the convolution kernel is to linear filter theory.

Dilation, in general, causes objects to dilate or grow in size; erosion causes objects to shrink. The amount and the way that they grow or shrink depend upon the choice of the structuring element. Dilating or eroding without specifying the structural element makes no more sense than trying to lowpass filter an image without specifying the filter [Giardina, 1998]. The two most common structuring elements, given a Cartesian grid, are the 4 – connected and 8 – connected sets (N_4 and N_8). They are illustrated in figure 3.3.2:



(a) N_4 (b) N_8

Figure 3.3.2: The standard structuring elements N_4 and N_8.

Dilation and erosion have the following properties:

- ***Commutative:*** $D(A,B) = A \oplus B = B \oplus A = D(B,A)$
- ***Non–Commutative:*** $E(A,B) \neq E(B,A)$
- ***Associative:*** $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- ***Translation Invariance:*** $A \oplus (B + x) = (A \oplus B) + x$
- ***Duality:*** $E^c(A,B) = D(A^c,-B)$ *and*

  $D^c(A,B) = E(A^c,-B)$

With A as an object and A^c as the background, equation says that the dilation of an object is equivalent to the erosion of the background. Likewise, the erosion of the object is equivalent to the dilation of the background.
Except for special cases:

- ***Non–Inverses:*** $D(E(A,B),B) \neq A \neq E(D(A,B),B)$

Erosion has the following translation property:

- ***Translation Invariance:*** $A-(B+x)=(A+x)-b=(A-B)+x$

Dilation and erosion have also the following important properties. For any arbitrary structuring element $\boldsymbol{B}$ and two image objects $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ such that $\boldsymbol{A}_1 \subset \boldsymbol{A}_2$ ($\boldsymbol{A}_1$ is a proper subset of $\boldsymbol{A}_2$):

- ***Increasing in A:*** $D(A_1,B)\subset D(A_2,B)$

  $E(A_1,B)\subset E(A_2,B)$

For two structuring elements $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ such that $\boldsymbol{B}_1 \subset \boldsymbol{B}_2$:

- ***Decreasing in B:*** $E(A,B_1)\supset E(A,B_2)$

The *decomposition* theorems below make it possible to find efficient implementations for morphological filters.

- ***Dilation:*** $A\oplus(B\bigcup C)=(A\oplus B)\bigcup(A\oplus C)=(B\bigcup C)\oplus A$

- ***Erosion:*** $A-(B\bigcup C)=(A-B)\bigcap(A-C)$

- ***Erosion:*** $(A-B)-C=A-(B\oplus C)$

- ***Multiple Dilations:*** $nB=\underbrace{(B\oplus B\oplus B\oplus\ldots\oplus B)}_{n\ \ times}$

An important decomposition is Vincent's theorem [Vincent, 1991]. First, some definitions are required. A convex set (in R^2) is one for which the straight line joining any two points in the set consists of points that are also in the set. Care must obviously be taken when applying this definition to discrete pixels as the concept of a "straight line" must be interpreted appropriately (in Z^2). A set is *bounded* if each of its elements has a finite magnitude, in this case distance to the origin of the coordinate system. A set is *symmetric* if $\boldsymbol{B} = -\boldsymbol{B}$. The sets $\boldsymbol{N_4}$ and $\boldsymbol{N_8}$ in Figure 3.3.2 are examples of convex, bounded, symmetric sets. Vincent's theorem, when applied to an image consisting of discrete pixels, states that for a bounded, symmetric structuring element $\boldsymbol{B}$ that contains no holes and contains its own centre, $[0,0] \in B$:

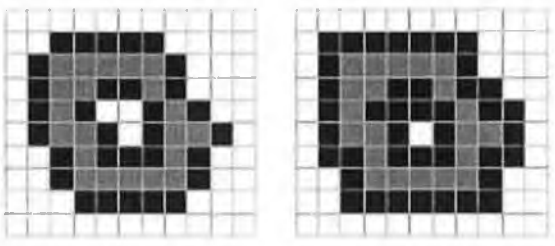
$$D(A,B) = A \oplus B = A \bigcup (\partial A \oplus B)$$

Where ∂A is the contour of the object. Thus, ∂A is the set of pixels that have a background pixel as a neighbor. The implication of this theorem is that it is not necessary to process all the pixels in an object in order to compute a dilation or an erosion; only the boundary pixels have to be processed. This also applies for all operations that can be derived from dilations and erosions. The processing of boundary pixels instead of object pixels means that, apart from the images from the medical field, computational complexity can be reduced from $O(N^2)$ to $O(N)$ for an N x N image. A number of fast algorithms can be found in the literature that are based on this result [Maragos, 1986]. The simplest dilation and erosion algorithms are frequently described as follows:

i) ***Dilation*** – Take each binary object pixel (with value "1") and set all background pixels (with value "0") that are C – connected to that object pixel to the value "1".

ii) ***Erosion*** – Take each binary object pixel (with value "1") that is C – connected to a background pixel and set the object pixel value to "0".

Comparison of these two procedures to equation where $B = N_{C=4}$ or $N_{C=8}$ shows that they are equivalent to the formal definitions for dilation and erosion. The procedure is illustrated for dilation in figure 3.3.3.



(a) $B = N_4$ (b) $B = N_8$

Figure 3.3.3: Illustration of dilation. Original object pixels are in grey; pixels added through dilation are in black.

3.4 Boolean convolution

An arbitrary binary image object (or structuring element) *A* can be represented as:

$$A \leftrightarrow \sum_{k=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} a[j,k] \bullet \delta[m-j, n-k]$$

where "Σ" and "•" are the Boolean operations OR and AND, *a*[*j*, *k*] is a *characteristic function* that takes on the Boolean values "1" and "0" as follows:

$$a[j,k] = \begin{cases} 1 & a \in A \\ 0 & a \notin A \end{cases}$$

and δ[*m*, *n*] is a Boolean version of the Dirac delta function that takes on the Boolean values "1" and "0" as follows:

$$\delta[j,k] = \begin{cases} 1 & j = k = 0 \\ 0 & otherwise \end{cases}$$

Dilation for binary images can therefore be written as:

$$D(A,B) = \sum_{k=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} a[j,k] \bullet b[m-j,n-k] = a \otimes b$$

which, because Boolean ***OR*** and ***AND*** are commutative, can also be written as:

$$D(A,B) = \sum_{k=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} a[m-j,n-k] \bullet b[j,k] = a \otimes b = D(B,A)$$

Using De Morgan's theorem:

$$\overline{(a+b)} = \overline{a} \bullet \overline{b} \quad and \quad \overline{(a \bullet b)} = \overline{a} + \overline{b}$$

on equation, erosion can be written as:

$$E(A,B) = \prod_{k=-\infty}^{+\infty} \prod_{j=-\infty}^{+\infty} (a[m-j,n-k] + \overline{b}[-j,-k])$$

Thus, dilation and erosion on binary images can be viewed as a form of convolution over a Boolean algebra. When convolution is employed, an appropriate choice of the boundary conditions for an image is essential. Dilation and erosion, being a Boolean convolution, are no exception. The two most common choices are that either everything outside the binary image is "0" or everything outside the binary image is "1".

3.5 *Opening and closing*

We can combine dilation and erosion to build two important higher order operations:

- ***Opening:*** $O(A,B) = A \circ B = D(E(A,B),B)$
- ***Closing:*** $C(A,B) = A \bullet B = E(D(A,-B),-B)$

The *opening* and *closing* have the following properties:

- ***Duality:*** $C^C(A,B) = O(A^C,B)$

  $O^C(A,B) = C(A^C,B)$

- ***Translation:*** $O(A+x,B) = O(A,B)+x$

  $C(A+x,B) = C(A,B)+x$

For the opening with structuring element ***B*** and images ***A***, $\boldsymbol{A}_1$, and $\boldsymbol{A}_2$, where $\boldsymbol{A}_1$ is a sub – image of $\boldsymbol{A}_2$ ($\boldsymbol{A}_1 \subseteq \boldsymbol{A}_2$):

- ***Antiextensivity:*** $O(A,B) \subseteq A$
- ***Increasing monotonicity:*** $O(A_1,B) \subseteq O(A_2,B)$
- ***Idempotence:*** $O(O(A,B),B) = O(A,B)$

For the closing with structuring element ***B*** and images ***A***, $\boldsymbol{A}_1$, and $\boldsymbol{A}_2$, where $\boldsymbol{A}_1$ is a sub – image of $\boldsymbol{A}_2$ ($\boldsymbol{A}_1 \subseteq \boldsymbol{A}_2$):

- ***Extensivity***: $A \subseteq C(A,B)$

- ***Increasing monotonicity:*** $C(A_1,B) \subseteq C(A_2,B)$

- ***Idempotence:*** $C(C(A,B),B) = C(A,B)$

The properties given by equations are so important to mathematical morphology that they can be regarded as the reason for defining erosion with ***–B*** instead of ***B*** in equation.

3.6 Hit–And–Miss operations

The *hit – or – miss operator* was first defined by Serra [Serra 1982], but it shall be referred to as the *hit – and – miss* operator and define it as follows. Given an image *A* and two structuring elements ***B***$_1$ and ***B***$_2$, the set definition and Boolean definition are:

- ***Hit–And–Miss:*** $(A,B_1,B_2) = \begin{cases} E(A,B_1) \cap \overline{E^C(A^C,B_2} \\ E(A,B_1) \bullet \overline{E(\bar{A},B_2)} \\ E(A,B_1) - E(\bar{A},B_2) \end{cases}$

where ***B***$_1$ and ***B***$_2$ are bounded, *disjoint structuring elements*. Two sets are disjoint if $B_1 \cap B_2 = \emptyset$, the empty set. In an important sense the hit – and – miss operator is the morphological equivalent of *template matching*, a well known technique for matching patterns based upon cross – correlation, as defined in section 2.7 (i.e. there is a template ***B***$_1$ for the object and a template ***B***$_2$ for the background).

3.7 Summary of the operations

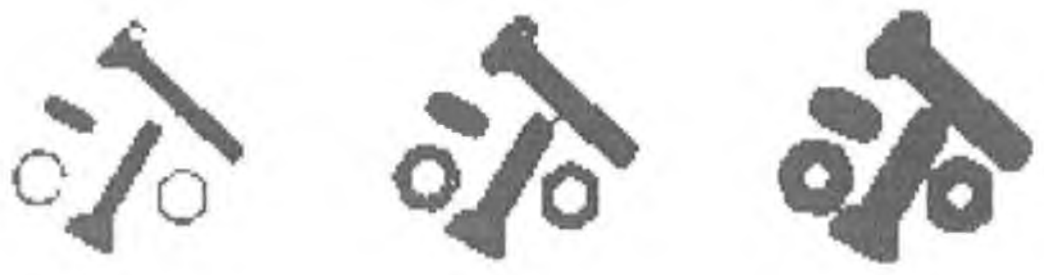
The results of the application of these basic operations on a test image are illustrated in figure 3.7.2. In figure 3.7.1 the various structuring elements used in the processing are defined. The mark "–" indicates an "irrelevant" value. All three structuring elements are symmetric.

$$B = N_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad B_1 = \begin{bmatrix} - & - & - \\ - & 1 & - \\ - & - & - \end{bmatrix} \quad B_2 = \begin{bmatrix} - & 1 & - \\ 1 & - & 1 \\ - & 1 & - \end{bmatrix}$$

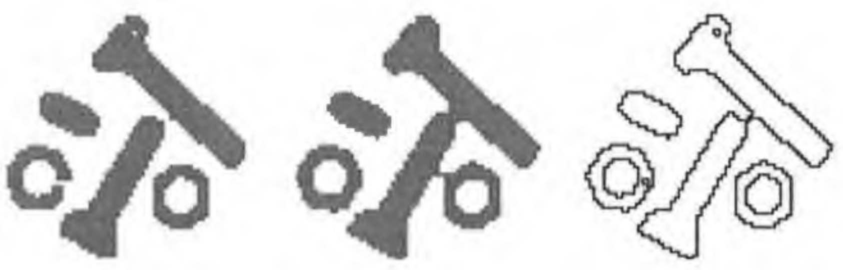
(a) (b) (c)

Figure 3.7.1: Structuring elements ***B***, ***B*₁**, and ***B*₂** that are 3 x 3 and symmetric.

The results of the processing are shown in Figure 3.7.2 where the binary value "1" is shown in black and the value "0" in white.



a) Image A b) Dilation with $\boldsymbol{B_2}$ c) Erosion with $\boldsymbol{B_2}$



d) Opening with $\boldsymbol{B_2}$ e) Closing with $\boldsymbol{B_2}$ f) Hit–and–Miss with $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$

Figure 3.7.2: Examples of various mathematical morphology operations.

The opening operation can separate objects that are connected in a binary image. The closing operation can fill in small holes. Both operations generate a certain amount of smoothing on an object contour given a "smooth" structuring element. The opening smoothes from the inside of the object contour and the closing smoothes from the outside of the object contour. The hit – and – miss example has found the 4 – connected contour pixels. An alternative method to find the contour is simply to use the relation:

- ***4 – connected contour:*** $\partial A = A - E(A, N_4)$

or:

- ***8 – connected contour:*** $\partial A = A - E(A, N_8)$

3.8 Skeleton

Skeletonization is a process for reducing foreground regions in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels. To see how this works, it can be imagined that the foreground regions in an input binary image are made of some uniform slow burning material. Light fires simultaneously at all points along the boundary of this region and watch the fire move into the interior. At points where the fire traveling from two different boundaries meets itself, the fire will extinguish itself and the points at which this happens form the so called skeleton. Under this definition it is clear that thinning produces a sort of skeleton [Davies, 1990].

The informal definition of a skeleton is a line representation of an object that is:

i) One pixel thick;

ii) Through the "middle" of the object;

iii) Preserves the topology of the object.

These are not always realizable; figure 3.8.1 shows why this is the case:

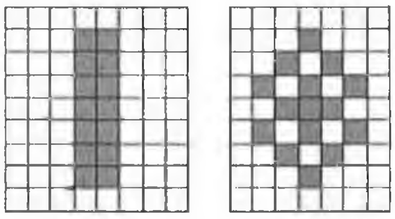

(a) (b)

Figure 3.8.1: Counterexamples to the three requirements.

In the first example, figure 3.8.1 – a, it is not possible to generate a line that is one pixel thick and in the centre of an object while generating a path that reflects the simplicity of the object. In figure 3.8.1 – b it is not possible to remove a pixel from the 8 – connected object and simultaneously preserve the topology – the notion of connectedness of the object. Nevertheless, there are a variety of techniques that attempt to achieve this goal and to produce a *skeleton*. A basic formulation is based on the work of Lantuéjoul [Lantuéjoul, 1977]. The *skeleton subset* $\boldsymbol{S_k(A)}$ is defined as:

- ***Skeleton subsets:*** $$S_k(A) = E(A,kB) - [E(A,kB) \circ B] \quad k = 0,1,...K$$

where K is the largest value of k before the set $\boldsymbol{S_k(A)}$ becomes empty (from equation $E(A,kB) \circ B \subseteq E(A,kB)$). The structuring element $\boldsymbol{B}$ is chosen (in Z^2) to approximate a circular disc, that is, convex, bounded and symmetric. The *skeleton* is then the union of the skeleton subsets:

- ***Skeleton:*** $$S(A) = \bigcup_{k=0}^{K} S_k(A)$$

An elegant side effect of this formulation is that the original object can be reconstructed given knowledge of the skeleton subsets $\boldsymbol{S_k(A)}$, the structuring element $\boldsymbol{B}$, and K:

- ***Reconstruction:*** $A = \bigcup_{k=0}^{K} (S_k(A) \oplus kB)$

This formulation for the skeleton, however, does not preserve the topology, a requirement described in equation.

An alternative point of view is to implement a *thinning*, an erosion that reduces the thickness of an object without permitting it to vanish. A general thinning algorithm is based on the hit – and – miss operation:

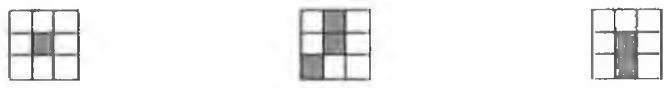
- ***Thinning:*** $Thin(A, B_1, B_2) = A - HitMiss(A, B_1, B_2)$

Depending on the choice of $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$, a large variety of thinning algorithms, and through repeated application skeletonizing algorithms, can be implemented.

A quite practical implementation can be described in another way. If the operation is restricted to a 3 x 3 neighborhood, similar to the structuring element $\boldsymbol{B} = \boldsymbol{N_8}$ (in Figure 3.7.1 – a), then the thinning operation can be viewed as a window that repeatedly scans over the binary image and sets the centre pixel to "0" under certain conditions. The centre pixel is *not* changed to "0" if and only if:

i) An isolated pixel is found (i.e. Figure 3.8.2 – a),
ii) Removing a pixel would change the connectivity (i.e. Figure 3.8.2 – b),
iii) Removing a pixel would shorten a line (i.e. Figure 3.8.2 – c).

As pixels are potentially removed in each of the iterations, the process is called *conditional erosion*. Three test cases of equation are illustrated in Figure 3.8.2. In general all possible rotations and variations have to be checked. As there are only 512 possible combinations for a 3 x 3 window on a binary image, this can be done easily with the use of a lookup table.



a) Isolated pixel b) Connectivity pixel c) End pixel

Figure 3.8.2: Test conditions for conditional erosion of the centre pixel.

If only condition (*i*) is used then each object will be reduced to a single pixel. This is useful if we wish to count the number of objects in an image. If only condition (*ii*) is used then holes in the objects will be found. If conditions (*i + ii*) are used each object will be reduced to either a single pixel if it does not contain a hole or to closed rings if it does contain holes. If conditions (*i + ii + iii*) are used then the "complete skeleton" will be generated as an approximation to equation. Illustrations of these various possibilities are given in Figure 3.10 – c, d.

3.9 Propagation

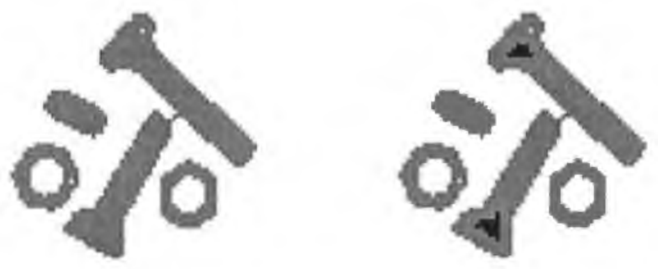
It is convenient to be able to reconstruct an image that has been transformed by several erosions or to fill an object that is defined, for instance, by a boundary. The mechanism for this has several names including *region – filling*, *reconstruction*, and *propagation*. The definition is given by the following algorithm: we start with a *seed image* $S^{(0)}$, a *mask image* **A**, and a structuring element **B**. Then the dilations of **S** are used with structuring element **B** and masked by **A** in an iterative procedure as follows:

- ***Iteration k:*** $S^{(k)} = \left[S^{(k-1)} \oplus B\right] \cap A \quad \textit{until} \quad S^{(k)} = S^{(k-1)}$

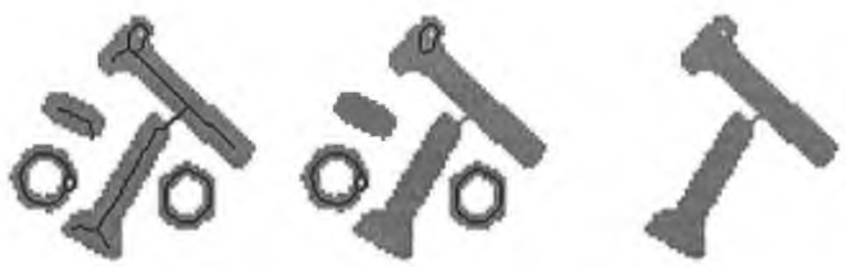
With each iteration the seed image grows (through dilation) but *within* the set (object) defined by **A**; ***S*** *propagates* to fill **A**. The most common choices for **B** are N_4 or N_8. Several remarks are central to the use of propagation. First, in a straightforward implementation, as suggested by equation the computational costs are extremely high. Each iteration requires $O(N^2)$ operations for an N x N image and with the required number of iterations this can lead to a complexity of $O(N^3)$. Fortunately, a recursive implementation of the algorithm exists in which one or two passes through the image are usually sufficient, meaning a complexity of $O(N^2)$. Second, although not much attention has been paid to the issue of object – background connectivity until now (figure 3.2.2), it is essential that the connectivity implied by ***B*** be matched to the connectivity associated with the boundary definition of ***A***. Finally, as mentioned earlier, it is important to make the correct choice ("0" or "1") for the boundary condition of the image [Serra, 1988]. The choice depends upon the application.

3.10 Summary of skeleton and propagation

The application of these two operations on a test image is illustrated in figure 3.10. In figure 3.10 – c,d, the skeleton operation is shown with the end pixel condition and without the end pixel condition. The propagation operation is illustrated in figure 3.10 – e. The original image (shown in light grey), was eroded by $E(\boldsymbol{A},6N_8)$ to produce the *seed* image (shown in black). The original was then used as the *mask* image to produce the final result. The border value in both images was "0".



a) Original picture b) Skeleton with isolated pixels



c) Skeleton with end pixels d) Skeleton without end pixels e) Propagation with $\mathbf{N_8}$

Figure 3.10: Examples of skeleton and propagation.

3.11 Morphological filters

3.11.1 Grey–value morphological processing

The techniques of morphological filtering can be extended to grey–level images. In order to simplify matters, the presentation will be restricted to structuring elements, ***B***, that comprise a finite number of pixels and are convex and bounded. The structuring element has grey values associated with every coordinate position as does the image ***A***.

Grey–level dilation, D_G, is given by:

- ***Grey–level Dilation:*** $D_G(A,B) = \max\limits_{[j,k]\in B}\{a[m-j,n-k]+b[j,k]\}$

For a given output coordinate [*m*, *n*], the structuring element is summed with a shifted version of the image and the maximum encountered over all shifts within the *j x k domain* of ***B*** is used as the result. Should the shifting require values of the image ***A*** that are outside the *M* x *N* domain of ***A***, then a decision must be made as to which model for image extension.

Grey–level erosion, E_G, is given by:

- ***Grey–level Erosion:*** $E_G(A,B) = \min\limits_{[j,k]\in B}\{a[m+j,n+k]-b[j,k]\}$

The duality between *grey–level erosion* and *grey–level dilation* (the grey–level counterpart of equation) is somewhat more complex than in the binary case:

- ***Duality:*** $E_G(A,B) = -D_G(-\tilde{A},B)$

Where "$-\tilde{A}$" means that $a[j,k] \to -a[-j,-k]$.

The definitions of higher order operations such as *grey–level opening* and *grey–level closing* are:

- ***Opening:*** $O_G(A,B) = D_G(E_G(A,B),B)$

- ***Closing:*** $C_G(A,B) = -O_G(-A,-B)$

The important properties that were discussed earlier such as idempotence, translation invariance, increasing in *A* (section 3.5), are also applicable to grey level morphological processing. In many situations the apparent complexity of grey level morphological processing is significantly reduced through the use of symmetric structuring elements where $b[j, k] = b[-j,-k]$. The most common of these is based on the use of B = *constant* = *0*. For this case, using again the domain $[j,k] \subset B$, the definitions above reduce to:

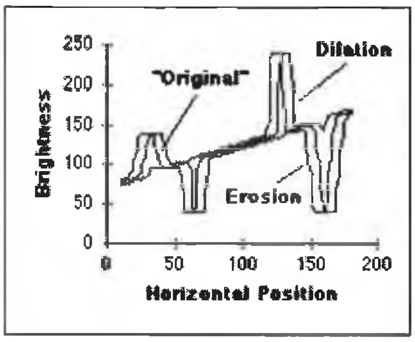
- ***Dilation:*** $D_G(A,B) = \max_{[j,k]\in B}\{a[m-j,n-k] = \max_B(A)\}$

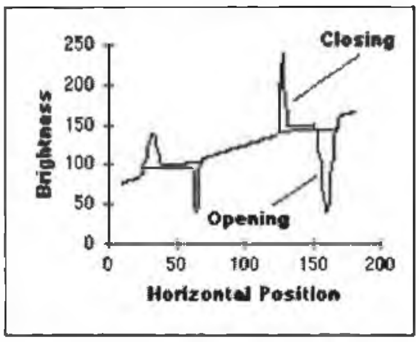
- ***Erosion:*** $E_G(A,B) = \min_{[j,k]\in B}\{a[m-j,n-k] = \min_B(A)\}$

- ***Opening:*** $O_G(A,B) = \max_B(\min_B(A))$

- ***Closing:*** $C_G(A,B) = \min_B(\max_B(A))$

The conclusion is that the maximum filter and the minimum filter are grey level dilation and grey level erosion for the specific structuring element given by the shape of the filter window with the grey value "0" inside the window. Examples of these operations on a simple one dimensional signal are shown in Figure 3.11.1:





a) Effect of 15 x 1 dilation and erosion b) Effect of 15 x 1 opening and closing

Figure 3.11.1: Morphological filtering of grey level images.

For a rectangular window, *j x k*, the two–dimensional maximum or minimum filter is separable into two, one–dimensional windows. More, a one dimensional maximum or minimum filter can be written in incremental form. This means that grey–level dilations and erosions have a computational complexity per pixel that is O(constant), that is, independent of *j* and *k*.

The operations defined above can be used to produce morphological algorithms for smoothing, gradient determination and a version of the Laplacian. All are constructed from the primitives for grey level dilation and grey level erosion and in all cases the maximum and minimum filters are taken over the domain $[j,k] \in B$.

3.11.2 Morphological smoothing

This algorithm is based on the observation that a grey level opening smoothes a grey value image from above the brightness surface given by the function *a*[*m*, *n*] and the grey level closing smoothes from below. A structuring element ***B*** based on equations above has been used:

- ***Morphological smoothing:***

$$MS(A,B) = C_G(O_G(A,B),B) = \min(\max(\max(\min(A))))$$

3.11.3 Morphological gradient

For linear filters the gradient filter yields a vector representation with a magnitude and direction. The version presented here generates a morphological estimate of the *gradient magnitude*:

- ***Gradient Magnitude:***

$$Gradient(A,B) = \frac{1}{2}\left(D_G(A,B) - E_G(A,B)\right) = \frac{1}{2}\left(\max(A) - \min(B)\right)$$

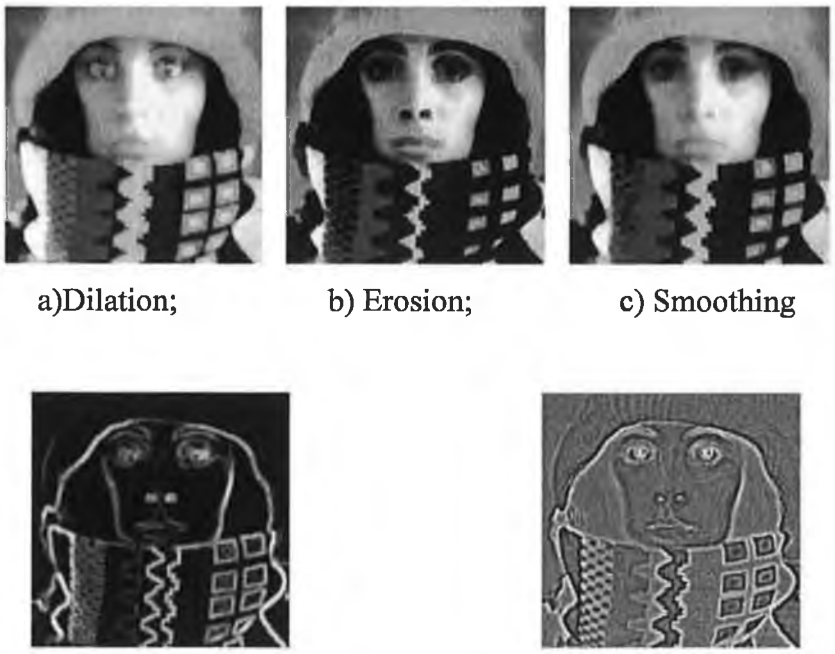
3.11.4 Morphological Laplacian

The morphologically–based Laplacian filter is defined by:

$$\begin{aligned} Laplacian(A,B) &= \frac{1}{2}\left(\left(D_G(A,B) - A\right) - \left(A - E_G(A,B)\right)\right) \\ &= \frac{1}{2}(D_G(A,B) + E_G(A,B) - 2A) \\ &= \frac{1}{2}(\max(A) + \min(A) - 2A) \end{aligned}$$

3.11.5 Summary of morphological filters

The effect of these filters is illustrated in figure 2.11.5. All images were processed with a *3 x 3* structuring element as described in equations through. Figure 2.11.5e was contrast stretched for display purposes.



a)Dilation; b) Erosion; c) Smoothing

d) Gradient; e) Laplacian

Figure 2.11.5: Examples of grey–level morphological filters.

3.12 Sample application

This section will present a small application that implements most morphological operations described in this chapter, but also some of the filters presented in previous chapter. The object of the exercise is to detect some electronic components on a circuit board. The example was built using the Image Processing Toolbox included in Matlab. The steps are presented below:

- ***Step 1*** – Read image.

  A sample image (figure 3.12.1) containing electronic components is read from disk:

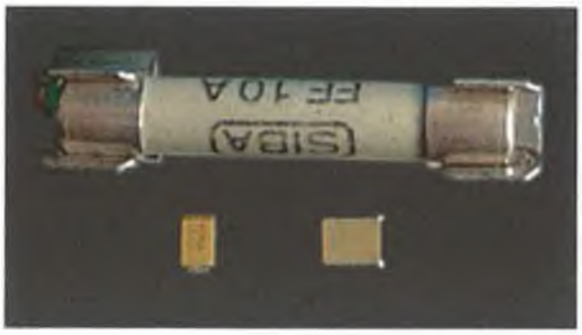


Figure 3.12.1: Sample image

- ***Step 2*** – The image is converted to grey scale and histogram equalized (as illustrated in section 2.3.2), so that it covers the entire dynamic range (*[0, 1]*).

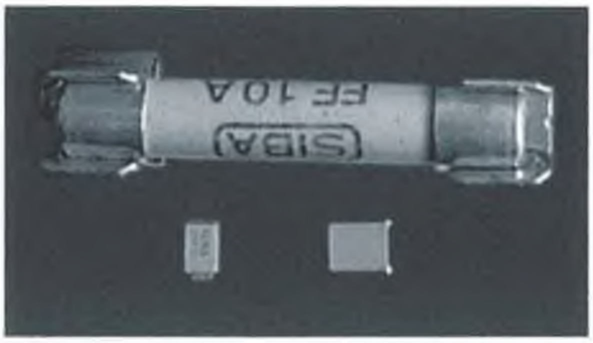


Figure 3.12.2: Grey scale, histogram equalized image

- ***Step 3*** – The objects to be segmented (detected) differ greatly in contrast from the background image. Changes in contrast can be detected by operators that calculate the gradient of an image. One way to calculate the gradient of an image is the Sobel operator (see section 2.6.1), which creates a binary mask using a user – specified threshold value. Figure 3.12.3 illustrates the binary gradient mask.

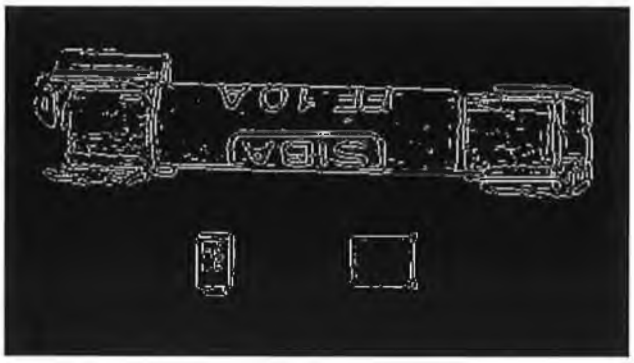


Figure 3.12.3: Binary gradient mask (edge detection)

- ***Step 4*** – The binary gradient mask above shows lines of high contrast in the image. As it can be noticed, these lines do not quite delineate the outline of the objects of interest. Compared to the original image, gaps can be seen in the lines surrounding the objects in the gradient mask. These linear gaps will disappear if the Sobel image is dilated (see section 3.3) using linear structuring elements. The binary gradient mask is dilated using a vertical structuring element (*i*) followed by a horizontal structuring element (*ii*). A similar example was presented in section 3.7.

  i) Vertical structuring element:

$$V = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

  ii) Horizontal structuring element:

$$H = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Next figure shows the effect of the dilation using the structuring elements above:

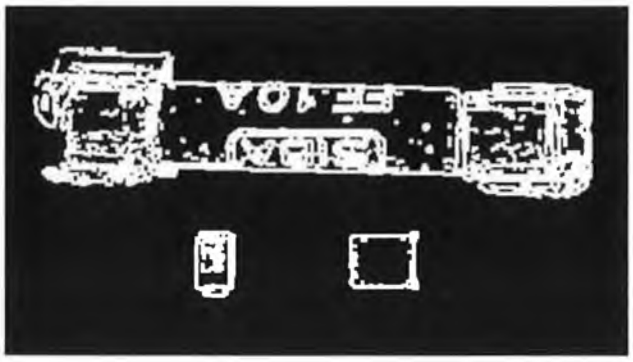


Figure 3.12.4: Dilated image

- ***Step 5*** – The outline of the components can now be seen quite clearly, but there are still holes in the interior. The gaps may be filled using a closing filter, as described in section 3.5. For someone who whishes to automate this process the closing may be applied several times until all the gaps are filled. Once all the gaps are filled, the filter has no effect. The result of the closing filter is presented in figure 3.12.5.

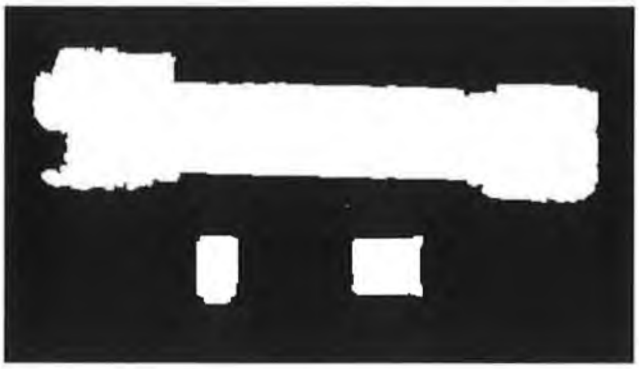


Figure 3.12.5: Closing filter

- ***Step 6*** – In order to make the segmented objects look natural, the objects are smoothed by eroding (see section 3.3) the image twice with a *diamond* structuring element. The diamond structuring element has the following form:

$$D = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and the result of the filter on the image is presented in figure below:

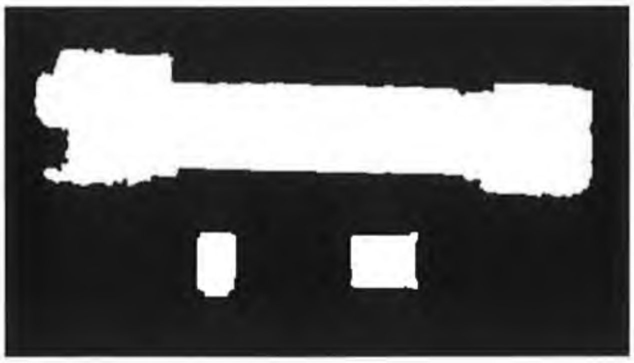


Figure 3.12.6: Eroded image

- ***Step** 7* – An alternate method for displaying the segmented objects would be to place an outline around the segmented components. The outline can be created by the hit – and – miss operator as described in section 3.6. The structuring elements have the following form:

$$S_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad S_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The effect of the filter is illustrated below:

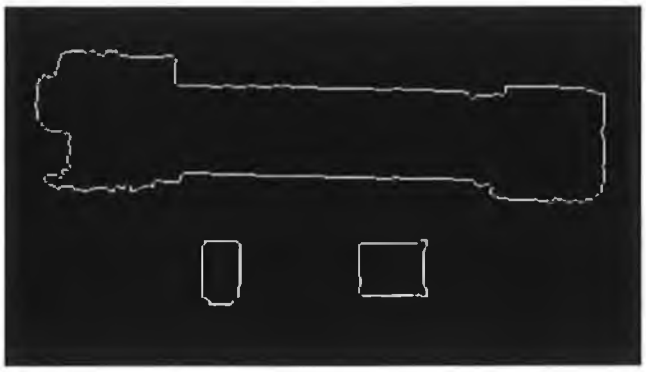


Figure 3.12.7: Hit – and – miss operator for contour detection

- ***Step 8*** – Finally, the result of the segmentation is overlaid on the original picture for visualization purposes. The overlaying is done by using a simple binary addition (***OR***), as defined in section 2.4.1.

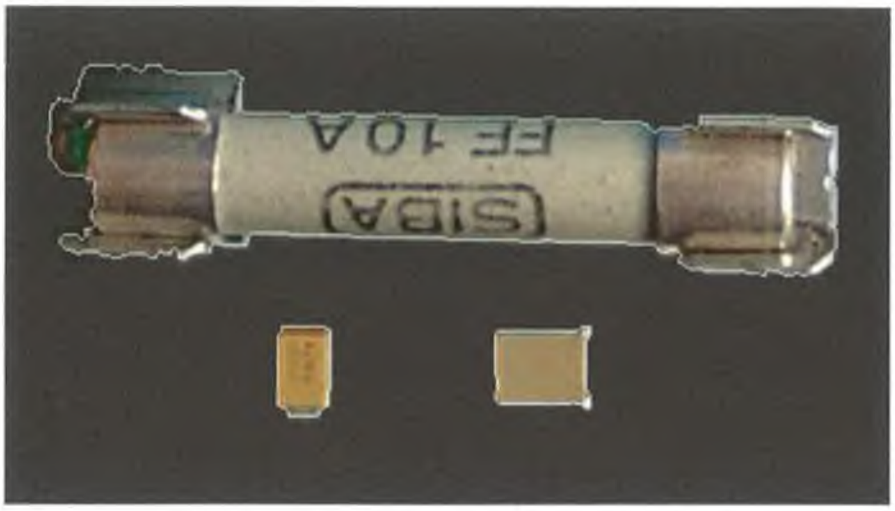


Figure 3.12.8: Final result

3.13 Conclusions

Mathematical morphology has been successfully applied to many problems of image processing (i.e. segmentation, thinning or object recognition). Most of the operations and filters defined in morphological operations also introduce a crisp element, the structuring element or mask. This depends on the nature of the image function used and the mask could be based on human intentions. Mathematical morphology treats with structure describing image operations. Most of the operations of mathematical morphology are based on local operators (i.e. filters) which separate specific image regions due to their structural diversity. For many applications the use of a morphological operation results in the enhancement, filtering, extraction or deletion of an image constituent. In general, mathematical morphology never marks or classifies an image constituent because it is based on local operators. It gives rather a base for a following final image evaluation. The major part of morphological operations can be defined as a combination of two basic operations, dilation and erosion, and set operations like difference, sum, maximum or minimum of two images.

Mathematical morphology usually applies to binary images (black and white) but it can be easily extended to grey–scale (intensity) images (256 shades of grey). The color images need to be brought to either intensity or binary form in order to be processed.

Due to its relative simplicity and fast implementation, most of the actual image processing, or image recognition software are using the mathematical morphology as a primary processing step. Again, like the filters described in the previous chapter, it is hard to think of a system that doesn't use the advantages offered by the mathematical morphology.

4 Image Acquisition Systems

4.1 Introduction

The elaboration and correct utilization of the algorithms used in image processing involve the understanding of the process of forming and digitizing images. It is also very important to understand some fundamental aspects regarding the human visual perception. The image is the result of the interaction between the matter and a certain form of radiation. The understanding of this process is useful in the industrial field and scientific research. The diversity of the applications excludes a complete coverage of this subject with deep roots in physics. Most of the pictures acquired for processing contain opaque objects. The image of such an object is the result of the reflections of the light radiation emitted by one ore more sources. Part of the light field is absorbed by the object while the rest is reflected. Depending on the way the light is reflected there are two types of surfaces: diffuse surfaces and mirrors. An ideal diffuse surface will reflect the light uniformly in all directions. In contrast, the mirrors reflect the light in one direction raising big problems for processing. As a general rule, in nature the surfaces are glossy, a mixture of both types. The spots of light that may appear on a surface have to be avoided as much as possible using proper selected illumination techniques.

This chapter will describe how the images are formed, how different illumination techniques are able to substantially reduce the processing tasks and it will also bring some examples of different software systems on the market.

4.2 Image formation in the eye

The view gives us, the humans, most of the information we are capable to percept from the surrounding environment (the other sensors, i.e. smell, tactile or hear are actually less than 10% of the total information). Image processing has as general objective to facilitate the interpretation of information contained in images. In most cases, image processing includes *artificial view* systems capable to recognize shapes and to offer a general interpretation of the pictures caught by the image sensors (cameras) [Gui, 1999].

Biological vision is the process of using light reflected from the surrounding world as a way of modifying behavior. Generally, with humans, we say that the surrounding environment is *interpreted* by visual input. This usually implies some form of conscious understanding of the 3D world from the 2D projection that it forms on the retina of the eye. However much of our visual computation is carried out unconsciously and often our interpretations can be fallacious.

This section will briefly overview the human visual system and tries to understand the ways in which this system uses computation as a means of interpreting its input. To do this, it will also be explained how images are formed on cameras, how images are stored in computers, and how computations can be carried out efficiently. Although not strictly correct, this analogy between machine vision and biological vision is currently the best model available. Moreover, the models interact in an ever increasing fashion: we use the human visual system as an existence proof that visual interpretation is even possible in the first place, and its response to optical illusions as a way to guide our development of algorithms that replicate the human system; and we use our understanding of machine vision and our ability to generate ever more complex computer images as a way of modifying, or evolving, our visual system in its efforts to interpret the visual world.

4.2.1 The eye

Any understanding of the function of the human eye serves as an insight into how machine vision might be solved. Indeed, it was some of the early work by Hubel and Wiesel [Hubel, 1962] on the receptive fields in the retina that has led to the fundamental operation of spatial filtering that nowadays dominates so much of early image processing. There are many good references to the function of the eye, although Frisby [Frisby, 1979] gives an excellent overview with a computational approach.

The eye is considered by most neuroscientists as actually part of the brain. It consists of a small spherical globe of about two centimeters in diameter, which is free to rotate under the control of six extrinsic muscles. Light enters the eye through the transparent *cornea*, passes through the *aqueous humor*, the *lens*, and the *vitreous humor*, where it finally forms an image on the *retina*.

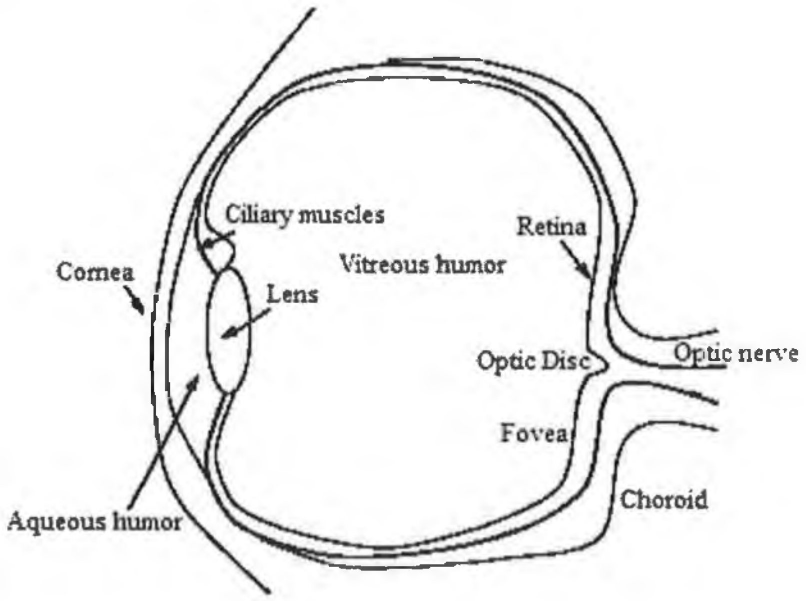


Figure 4.2.1.1: Sketch of a cross–section of the eye

It is the muscular adjustment of the lens, known as *accommodation,* which focuses the image directly on the retina. If this adjustment is not correctly accomplished, the viewer suffers from either nearsightedness or farsightedness. Both conditions are easily corrected with optical lenses.

The retina itself is a complex tiling of photoreceptors. These photoreceptors are known as *rods* and *cones*. When these photoreceptors are stimulated by light, they produce electrical signals that are transmitted to the brain via the *optic nerve*. The location of the optic nerve on the retina obviously prohibits the existence of photoreceptors at this point. This point is known as the *blind spot* and any light that falls upon it is not perceived by the viewer. Most people are unaware of their blind spot, although it is easy to demonstrate that it exists. And its existence has been known about for many years as it is reputed that the executioners in France during the revolution used to place their victim so that his or her head fell onto the blind spot, thus eliciting a pre–guillotine perception of the poor character without a head [Frisby, 1979].

The rods and cones do not have a continuous physical link to the optic nerve fibers. Rather, they communicate through three distinct layers of cells, via junctions known as *synapses*. These layers of cells connect the rods and cones to the *ganglion cells*, which respond to the photostimulus according to a certain *receptive field*. We can see from Figure 4.2.1.2 that the rods and cones are at the *back* of the retina. Thus the light passes through the various cell layers to these receptive fields, and is then transmitted via various synaptic junctions back towards the optic nerve fiber [Marr, 1982].

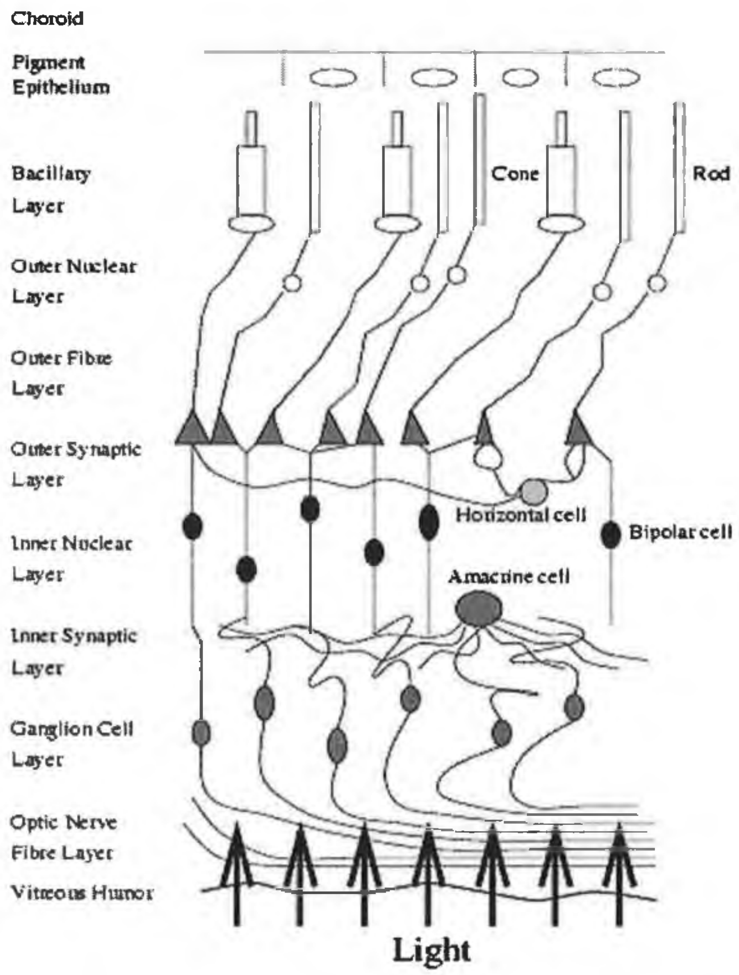


Figure 4.2.1.2: Schematic of a cross section of the retina

The human eye is a remarkable organ, whose sensitivity and performance characteristics approach the absolute limits set by quantum physics. The eye is able to detect as little as a single photon as input, and is capable of adjusting to ranges in light that span many orders of magnitude. No camera has been built that even partially matches this performance.

Very little is known about what happens to the optic signal once it begins its voyage down the optic nerve (Figure 4.2.1.3). The optic nerve has inputs arriving from both the left and right sides of both eyes, and these inputs split and merge at the *optic chiasma*. Moreover, what is seen by one eye is slightly different from what is seen by the other, and this difference is used to deduce *depth* in stereo vision. From the optic chiasma, the nerve fibers proceed in two groups to the *striate cortex*, the seat of visual processing in the brain. A large proportion of the striate cortex is devoted to processing information from the *fovea*.

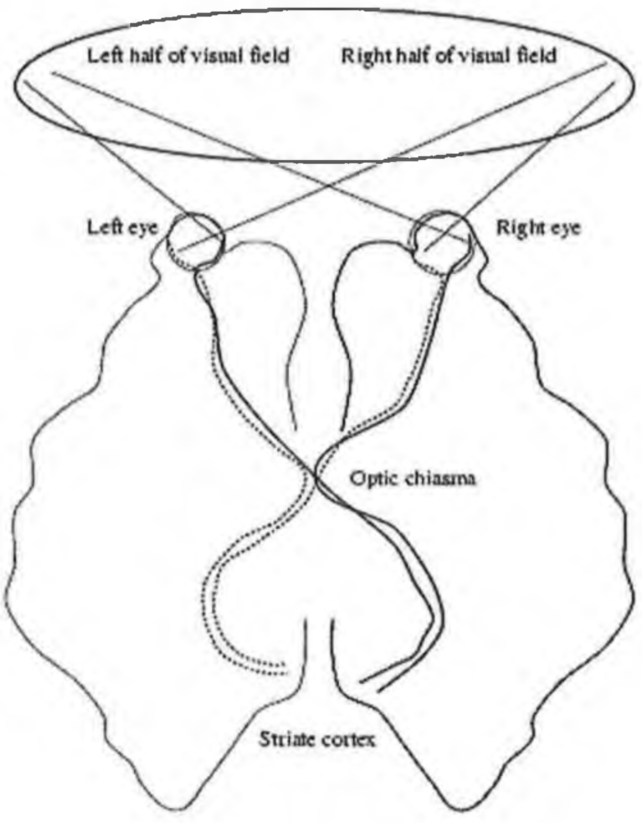


Figure 4.2.1.3: Overview of the visual pathways from eyes to striate cortex

4.3 Camera models

To understand how vision might be modeled computationally and replicated on a computer, first it has to be explained the image acquisition process. The role of the camera in machine vision is similar to that of the eye in biological systems [Horn, 1986].

4.3.1 Pinhole camera model

The pinhole camera is the simplest, and the ideal, model of camera function. It has an infinitesimally small hole through which light enters before forming an inverted image on the camera surface facing the hole. To simplify things, usually it is modeled a pinhole camera by placing the image plane between the focal point of the camera and the object, so that the image is not inverted. This mapping of three dimensions onto two, is called a perspective projection (Figure 3.2.1), and perspective geometry is fundamental to any understanding of image analysis [Serra, 1982].

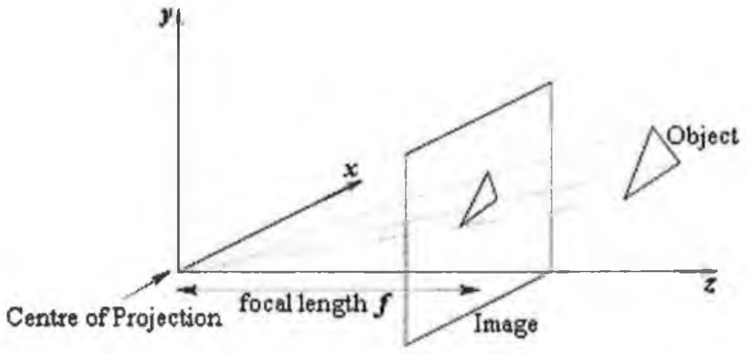


Figure 4.3.1: Perspective projection in the pinhole camera model

4.3.2 Perspective geometry

Euclidean geometry is a special case of perspective geometry, and the use of perspective geometry in computer vision makes for a simpler and more elegant expression of the computational processes that render vision possible. A good overview of the geometric viewpoint in computer vision is given by Faugeras [Faugeras, 1993].

A perspective projection is the projection of a three–dimensional object onto a two–dimensional surface by straight lines that pass through a single point. Simple geometry shows that if we denote the distance of the image plane to the centre of projection by f, then the image coordinates (u, v) are related to the object coordinates (x, y, z) by:

$$u = \left(\frac{f}{z}\right)x \quad and \quad v = \left(\frac{f}{z}\right)y$$

These equations are non–linear. They can be made linear by introducing homogeneous transformations, which is effectively just a matter of placing the Euclidean geometry into the perspective framework. The linear version is simply:

$$\begin{bmatrix} U \\ V \\ S \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Here u = U/S and v = V/S, when $S \neq 0$.

Thus, in the general projective representation, each point in the n–dimensional projective space is represented by an n + 1 vector (Sx_1, ..., Sx_n, S), where $S \neq 0$.

4.3.3 Simple lens model

In reality, one must use lenses to focus an image onto the camera's focal plane. The limitation with lenses is that they can only bring into focus those objects that lie on one particular plane that is parallel to the image plane. Assuming the lens is relatively thin and that its optical axis is perpendicular to the image plane, it operates according to the following lens law:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f}$$

where ***u*** is the distance of an object point from the plane of the lens, ***v*** is the distance of the focused image from this plane, and ***f*** is the focal length of the lens (Figure 4.3.3).

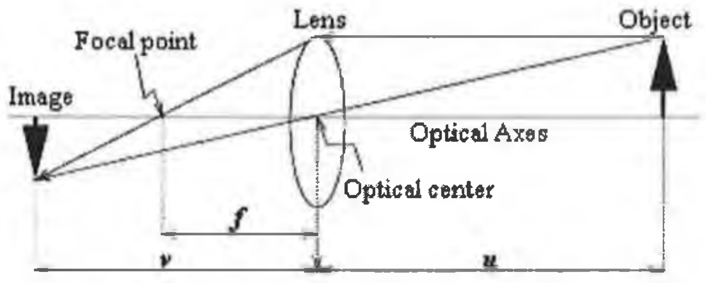


Figure 4.3.3: Simple lens model

4.3.4 Camera calibration

To deduce three–dimensional geometric information from an image, one must determine the parameters that relate the position of a point in a scene to its position in the image. This is known as *camera calibration.* Currently this is a cumbersome process of estimating the *intrinsic* and *extrinsic* parameters of a camera. There are four intrinsic camera parameters: two are for the position of the origin of the image coordinate frame, and two are for the scale factors of the axes of this frame. There are six extrinsic camera parameters: three are for the position of the centre of projection, and three are for the orientation of the image plane coordinate frame. However, recent advances in computer vision indicate that we might be able to eliminate this process altogether [Castleman, 1996].

4.4 A model of visual computation

Because very little is known about how the brain processes visual information, we can instead look at the computer–vision paradigm and highlight methods of computation that are biologically inspired. Much that is done in computer vision has an engineering approach – that is, find a solution, regardless of how it relates to any biological system. However, there are some techniques in computer vision that are inspired by what is known about how biological systems function. Most industrial applications of computer vision still attempt to extract as much information as possible from an impoverished binary image, and indeed much can be done with this data. In the computer–vision paradigm (Figure 4.4), an image is first acquired and then some form of data–reduction is performed to help in the subsequent analyses. This in itself is biologically inspired: the retina has something like 120 million rods and 5 million cones, but the total number of nerve fibers leaving the eye is only about 1 million. Some form of compression must be applied very early in visual processing; the compressed format must be high in information, and low in redundancy, whereas we know that raw image data is highly redundant [Russ 1995].

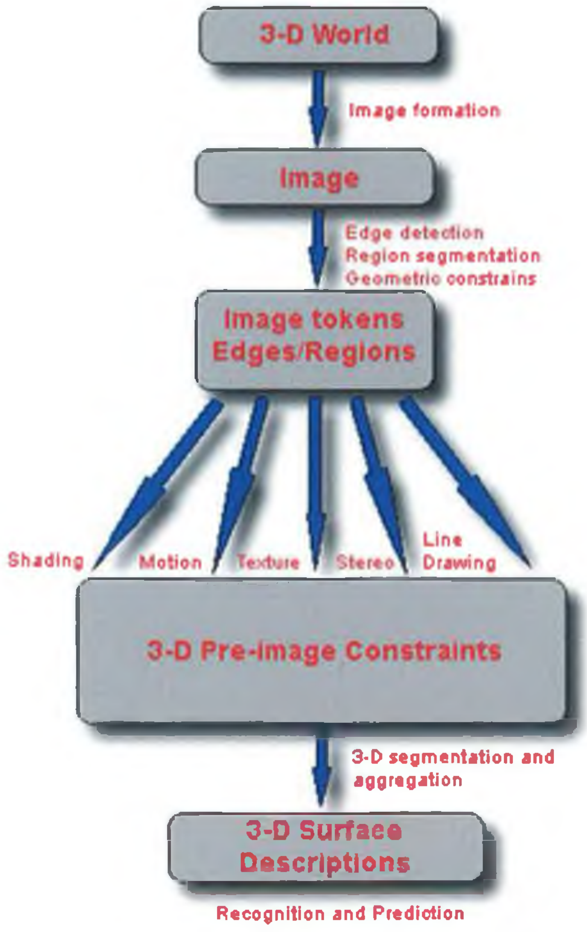


Figure 4.4: Model for computer vision [Russ 1995]

Most researchers agree that the early stages in vision perform some sort of *edge detection*. Following the data compression stages, image interpretation needs to be undertaken. How do we infer information about the 3D world from mere 2D projections? Mathematicians would say that the problem is ill–posed, in the sense that there are an infinite number of solutions or interpretations, but the physical constraints imposed by the real world limit these interpretations substantially. Thus, after the compression stage comes a series of modular, but perhaps interlinked, stages of what is known as *Shape–from–X*. Examples include shape from stereo, shape from motion, shape from texture, or shape from shading. In all these cases, additional information renders an ill–posed problem soluble, and allows us to interpret the 3D geometry of the scene under view. Having obtained the 3D geometry of the scene under view, we then need to interpret it, or match it with our existing knowledge of the world. We understand what it is we see, even when we see that object from a completely novel view. In order to match the geometry of the objects in our scene with that of the objects in our known database, we use *invariants* that are geometrical descriptions of the object that are invariant to the types of transformations that the visual process performs [Russ 1995].

4.5 Base components of an image processing system

An image processing system generally has the following components: A data acquisition module, a processing unit, storage module and display module. In some applications that are less critical regarding the speed of processing (not necessarily easier to solve) the processing system is a personal computer which also has the role of storage and display module.

- ***Data acquisition module***: – has two essential parts: the *image sensor* and the *digitizer* (in case of a digital camera the two components are merged in one system). The image sensor is sensible to light radiation or other form of radiant energy which generates the image field. It transforms the energy field received in electrical signal. The digitizer converts the analogical electrical signal supplied by the image sensor into a numerical form using quantification and sampling techniques. Among the image sensors, the CCD's (Charge – Coupled Devices) that are a part of the modern cameras are the most common choice due to some important advantages:

  - Precise and stable geometry with variations of the temperature;
  - High quality
  - Small size

The CCD cameras are available in a large variety of resolutions starting from a matrix of 300x200 pixels up to more than 4000x4000 pixels. Of course the price grows as well with the resolution. It is worth to mention that the resolution of the camera is not an absolute barrier to obtain accurate measurements. Using ingenious techniques for the measurement of the position or other geometrical parameters, good results can be obtained with a smaller error than the distance between two adjacent cells (pixels) [Nalwa, 1993].

The range of the applications is not constrained by the visual spectrum. The radiation spectrum used for image acquisition starts with gamma radiations domain, continues with X radiation, visible, infrared, microwave, radio wave, and ends up at ultrasound domain. The microwaves and the radio waves are not completely absorbed by the clouds, which make them attractive for environment surveillance systems installed on satellites. Images generated by ultrasounds are obtained by using the echo. At this kind of images the grey color is the distance from the sensor which allows a much direct approximation of the shape of the surfaces than the conventional images.

- ***The processing unit*** – usually the higher the power of the processor the faster the image is processed. It is also important that the algorithms are highly optimized so no processing power is wasted on redundant program loops.
- ***Storage module*** – it is important to mention that the images may consume huge amounts of memory (i.e. a single image with the resolution of 512x512 pixels and a color depth of 8 bits per pixel will occupy 256Kb of memory). The larger the resolution and the color depth the more memory will be necessary for the system.
- ***Display module*** – is the monitor of the computer; displaying high – resolution pictures on a monitor may be a problem. It is important that the video card of the system to have as much memory as possible. Also small monitors will crop the image and the software has to be able to perform zooming and scrolling operations.

Figure 4.5 shows a general diagram for an image processing system:

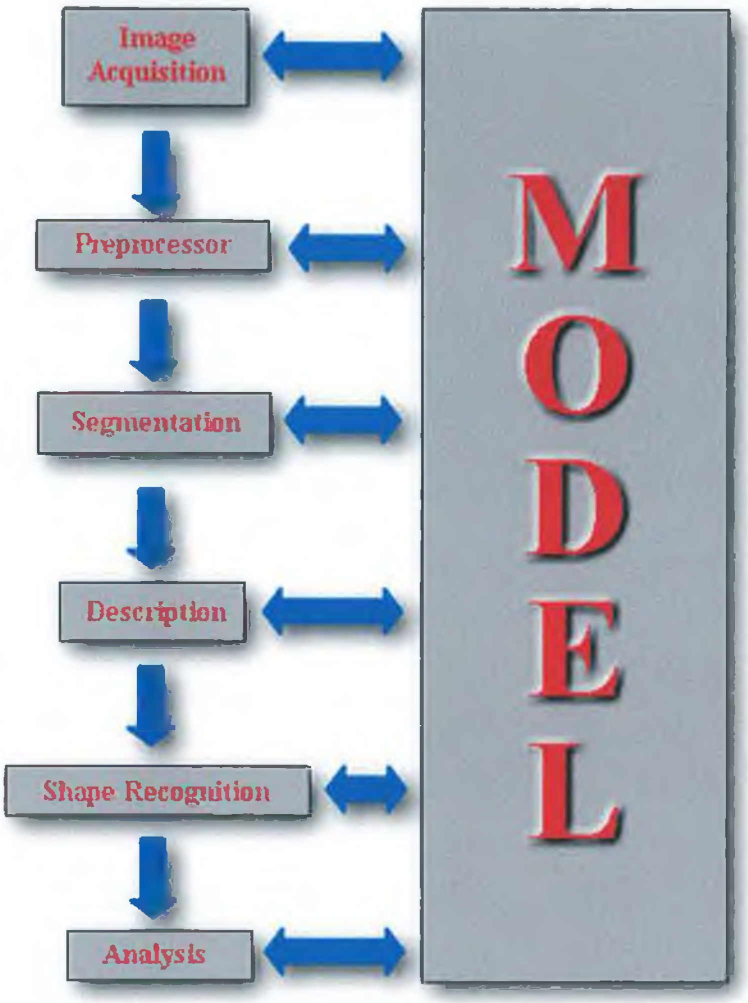


Figure 4.5: General diagram for an image processing system

4.6 Illumination Techniques

This section will describe different illumination techniques for some representative situations. It is recommended exclusive use of artificial light in all cases when the environment conditions allow it because the artificial light is much easier to control than natural light [Batchelor, 1985].

- ***Diffuse illumination*** – is recommended when the object analyzed has regular and smooth surfaces and these surfaces are important for processing or recognition. It is suitable for metallic or glossy surfaces as it eliminates shadows and excessive light reflection.

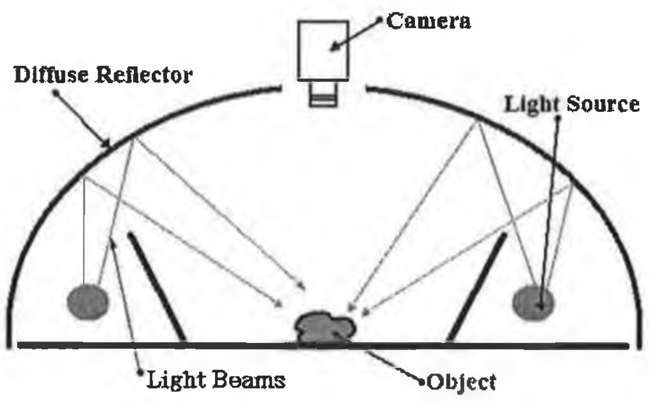


Figure 4.6.1: Diffuse illumination

- ***Background illumination*** – useful for automatic visual inspection, when the contour is important in analysis.

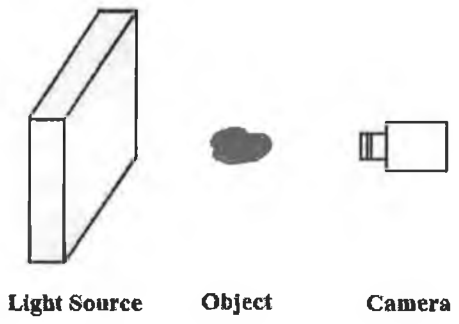


Figure 4.6.2: Background illumination

- ***Side Illumination*** – useful for those applications dedicated to perform automatic quality control of surfaces. The textile, paper and metal processing industry can benefit of this technique. Such technique has been used to detect liquid leaks in a nuclear power plant: when the liquid is present the surface becomes glossy so the reflections are stronger. When they are above a certain threshold value the alarm triggers [Gui, 1999].

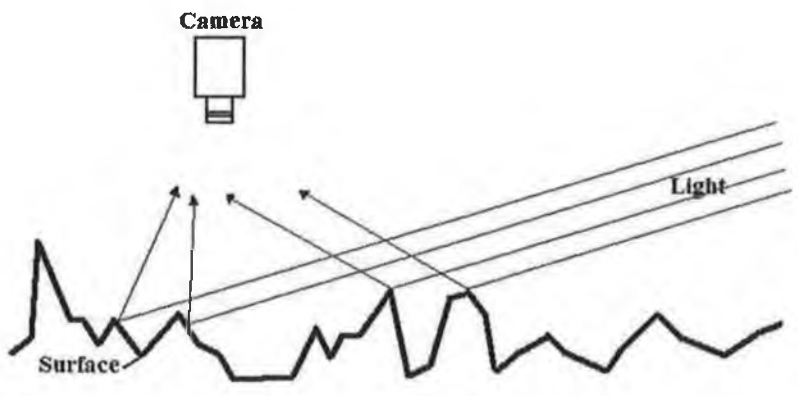


Figure 4.6.3: Side illumination

- ***Structural illumination*** – allows estimating the shape of the surfaces with cameras in visible spectrum as an alternative to the ultrasonic cameras. Depending on the specific of each problem different setups can be made:

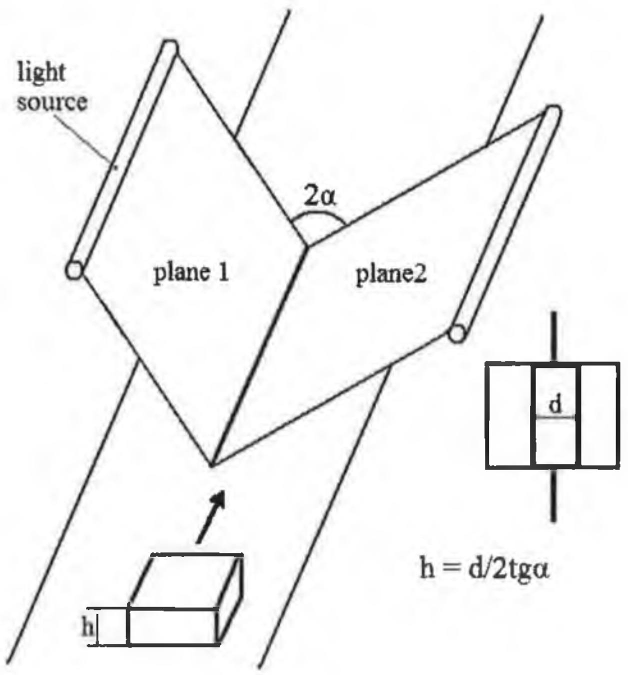


Figure 4.6.4: Structural illumination

The setup can be used to detect the thickness of the object. The two light sources lay symmetrical on each side of the object and generate two light planes that intersect at an angle *2α*. Knowing the angle the thickness *h* can be calculated using the formula: $h = d/2tg\alpha$.

4.7 Image compression

In the modern communication systems, the image compression has a very important role. The impact of the multimedia technologies and their advance in comparison with the traditional systems of communication is based on the capacity to provide a large category of high – fidelity audio – video information integrated with text and numerical data. In this context the images raises many problems for a simple reason: it takes a lot of bytes to be described. A single color image represented on 8 bits per pixel and at a resolution of 512x512 pixels needs 256kb to be stored or transmitted on a communication channel. The situation becomes critical if we are talking about a sequence of images (movie): only one minute of movie, created from pictures having the same parameters as the previous example, running at a frequency of 50Hz takes about 760 Mb of storage or transmission. In this conditions even with the advances of the actual technology a full *c*ompact *d*isk (CD) can hold only a few seconds of movie or a *d*igital *v*ideo *d*isk (DVD) a few minutes, not to mention the case of transmission over a local area network or internet.

The most important achievements in building a standard in image compression are the Joint Picture Expert Group (JPEG) [JPEG, 1990] for still images and Motion Picture Expert Group (MPEG) [MPEG, 1994] for movies. The JPEG algorithm compresses each frame of video data by removing information to which the human eye is not sensitive. MPEG extends this compression technique by transmitting only the part of each video frame image that differs from the one before it. Note that MPEG technology incorporates audio signal too. With JPEG technology, video can be compressed enough to store and manipulate video on a personal computer. With MPEG technology, over 70 minutes of audio and video data can be stored on and played from a 650 Mb compact disc, providing a less expensive distribution medium than videotape. Efficient implementations of image compression are described in Rissanen [Rissanen, 1976] and Pasco [Pasco, 1976]. An excellent tutorial on image compression has been made by Langdon [Langdon, 1984].

4.7.1 Types of compression

No matter how old or what algorithms have been used, the compression techniques can be classified in two categories:

- *Precise* (lossless) techniques (no loss of information)
- *Approximate* techniques (with loss of information)

The *precise* techniques are taking advantage of image redundancy from a statistical point of view. Through the elimination of the redundancy a more compact, perfect reversible image is obtained. As expected the compression factor is smaller than that achieved using *approximate* techniques, but is useful to study it for at least two reasons: 1. there are images that have to be reconstructed perfectly after compression (i.e. biomedical images or the images used in astronomy); 2. the lossless compression is embedded in approximate compression for statistical reasons.

The *approximate* techniques try to find a close representation of the image using as less bytes as possible. Small medium square errors may give a big gain in terms of space. For instance the JPG file format provides a *scale of quality* which goes from excellent to very poor (excellent – good – medium – poor – very poor), depending of the square errors.

4.8 Software solutions

This section will analyze different software packages (dedicated to image processing) existent on the market and the techniques used by the developers.

4.8.1 The Image Analysis and Communications Lab (IACL)

The research is conducted in the Department of Electrical and Computer Engineering of The Johns Hopkins University [http://iacl.ece.jhu.edu/projects]. The areas of concern are Nonlinear Image Processing (Markov Random Fields, Mathematical Morphology), Medical Image Processing, Information theory, and Communications. The targets of the applications are: brain mapping, cardiac motion estimation, automatic target detection, image processing and analysis.

An interesting technique used by IACL is *active contours* or *snakes*. They are computer generated curves that move within images to find object boundaries. They are often used in computer vision and image analysis to detect and locate objects, and to describe their shape. For example, a snake might be used to automatically find a manufactured part on an assembly line; one might be used to find the outline of an organ in a medical image; or one might be used to automatically identify characters on a postal letter. The snakes developed by IACL solve two key problems that have concerned the computer vision community since the introduction of snakes in 1988. These problems are that snakes cannot move toward objects that are too far away and that snakes cannot move into boundary concavities or indentations (such as the top of the character U). Previous efforts to solve these problems have not been completely successful, and have often created new problems along with the proposed solutions.

- ***Method*** – The snake, which has been called the *g*radient *v*ector *f*low (GVF) snake, begins with the calculation of a field of forces, called the GVF forces, over the image domain. The GVF forces are used to drive the snake, modeled as a physical object having a resistance to both stretching and bending, toward the boundaries of the object. The GVF forces are calculated by applying generalized diffusion equations to both components of the gradient of an image edge map. The GVF external forces are what make the snake inherently different from previous snakes. Because the GVF forces are derived from a diffusion operation, they tend to extend very far away from the object. This extends the "capture range" so that snakes can find objects that are quite far away from the snake's initial position. This same diffusion creates forces which can pull active contours into concave regions. A fundamental difference between this formulation and the traditional formulation is that GVF forces are not purely irrotational (curl–free) forces. In fact, they typically comprise both irrotational and solenoidal (divergence–free) fields. Because of this property, they cannot be derived from the energy minimization framework of traditional snakes [Xu, 1997, 1998, 2000].

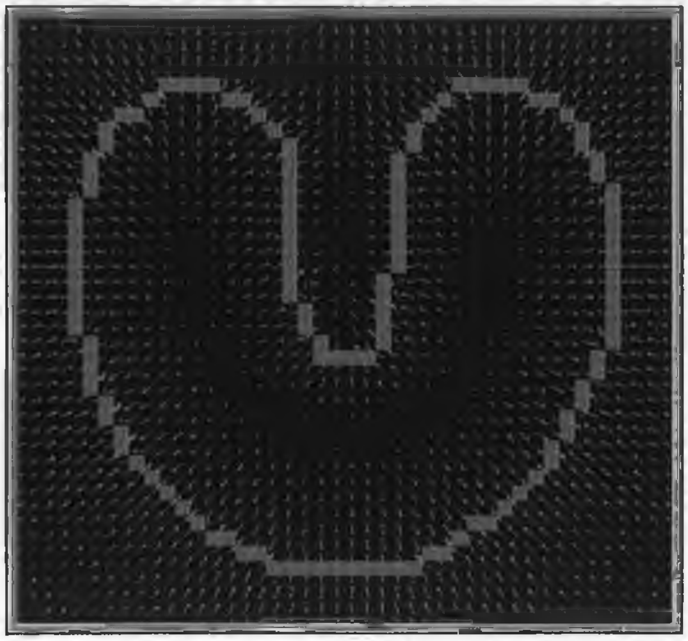
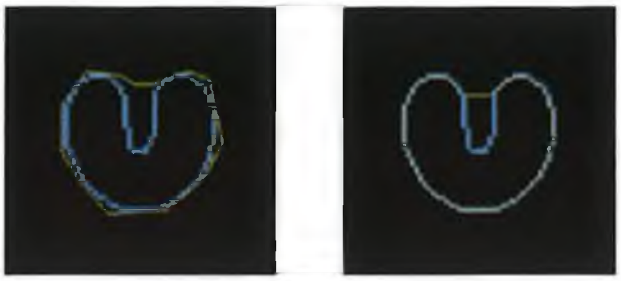


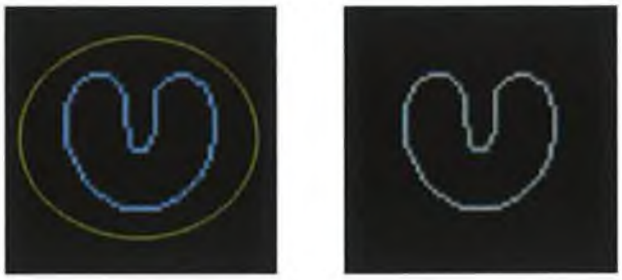
Figure 4.8.1.1: A gradient vector flow (GVF) field for a U–shaped object. These vectors will pull an active contour toward the boundary of the object.

- ***Results*** – The group provides the source code in Matlab, so extensive testing has been possible. The GVF snake has been tested on many types of objects, from simple shapes to magnetic resonance images of the heart and brain. The following examples demonstrate some of the properties of the GVF snake.



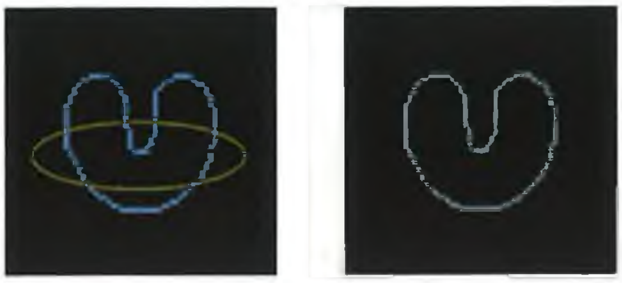
a) Initial curve; b) Final curve

Figure 4.8.1.2: A traditional snake must start close to the boundary and still cannot converge to boundary concavities.



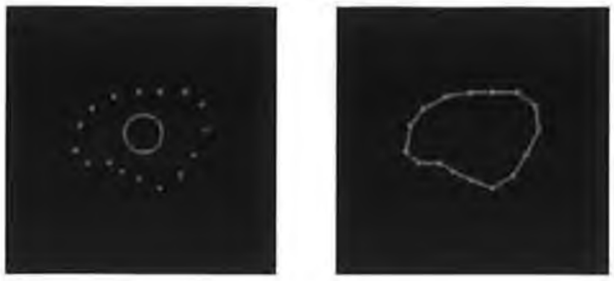
a) initial curve; b) final curve

Figure 4.8.1.3: A GVF snake can start far from the boundary and will converge to boundary concavities.



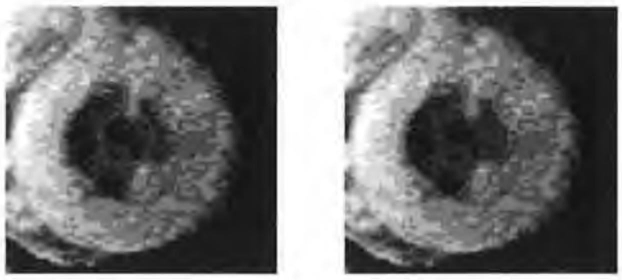
a) initial curve; b) final curve

Figure 4.8.1.4: A GVF snake can even be initialized across the boundaries, a situation that often confounds traditional snakes and balloons.



a) initial curve; b) final curve

Figure 4.8.1.5: GVF snakes converge to subjective contours – contours that are not really there.



a) initial curve; b) final curve

Figure 4.8.1.6: GVF snakes are defined for grayscale images as well as binary images. Here the GVF snake finds the heart wall on a magnetic resonance image.

- ***Disadvantages*** – The snakes work very slowly on large images. The examples above used a 64x64 bpp image and it took about 15 seconds to be processed on a Pentium III CPU at 866MHz system. A 512x512 bpp image it took about 4 minutes on the same system. Noisy images may produce poor results. The algorithm only works on grey–scale images, although color separation may be used for color pictures.

4.8.2 Computational Intelligence Research Laboratory

The Computational Intelligence Research Laboratory (University of Columbia – Missouri) [http://sun16.cecs.missouri.edu/] develops projects in image recognition field; the area of research is addressing the military (automated surveillance) and medical (automated cell count) field.

The military research refers to automatic target recognition. The goal of the project is to detect military vehicles and weapons. Several detectors attempt to locate these targets on approximately 100 different images. The detectors are then fused together, the resulting image is thresholded, and the coordinates of the bounding boxes are sent to a scoring program. The detector bounding boxes are scored against a manually recorded ground truth database, and each detection is grouped into either the "target" or the "non–target" category. The detectors can find most of the actual targets from the range images, but a large number of non–targets like rocks, scrap metal, and even decoys are also included in the detections. This undesirable abundance of false alarms makes it necessary to train a classifier with these objects in order to reduce the number of false alarms. Features are extracted from both classes and used to train a neural network that will decide if a detection is actually a military target or a false alarm.

The final results, in a fully developed and finished product, would then send an "alarm" which would alert personnel of a possible military threat.

The following pictures demonstrate the process of target counting:

Figure 4.8.2.1: Original (input) image

Figure 4.8.2.2: Preprocessed image

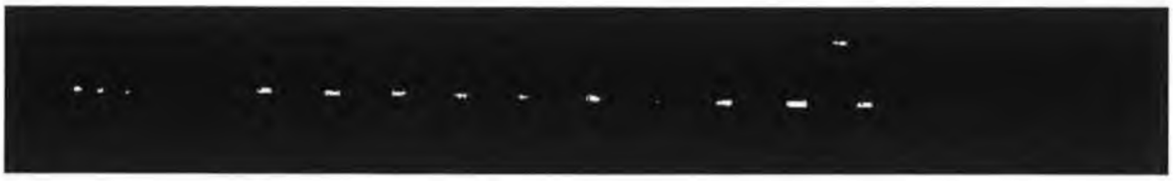


Figure 4.8.2.3: Thresholded image

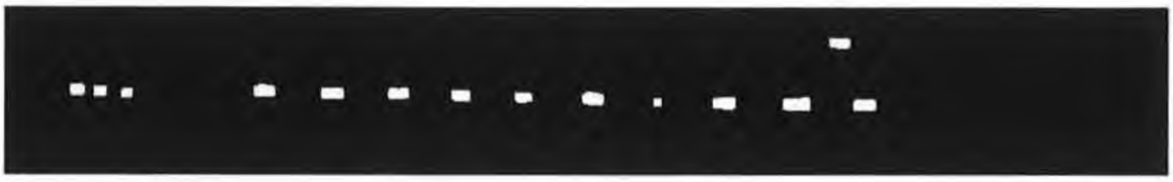


Figure 4.8.2.4: Final, enhanced image (14 targets found)

Another area of research of the Computational Intelligence Research Laboratory is the medical field. The group is currently developing a project on automatic bone marrow cell classification. The idea of the project is that, given the many different classes of white blood cells present in bone marrow smears, a differential count of the various types of cells will give pathologists valuable information regarding the normalcy of the patient, and clues to various cancers. The researchers are trying to automate this activity.
The automatic cell analysis is performed in four successive steps:

- *Find white blood cells in bone marrow smear images;*
- *Split touching white blood cells and segment each cell into cytoplasm and nucleus;*
- *Extract features using the segmentation results;*
- *Classify each white blood cell into various classes.*

The most difficult part of the automatic cell analysis is the segmentation because there is considerable uncertainty in the microscopic images: the maturity classes of the white blood cells actually represent a continuum, cells frequently overlap each other, there are staining and illumination inconsistencies, and there is fairly wide variation of size and shape of nuclear and cytoplasmic regions within given cell classes. This uncertainty makes bone marrow image segmentation a difficult problem. The following section describes the procedure of white blood cell extraction:

Even though high resolution images are needed to segment and classify the white blood cells in bone marrow smears, it is waste of time if high resolution images are scanned only to find a few white blood cells. Hence, a low magnification ratio (200x) have been used to locate white blood cells and continue the segmentation and classification processes at a high magnification ratio (600x).
The following steps are performed in order to extract the white blood cells:

1. Read reduced bone marrow image.

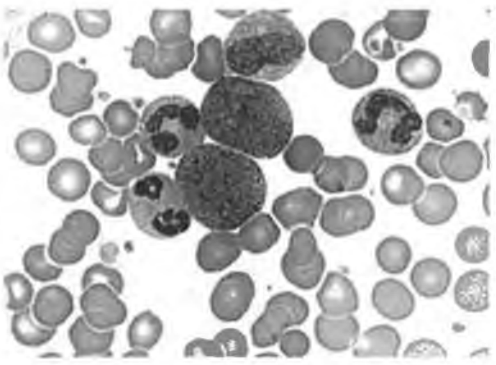


Figure 4.8.2.5: Reduced bone marrow image

2. Compute the histogram of the image.

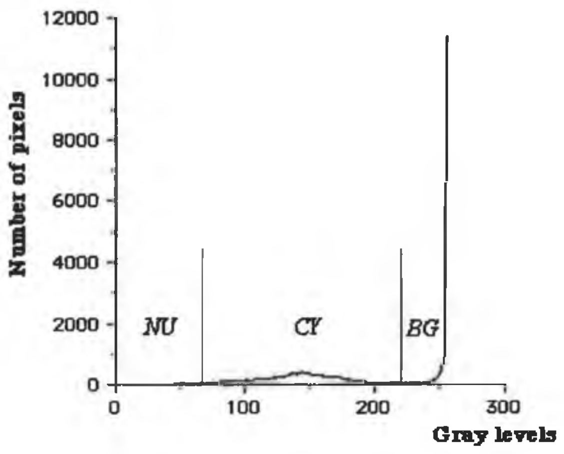


Figure 4.8.2.6: Image Histogram

3. Build a cell mask and nucleus mask by thresholding.

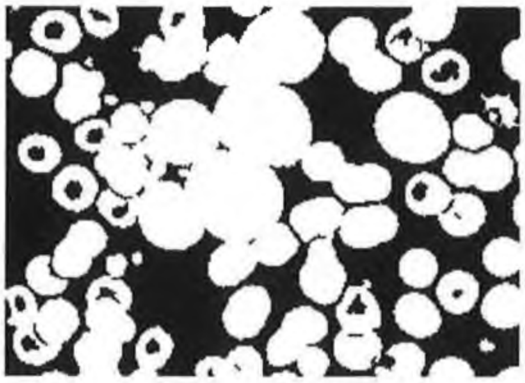


Figure 4.8.2.7: Cell and nucleus mask

4. Make a distance transform image of the cell mask.

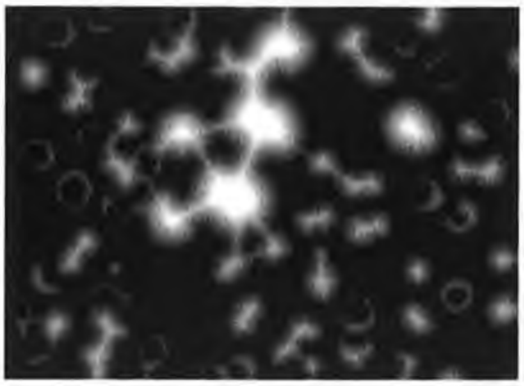


Figure 4.8.2.8: Distance transform

5. Find patches which are big and contains nucleus inside

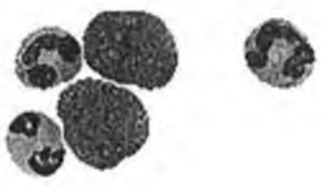


Figure 4.8.2.9: Cells that contain nucleus inside

6. Place an initial snake inside of the selected patches

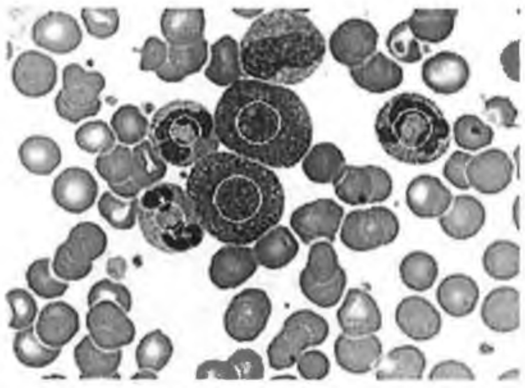


Figure 4.8.2.10: Initial snake

7. Run the snake algorithm to find white blood cell boundaries.

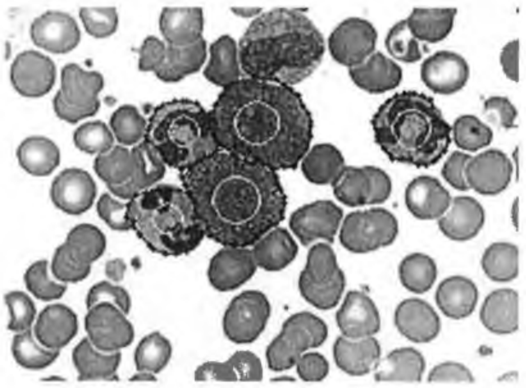


Figure 4.8.2.11: Snake algorithm completed

There are two observations that are to be made regarding these projects:

i) Due to the fact that the projects are still in research phase, there is not much information offered about the exact steps followed, about the algorithms or a test version of the software.

ii) The use of shakes in the second project proves their popularity especially on those image processing applications that target the medical field.

4.9 Conclusions

This chapter has briefly overviewed the human visual system and has attempted to understand the ways in which this system uses computation as a means of interpreting its input. Analogies have been done between human eye system and how images are formed on cameras. Although not strictly correct, this analogy between machine vision and biological vision is currently the best model available for researchers. The model of the human eye, its response to optical illusions, is used as a way to guide our development of algorithms that try to replicate it. We use our understanding of machine vision and our abilities to generate ever more complex computer images as a way of modifying, or evolving, our visual system in its efforts to interpret the visual world. Furthermore, later in the chapter, the attention has been focused on image processing systems and the way its components interact. It has been highlighted the importance of choosing the right camera and illumination system for a certain type of problem.

The chapter was concluded with a study upon the work done by others researchers in image processing field. It has been examined the research conducted by two separate groups (*The Image Analysis and Communications Lab* and *Computational Intelligence Research Laboratory*) that aimed the medical and the military field.

5 Software Module – “Image Acquisition”

5.1 Introduction

The software module was built to meet the requirements of Tyco/Mallinckrodt. The whole package includes not only the software but also the illumination and positioning system and the camera. The aim of this chapter is to describe in detail all these components.

A general overview of the illumination systems was given in the previous chapter, now a 3D model of the system that has been build will be introduced. Also, the previous chapter made an analogy between the human eye and a camera; this chapter gives details about what camera was chosen and why was it chosen. The software package (that analyzes the pictures, performs the filtering, builds the libraries, processes the images and generates log files), uses most of the filters described in chapter two and three. The software was initially built with Microsoft Visual C++ 6.0 and then, in the final phase, it was recompiled with Microsoft Visual C++ .NET in order to take advantage of the new, better optimized, compiler and also the new .NET framework which allows development of applications compatible with the WindowsXp operating system. The image processing algorithms were built using Mathworks Matlab 6.1, compiled and integrated with the software as a Dynamic Link Library.

5.2 Illumination and Positioning System

The acquisition system needed an illumination system able to eliminate shadows and excessive light reflections from metallic surfaces. Constant light on the object was another requirement for the illumination system. An illumination system (similar with the one described in the last chapter, section 4.6) was built in order to meet these requirements.

An overview of the illumination system is presented in figure 5.2.1:

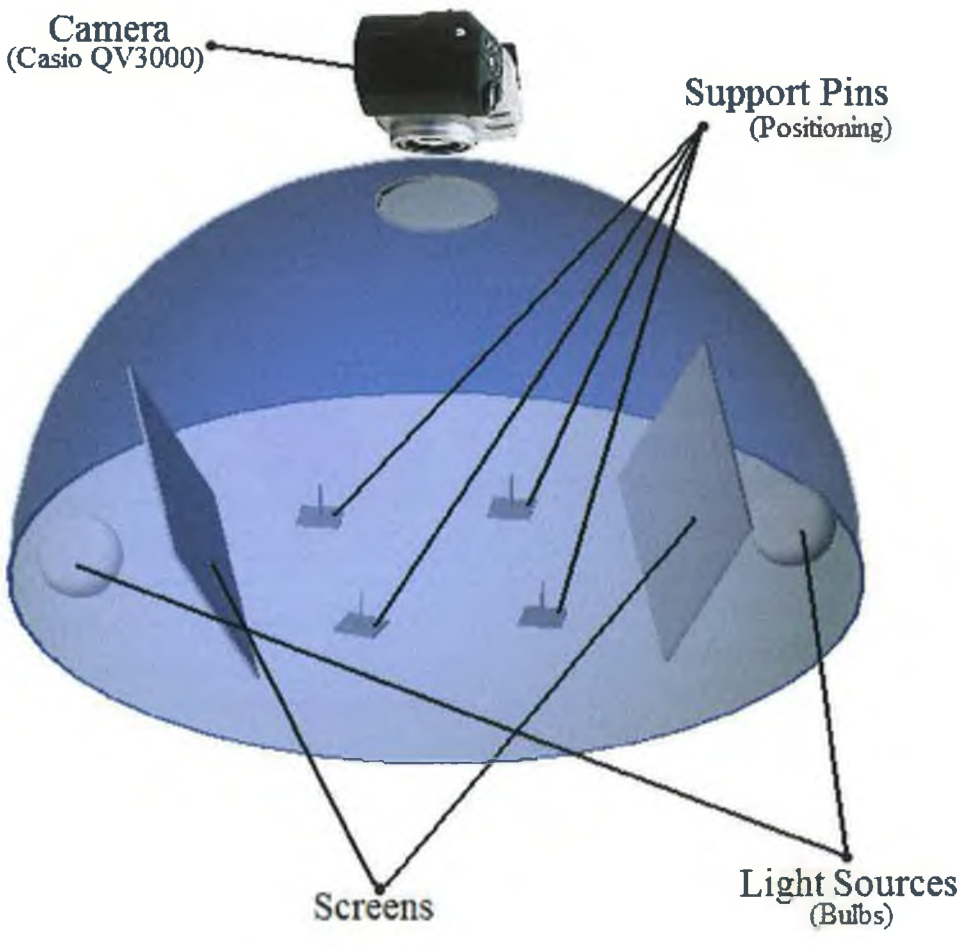


Figure 5.2.1: Illumination system and camera (SolidWorks model)

The light emitted by the light sources is reflected by the cupola and concentrated in center of the imaginary sphere, where the object is. Direct light cannot fall on the object as it is blocked by the two screens; also, it cannot fall straight on the camera objective due to the construction of the cupola (as described in section 4.6.1). The radius of the sphere is *50 cm* and it was chosen to match the minimum focal distance of the camera.

The positioning system consists of four support pins that have the purpose to elevate and keep the PCBs (Printed Circuit Boards) level as shown in figure below.

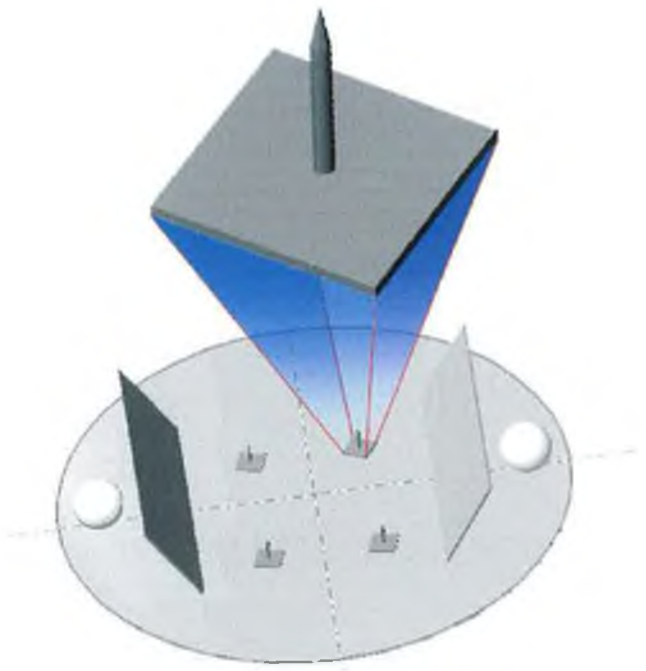


Figure 5.2.2: Support pin (*SolidWorks* model)

The PCBs have to be elevated, as many of them have components on both sides and they have to be level as if the pictures are taken on a certain angle on Z axis errors due to projection are introduced.

5.3 Camera

Theoretically, for any image acquisition system, the higher the resolution of the camera, the more details can be captured in the pictures so a more accurate recognition can be performed. In practice, a compromise has to be reached between the price and the resolution of the camera. Also, even if the camera can produce high resolution images, inconveniences may appear: the higher the resolution of the pictures the more computing power is needed to process them (the computer has to deal with bigger matrixes).

Regarding all the considerations above, the camera chosen for the project is a LCD (liquid crystal display), Casio QV–3000EX digital still camera (figure 5.3).

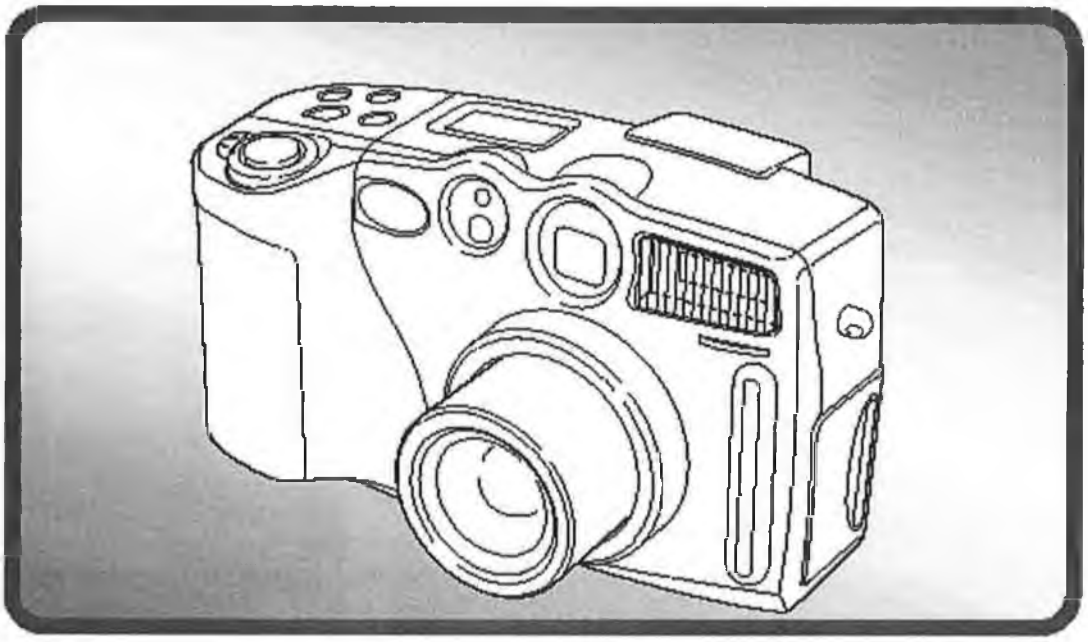


Figure 5.3: Casio QV–3000EX.

The camera has the following specifications:

• *Maximum resolution:*	*2048x1536 pixels*
• *Total pixels:*	*3.34 millions*
• *Lens:*	*33–100mm*
• *Memory:*	*4Mb*

• *Optical zoom:*	*3x*
• *Digital zoom:*	*6x*
• *Shutter:*	*1/1000s (CCD electronic shutter)*
• *Image format:*	*JPG and AVI (for movie mode)*
• *Computer interface:*	*USB*

5.4 The Software – “Image Acquisition”

The software application “Image Acquisition” is the result of the research; it is a complete software solution able to acquire and analyze the pictures, to perform the filtering, build the libraries, process the images and generate log files.

The application was designed to work with Intel (or other compatible) based processors, on a Microsoft Windows (NT/2000/XP) platform.

“Image Acquisition” was built with Visual C++ 6.0 and the algorithms with Matlab 6.1. At the final stage of the project, the program was recompiled with Visual C++ 7.0 (.NET). The algorithms were compiled in a Dynamic Link Library (DLL). The functions contained in the library are imported at the run–time by the application.

The program requires only one qualified operator in order to run. The base idea of the program is quite simple. A PCB is visually inspected by the operator; once he makes sure the board is defect free, a picture is taken. This picture will be called *main picture*. The main picture will be used as reference for all the other boards that are similar with it. After the picture is taken, it will be imported into the application. Now a library has to be built for this particular board. The user selects the areas of interest (the components that need to be analyzed) and saves them to the library. This procedure has to be done only once for a certain model of PCB. When the library is build, pictures of unverified boards, called *target pictures*, are taken and passed to the application in order to be checked for defects. The operator has the freedom to modify the library at any time. The only thing that he has to do after the libraries are built is to make sure that he correctly associates the target pictures with the main picture. The results obtained after the target picture is analyzed, are stored in a log file (that contains a detailed description of each component analyzed and also statistics like time needed or total number of components) associated with the name of the target picture. The defects can also be visualized on a graphic form on the screen.

In general, there are no upper limits regarding the computing power of an application dedicated to image processing. The rule is: the faster the processor and the larger the amount of memory, the faster the application will run. In order to create a general idea about the system requirements, an example of a minimal computer configuration might be helpful:

• Processor:	Intel (or compatible) running at 300MHz
• RAM:	128 Mb
• Video Card:	8 Mb of RAM, capable to display true color images (at least 24bpp)
• Operation System:	Windows NT 4.0 (Service Pack 6)

The application has been tested on an Intel Pentium III at 866 MHz with 512 Mb of RAM and 64 Mb of video memory, using Windows XP. The maximum time (calculated for the slowest filter) needed to process a color selection of 60x30 pixels in size was about six seconds. The maximum amount of memory required by the application is 16Mb (the code) plus six times the size of the image that has to be processed (i.e. given an image of 9Mb, the total amount of memory used is 16+9x6=70 Mb).

The following sections will describe thoroughly all the elements of the application.

5.5 *Image Acquisition – User interface*

The application is divided in three main sections:

- ***Select Components*** – Is the area of the application where the components are selected for analysis, the type of analysis is selected for a specific element and where the components are saved to library.
- ***View Library*** – Is the area where the components saved in the previous section can be visualized, modified or deleted as needed by the user.
- ***Analyze*** – In this section all the components in the library are analyzed for defects and log files are generated.

The following picture presents a screen capture of the application main window:

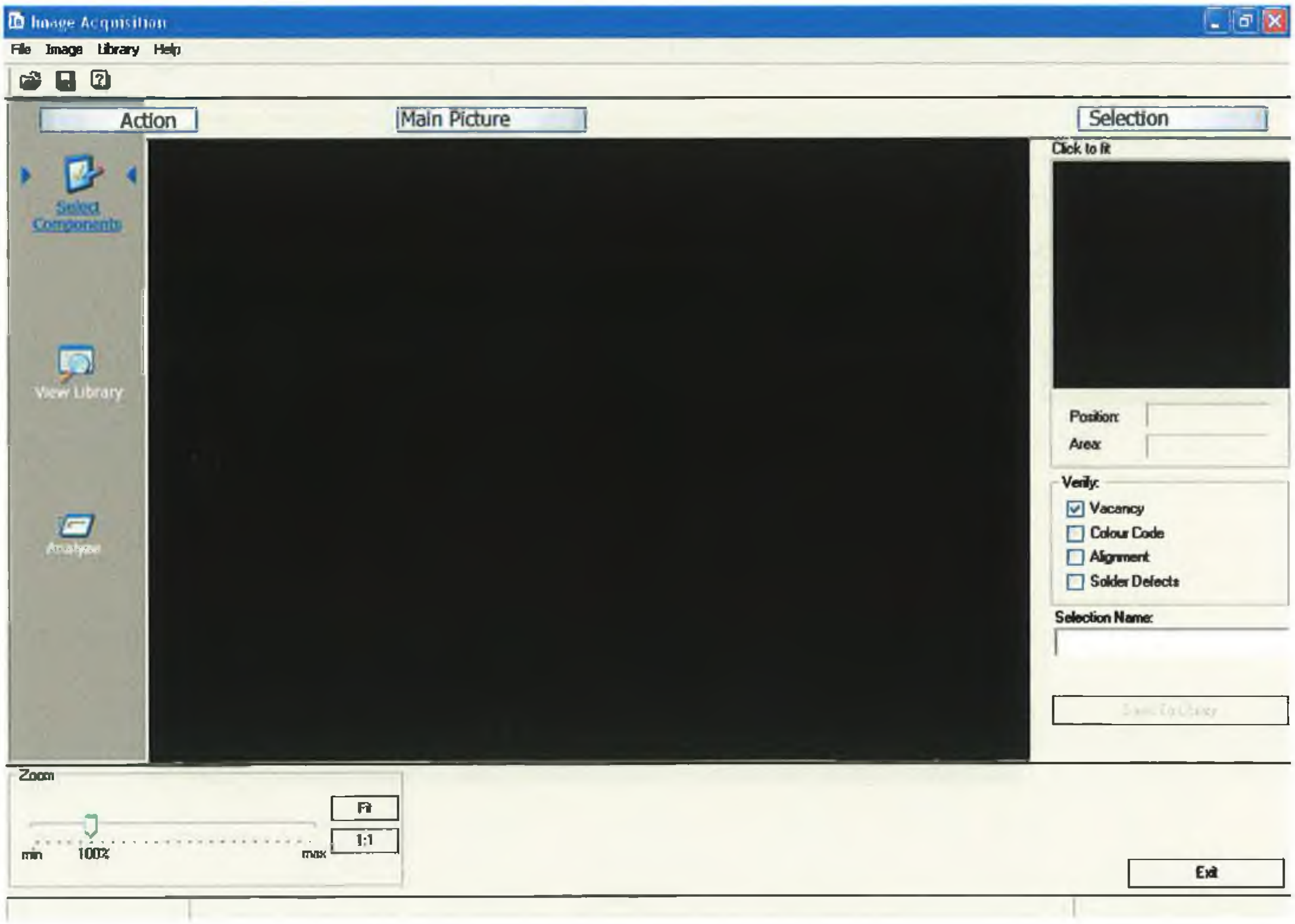


Figure 5.5: Application main window

The elements of the application in *Select Components* mode are:

1. *Main Picture holder*
2. *Selection Picture holder*
3. *Selection Picture options panel*
4. *Application title*
5. *Menu bar*
6. *Tool bar*
7. *Status bar*
8. *Control Panel*
9. *Zoom panel*

5.6.1 *Main Picture holder*

The main picture contains an image of a board that was previously visually analyzed by an operator. This operation ensures that the board is defect free so it can be used as a template for all the other boards that will be analyzed, the *target pictures*. In order to load a picture the user can use the *menu bar (5)* by selecting **File → Open** (figure 5.6.1.1).

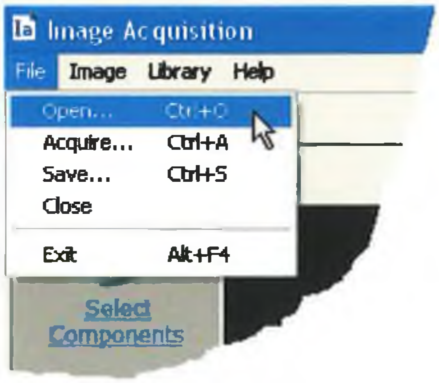


Figure 5.6.1.1: **File → Open** menu

A picture can also be loaded by using the *tool bar (6)* by clicking on the open icon (figure 5.6.1.2).

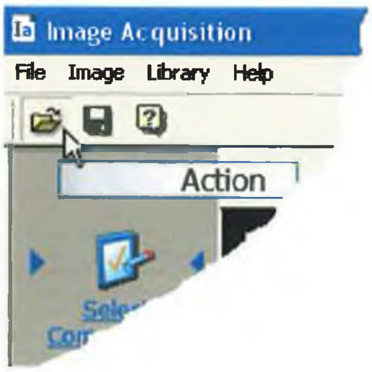


Figure 5.6.1.2: **Toolbar → Open** menu

Once this operation is performed a standard Windows *Open Dialog Box* (figure 5.6.1.3) is displayed prompting the user to select the picture to open.

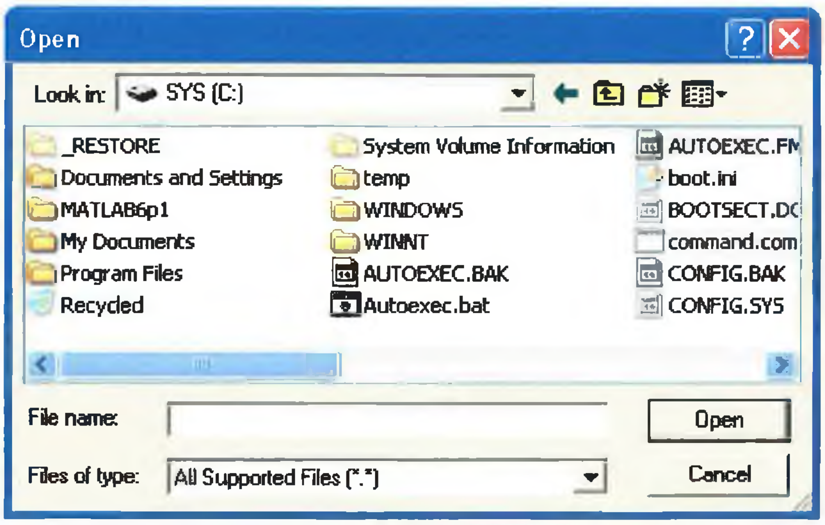


Figure 5.6.1.3: *Open File* Dialog Box

The application is able to recognize among more than 30 image formats which includes file support for reading BMP, DIB, EMF, PIC, GIF, ICO, JPG, CMYK–JPEG, PCX, PNG, RLE, TGA, TIFF, WMF image file formats. However some of these file formats are not recommended to be used as they contain loose compression algorithms which deteriorates the images in order to save disk space (section 4.7.1). In case the image is corrupted or has an unknown format, the application will generate an error message stating the cause. Also a filter by extension can be applied by selecting from the "*Files of type:*" field a certain extension.

In general it is not recommended to use compressed image formats even if the compression is very small or none and data is not lost. The purpose of compressed images is to save disk space but the price to pay is image quality (which is poorer). Even when the image is compressed using lossless algorithms, not one byte of memory is saved as the image in memory has the same size as an uncompressed image. For instance a JPG file which on disk has about 300 Kb, once it's loaded in memory will occupy about 2.3 Mb. The application is able to provide this information by selecting from the *menu bar (5)* **Image → Information** (figure 5.6.1.4).

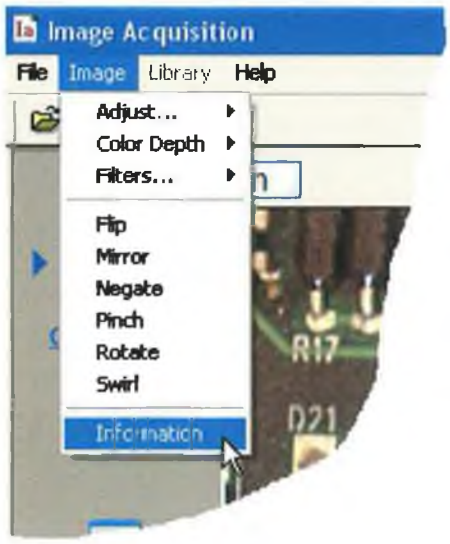
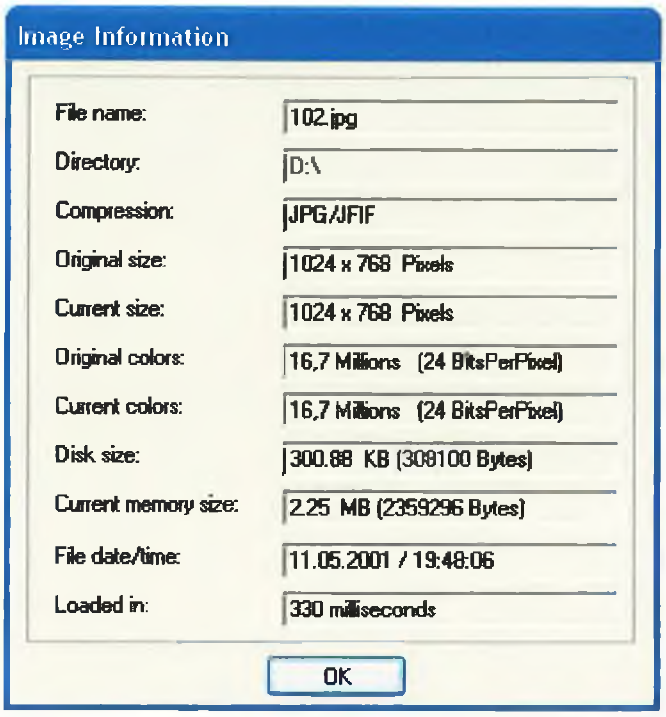


Figure 5.6.1.4: **Image → Information** menu

It is obvious that an image has to be previously loaded otherwise an information message will be generated stating that an image has to be loaded prior displaying information about it. Once this action is performed a window containing information about the file will be displayed:



File name:	102.jpg
Directory:	D:\
Compression:	JPG/JFIF
Original size:	1024 x 768 Pixels
Current size:	1024 x 768 Pixels
Original colors:	16,7 Millions (24 BitsPerPixel)
Current colors:	16,7 Millions (24 BitsPerPixel)
Disk size:	300.88 KB (308100 Bytes)
Current memory size:	2.25 MB (2359296 Bytes)
File date/time:	11.05.2001 / 19:48:06
Loaded in:	330 milliseconds

Figure 5.6.1.5: Image information window

The frame displays information about the file name, location, compression type, original size in pixels, current size in case image is zoomed, the original number of colors, the current number of colors in case image was resampled, the space occupied on disk, the space occupied on RAM, the date and time when the file was created as recorded on the File Allocation Table (FAT) of the system and the time needed to load the image.

Another way to load a picture is by using acquire method located on the *menu bar (5)* by selecting **File → Acquire**:

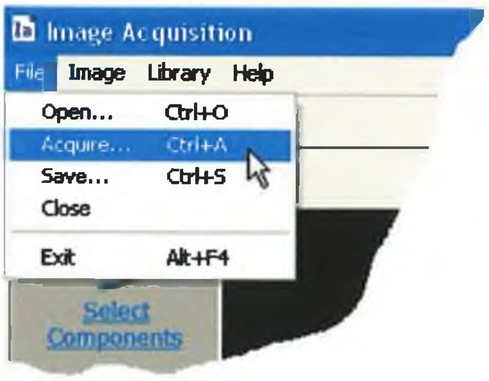


Figure 5.6.1.6: **File → Acquire** menu

The *Acquire* method will grab an image captured form certain scanners, digital cameras, and frame grabber devices using TWAIN cross – platform interface. This requires that this devices support TWAIN interface and proper drivers are installed on the system. In case that more than one TWAIN device is installed a window will pop–up asking the user to select the desired device. If no TWAIN devices are found on the system an error message will be thrown stating the reason. If the image is successfully acquired it will be displayed as main picture. The image has to be saved in order to be able to process it.

Once the command to open a valid image file is given using either a file from the hard disk or an image acquired from an external device, the *status bar (7)* is updated; it will show on the left side the action performed (*loading file...* or *acquiring image...*), the middle area will show a progress bar which indicates loading progress and the area on the right side will display the pixel position where the tip of the mouse pointer is located. The progress bar is particularly useful in this case when an image is acquired using TWAIN interface as this operation is often slow depending on the speed of the input device. Once image upload is complete the left area describing the action and the progress bar will be cleared until another action is performed.

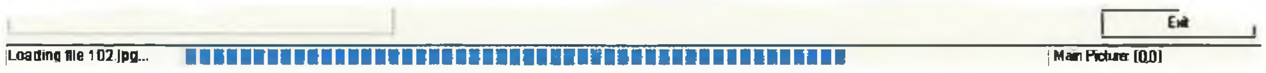


Figure 5.6.1.7: File load progress on the *Status Bar*

Also, just after the picture is uploaded, the program searches for the database associated with the graphic image. The database has the same name with the image but the extension is TXT. If no database is found a new one will be created. The structure of the database and the way it is updated, modified or maintained will be described later on in this chapter on *View Library* section.
In the same time the area just after *Main Picture* label will be updated with the name of the picture.

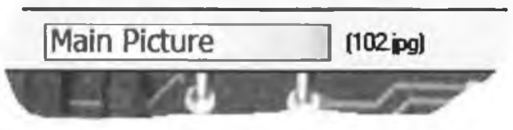


Figure 5.6.1.8: Main Picture label and the file name

Everything is set now and the program is ready to process or analyze the image.

5.6.2 Selection Picture holder

The selection picture is a temporary image buffer that contains the components selected by the user in order to save them in the database. The user selects a component from the main picture. When the mouse is moved over the main picture the pointer turns in a cross – hair (+). By left mouse button click – and – drag a rubber band will appear around the selected area, as shown in figure 5.6.2.1:

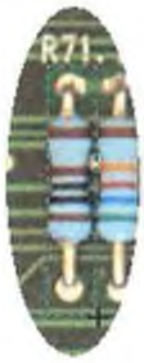
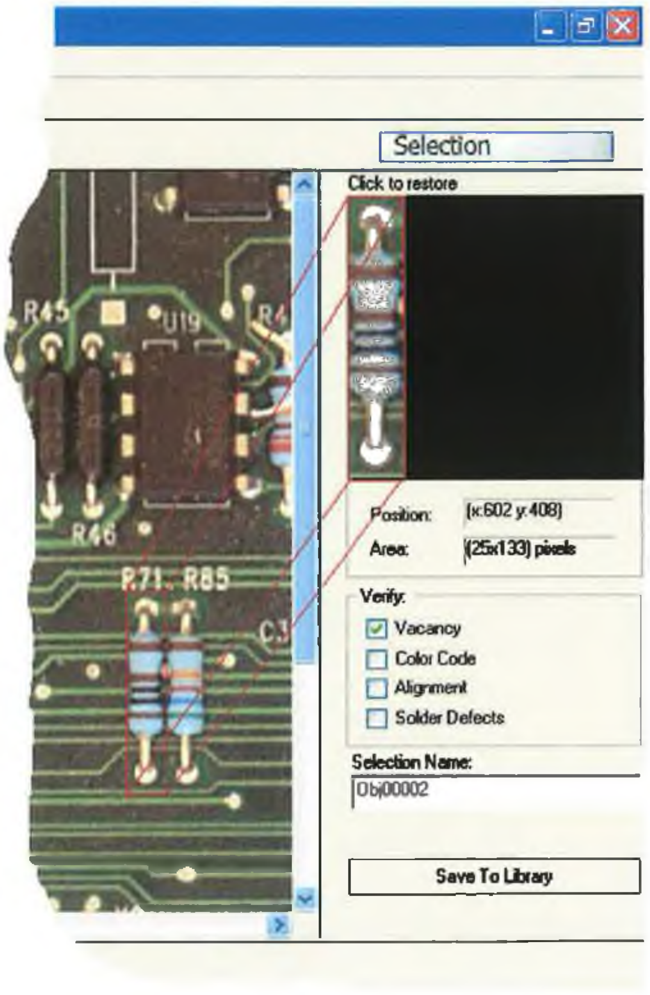


Figure 5.6.2.1: A selected component

Once the button is released the area inside the rubber band is copied on the *selection picture:*



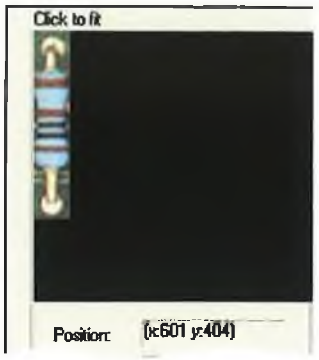
Position: (x:602 y:408)
Area: (25x133) pixels

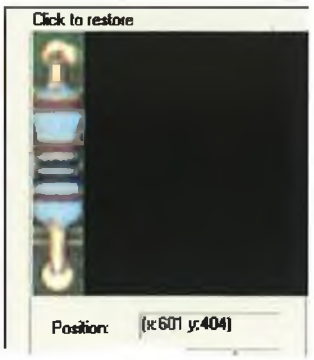
Verify:
- [✓] Vacancy
- [ ] Color Code
- [ ] Alignment
- [ ] Solder Defects

Selection Name: Obj00002

Figure 5.6.2.2: Component pasted onto selection holder

The red frame doesn't appear on the screen; in this screen shoot it just illustrates the copy process. If the user clicks the left mouse button on the selection picture it will toggle between original size and a magnified version:



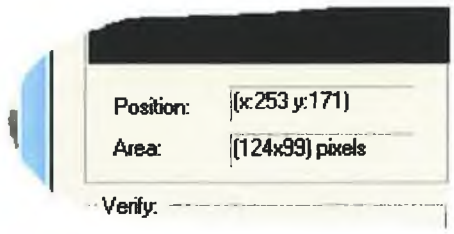


a) Normal view b) Magnified view

Figure 5.6.2.3: Selection picture

5.6.3 *Selection Picture Options Panel*

When the user decides that one component is properly selected, the options for that component can be adjusted. First, just below the *Selection Picture* some information is displayed about the selection:



Position: (x:253 y:171)
Area: (124x99) pixels
Verify:

Figure 5.6.3.1: Position and area

The boxes state the initial *x* and *y* position of the selection in the *main picture* and the area of the picture in pixels. This are provided so the selection can be pin – pointed within the *main picture*.

The next step is to decide what defects will be analyzed. The user can choose between *Vacancy, Color code, Alignment* and *Solder defects*, by ticking the appropriate check box.

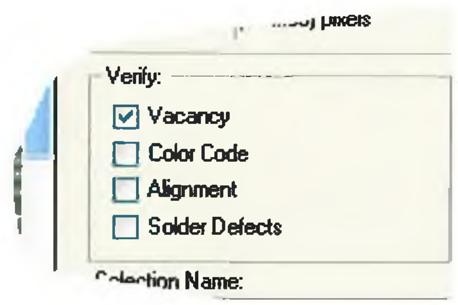


Figure 5.6.3.2: Defects checking options

Vacancy is selected by default, as it can be applied in conjunction with any of the other types of defects, exempt the solder defects. It is very important to select the right type of defect checking for each selection in order to maximize the program efficiency. More about how the selection should be made is described later on this chapter on section 5.9.

When the user decides what kind of defects will be checked for the selection all is left to do is to set a name for the selection and to save it into the database.

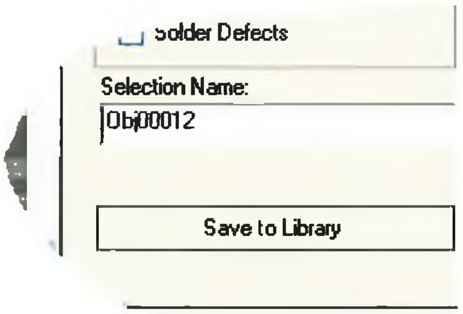


Figure 5.6.3.3: *Selection Name* and *Save to Library*

The program will automatically propose an unique name for each selection and fill it into the *Selection Name* edit box. However, the user is free to choose any name for the selection and override the name proposed by the program. In case the user types a duplicate name, the program doesn't for duplicates at this stage as duplicate names are useful for those types of boards where equivalent components are used. For instance two different boards may use resistors having the same physical properties (same resistance) but they look totally different in terms of color or size.

The *Save to Library* button is disabled until the selection is made then, when pressed, the selection is saved into the database and the button disable itself until the next selection is made.

5.6.4 Application title

The application title is a standard Windows title bar containing the system menu for the program, the title and the three control buttons: minimize, maximize/restore and close.

5.6.5 Menu bar

The main menu contains four items: *File, Image, Library* and *Help*. Some of the items in the menu were already described in the previous sections. When the program runs in *Select Components* mode, the *Library* menu item is disabled, when it runs in *View Library* mode, the *Image* item is disabled and when it runs in *Analyze* mode both *Image* and *Library* items are disabled. This is done to protect accidental modification of the images and to delimitate the sections of the program more clearly. On the *File* submenu there are three more items to describe:

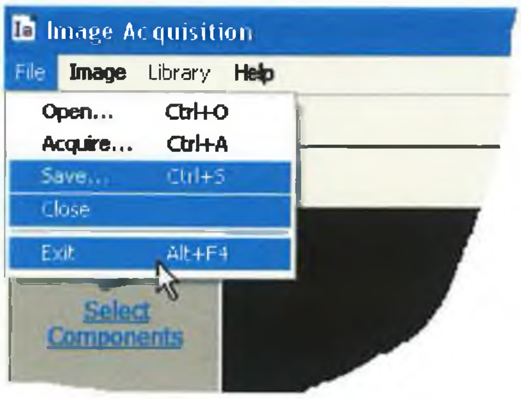


Figure 5.6.5.1: **Menu** → **File**, more options

- *Save* – Will save on the hard disk any modifications brought to the *main picture.* A standard *Save Dialog Box* is displayed prompting the user to select the name and path of the picture to be saved.

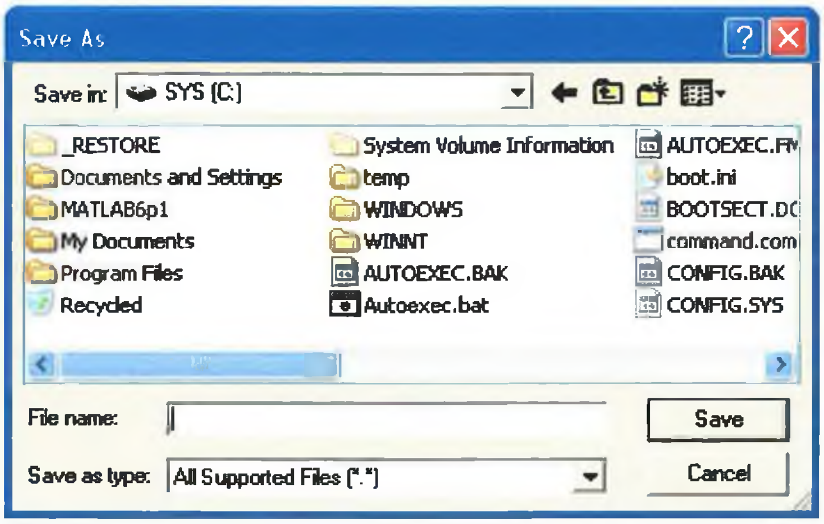


Figure 5.6.5.2: Standard Windows *Save Dialog Box*

For saving, the same file extensions are supported like in the *Open Dialog Box,* only this time the user types the name and the extension of the desired output file. In case an unknown extension is typed, an error message will be thrown stating that the program is unable to save the image under that format.

- *Close* – Will flush all buffers to disk, close all the files including the *main picture* and the *library* files but the program doesn't quit. This is useful when the user decides to start working on another picture.
- *Quit* – Will perform the same action as *Close* menu item, but after closing all the files the application will release the memory and quit.

The *Image* submenu contains a collection of filters and transformations that can be applied to *the main picture.* All the filters and transformations will update the *status bar (7)* by displaying the action (the name of the filter) on the left and a *progress bar* on center, indicating the progress. The changes made by a filter are permanent when the user presses *OK* button. If *Cancel* button is pressed the image will be brought to its original state. Also by pressing any of the *OK* or *Cancel* buttons will remove the filter frame from the screen. The *Image* submenu is available only when the program runs in *Select Components* mode.

a) *Adjust...*

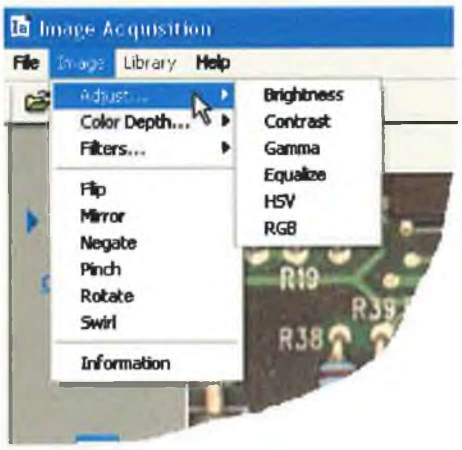


Figure 5.6.5.3: *Adjust...* filters group

- *Adjust...* → *Brightness* – Will enhances the brightness attributes of the pixels in the image. Once this item is selected, a new frame will appear on the bottom of the main window prompting the user to adjust the brightness.

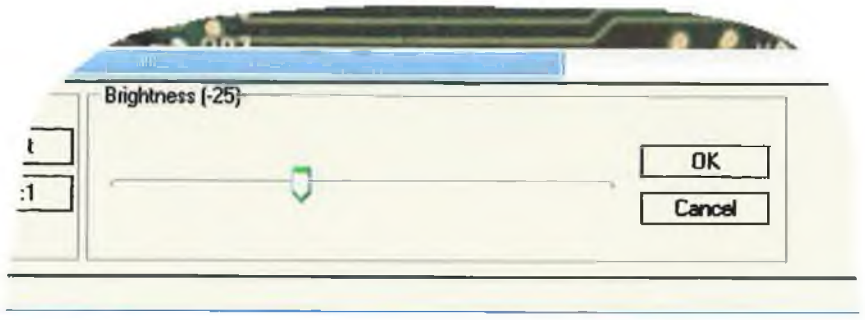
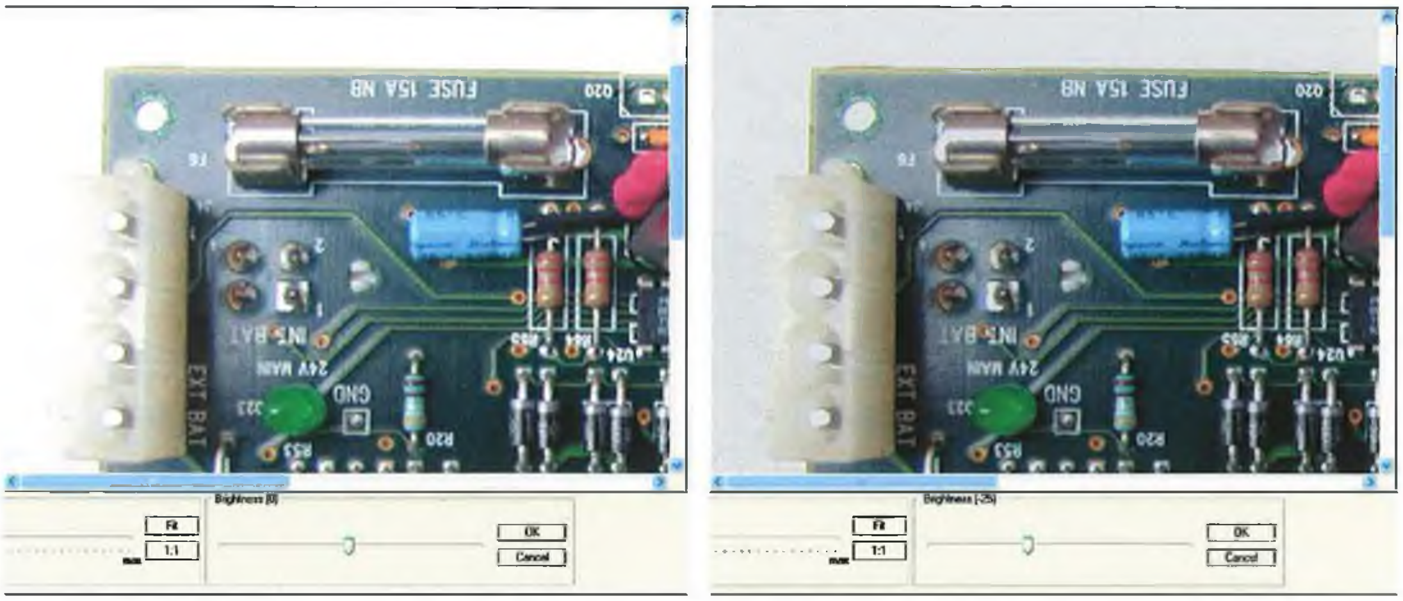


Figure 5.6.5.4: Brightness Filter

By moving the slider to the left the brightness will be decreased, and by moving it to the right left the brightness will be increased. The range of the slider is from –100 to +100 percent, by default the slider is set to 0.

The following images show the brightness filter set at –25% applied to an image:



a) Brightness – before filter b) Brightness – after filter (–25%)

Figure 5.6.5.5: Brightness adjustment

- *Adjust...* → *Contrast* – Will adjust the contrast of the main picture. Images with a higher contrast have a higher frequency of pixel intensities near the ends of the intensity spectrum while images with a lower contrast have a higher frequency of pixels near the middle of the intensity spectrum. A more detailed description of how contrast works is given in section 2.3.1 Like in previous case, once this item is selected, a new frame will appear on the bottom of the main window prompting the user to adjust the contrast.

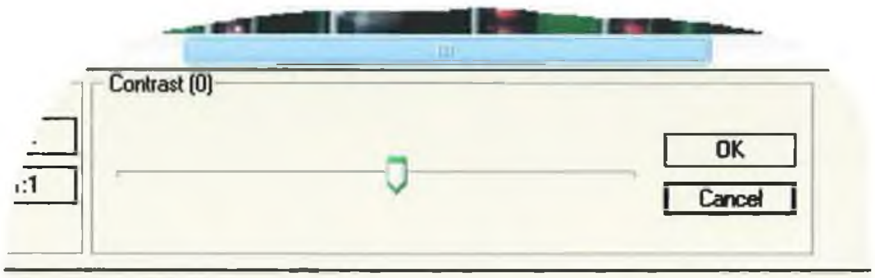
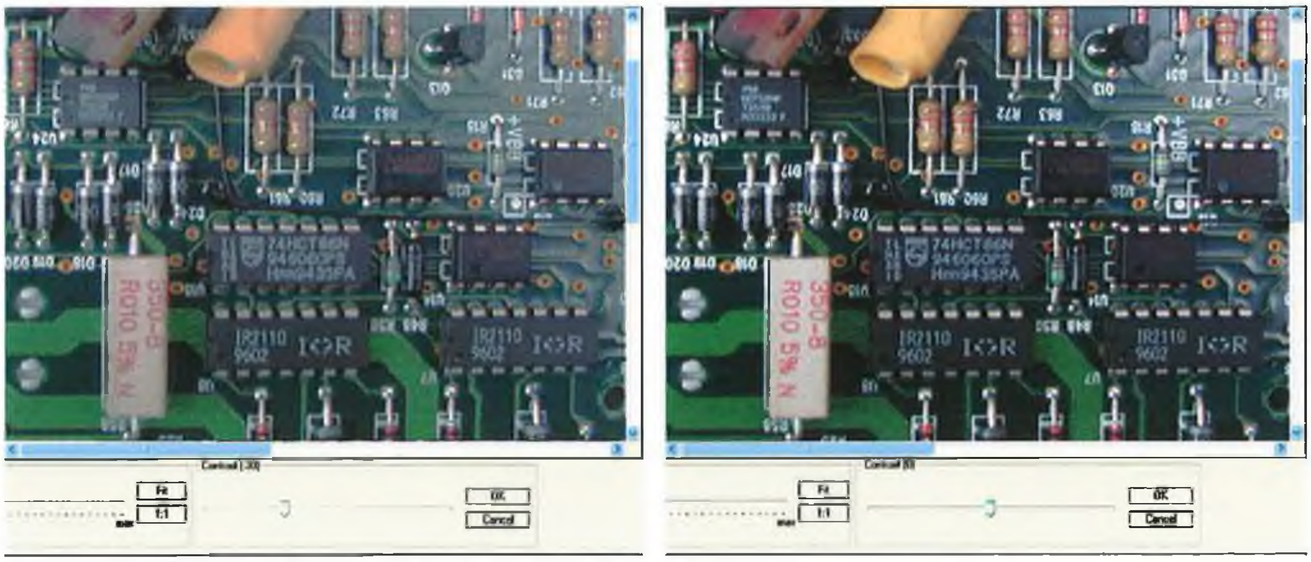


Figure 5.6.5.6: Contrast Filter

By moving the slider to the left the contrast will be decreased, and by moving it to the right left the contrast will be increased. The range of the slider is from –100 to +100 percent, by default the slider is set to 0. The following images show the contrast filter set at –30% applied to an image:



a) Contrast – before filter b) Contrast – after filter (–30%)

Figure 5.6.5.7: Contrast stretching

- *Adjust... → Gamma* – Increases or decreases the intensity of the image using the gamma function by raising or lowering the brightness midpoint. Gamma values greater than *1.0* produce a brighter image. Gamma values less than *1.0* produce a darker image.

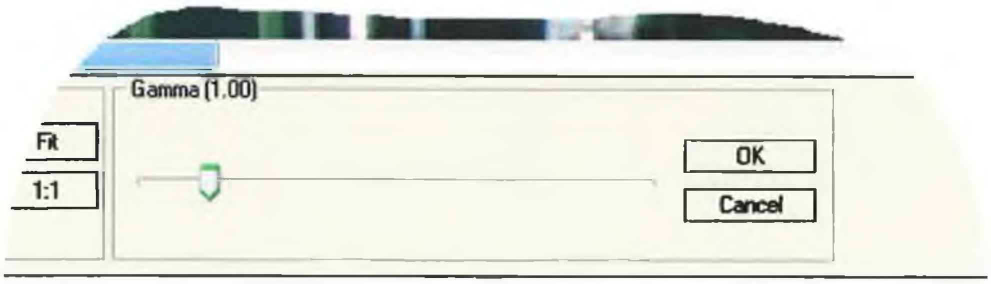
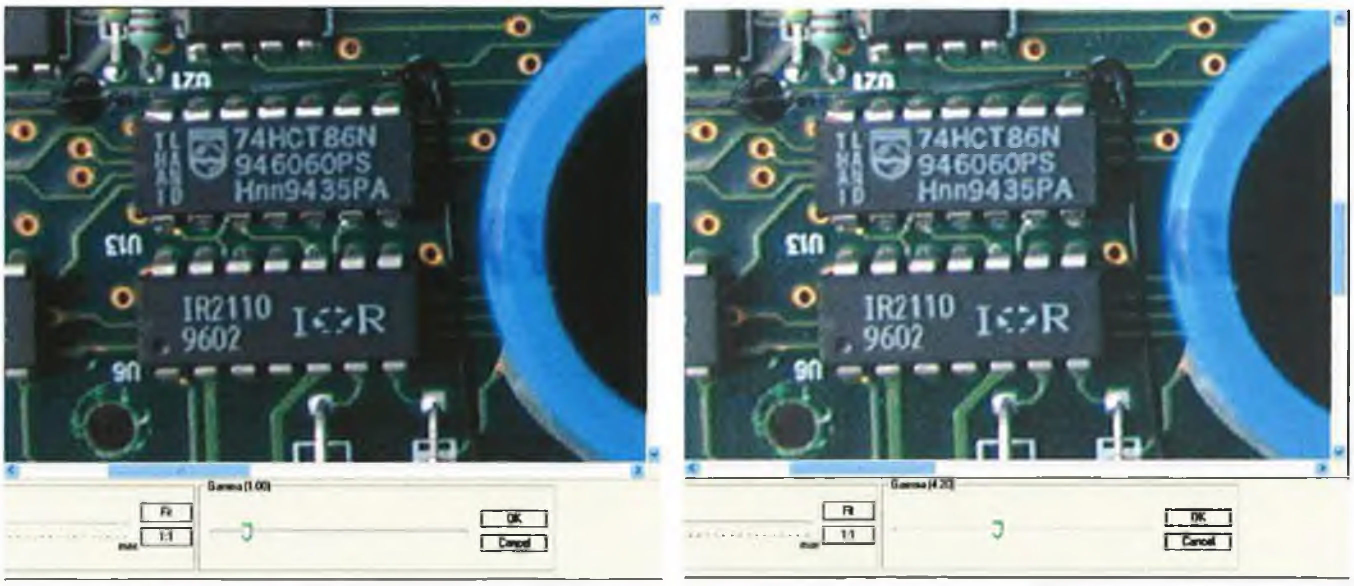


Figure 5.6.5.8: Gamma Filter

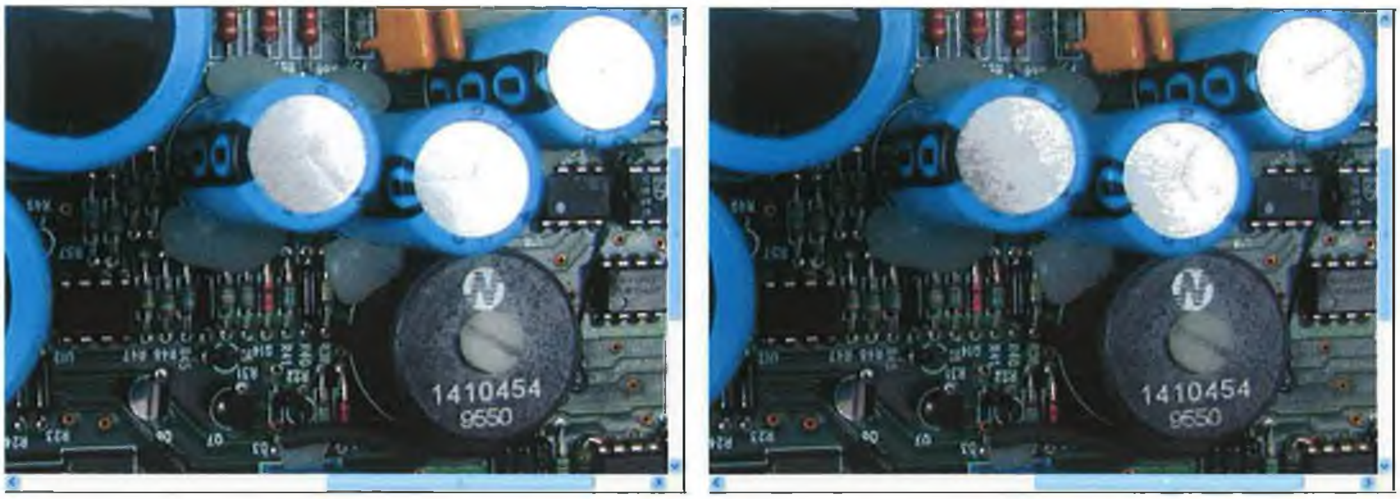
The range of the slider is from *0* to *10*, by default the slider is set to *1*. The increments of the slider are in *0.01* units. The following images show the gamma filter set at *4.20* applied to an image:



a) Gamma – before filter b) Gamma – after filter (4.20)

Figure 5.6.5.9: Gamma operation

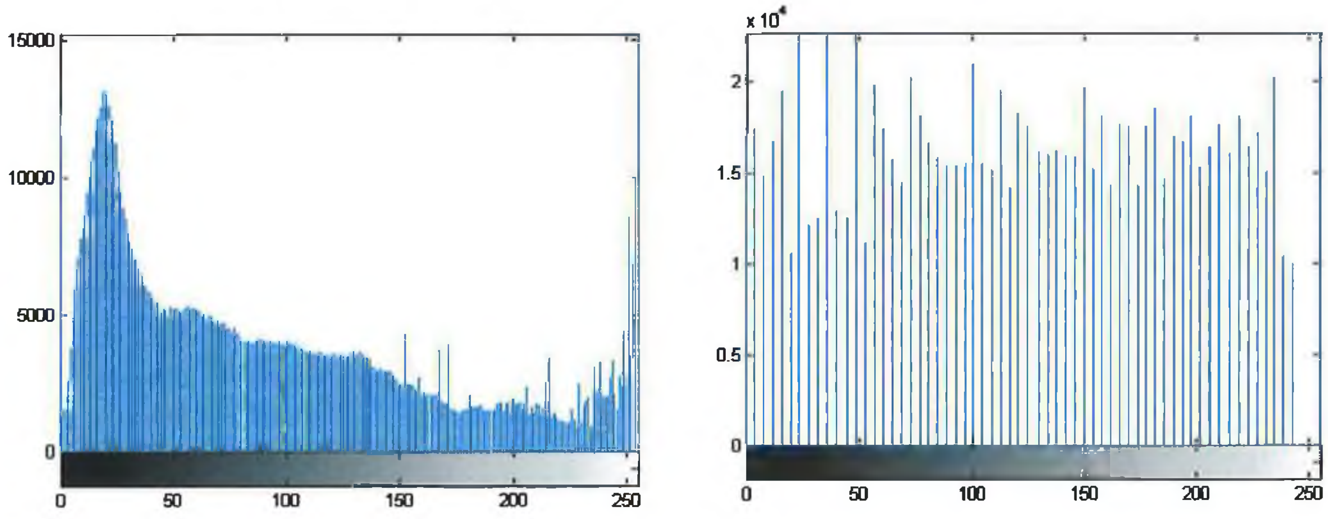
- *Adjust... → Equalize* – Performs contrast equalization on the current image as described in section 2.3.2. A frequency histogram of the colors of the image is computed and adjusted to equalize the colors across the full spectrum of intensity values. The equalized colors are applied to each pixel in the image. The following images show the equalize filter applied to an image:



a) Equalize – before filter b) Equalize – after filter

Figure 5.6.5.10: Equalize filter

A better view of the effects of the *Equalize* filter can be obtained by displaying the histogram graph:



a) Equalize – histogram before filter b) Equalize – histogram after filter

Figure 5.6.5.11: Histogram equalization

On the vertical axis is the frequency, and on the horizontal axis is the color.

- *Adjust...* → *HSV* – Will adjust the *Hue, Saturation* and the *Value* of the picture. The *Hue* component controls the overall color space of the image. Adjusting the hue of the image adjusts the color spectrum from red through yellow, green, blue and violet; *Saturation* controls the purity of a color. Higher saturations result in richer and deeper colors. Lower saturations result in paler colors; *Value* controls the overall brightness of the image in a similar way with the brightness filter.

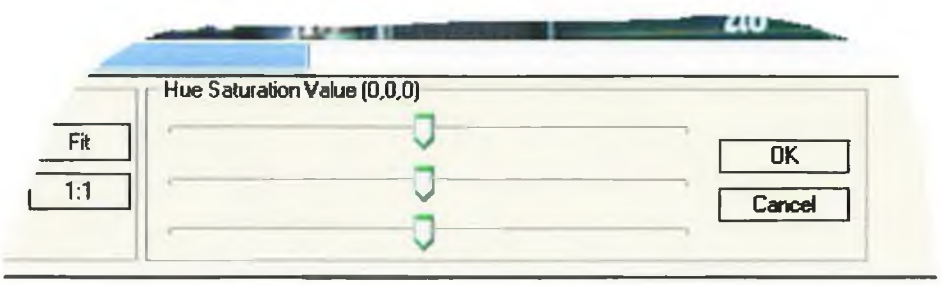
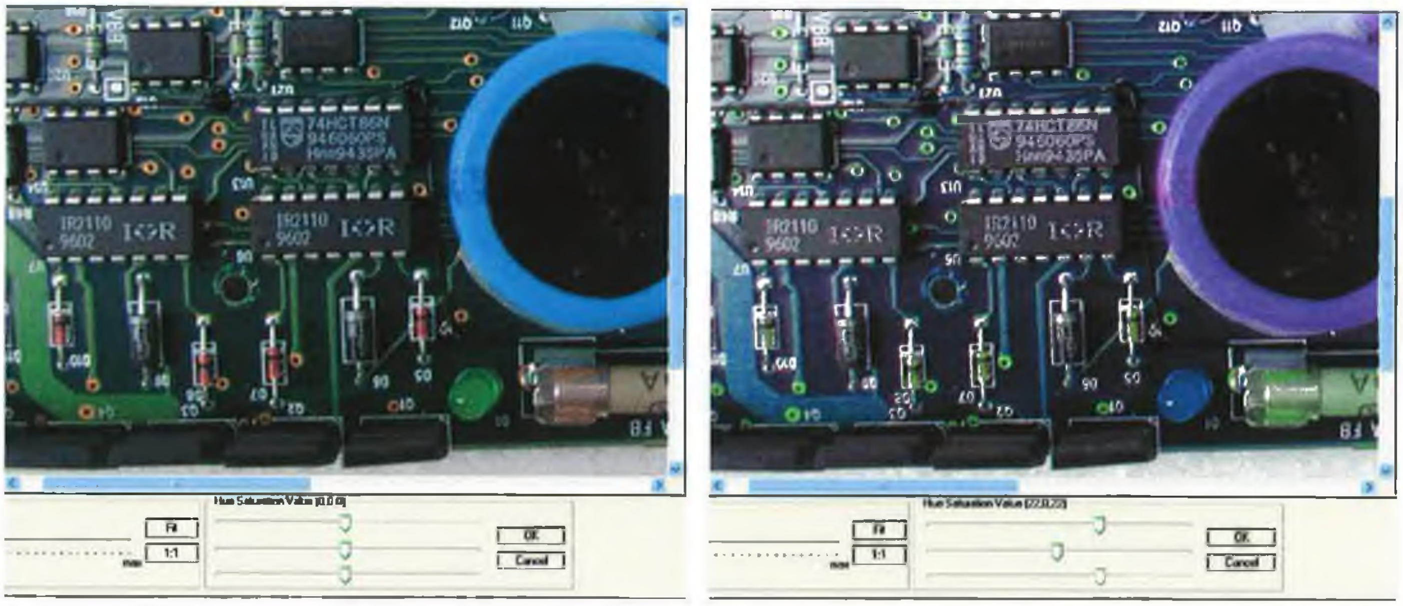


Figure 5.6.5.12: HSV filter

The range of each slider is from *–100* to *+100*, by default the sliders are set to *0*. The following images show the HSV channels set to *H=22, S=0, V=22* applied to an image:


Hue Saturation Value (0,0,0)

Hue Saturation Value (22,0,22)

a) HSV channels – before filter b) HSV channels – after filter (*H=22, S=0, V=22*)

Figure 5.6.5.13: HSV adjustment

- *Adjust... → RGB* – Will adjust the *Red, Green* and *Blue* channels for a *24 bits* image. Each pixel or data point in a true color (*24 – bits*) image contains these *3* components that describe the color of the pixel. Each of the components can have a value between *0* and *255*, *0* being the darkest shade of color (black) and *255* being the lightest shade of color (white).

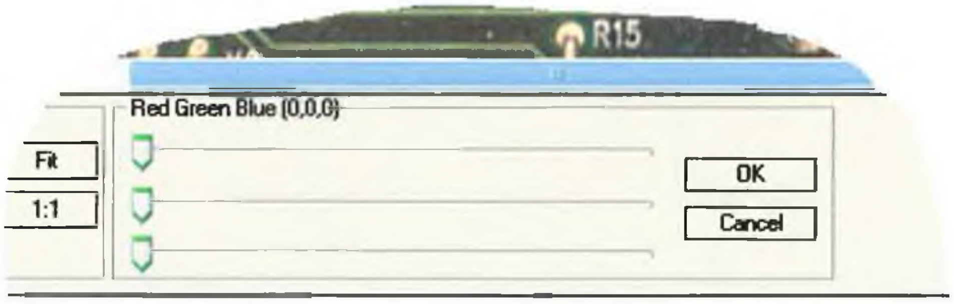
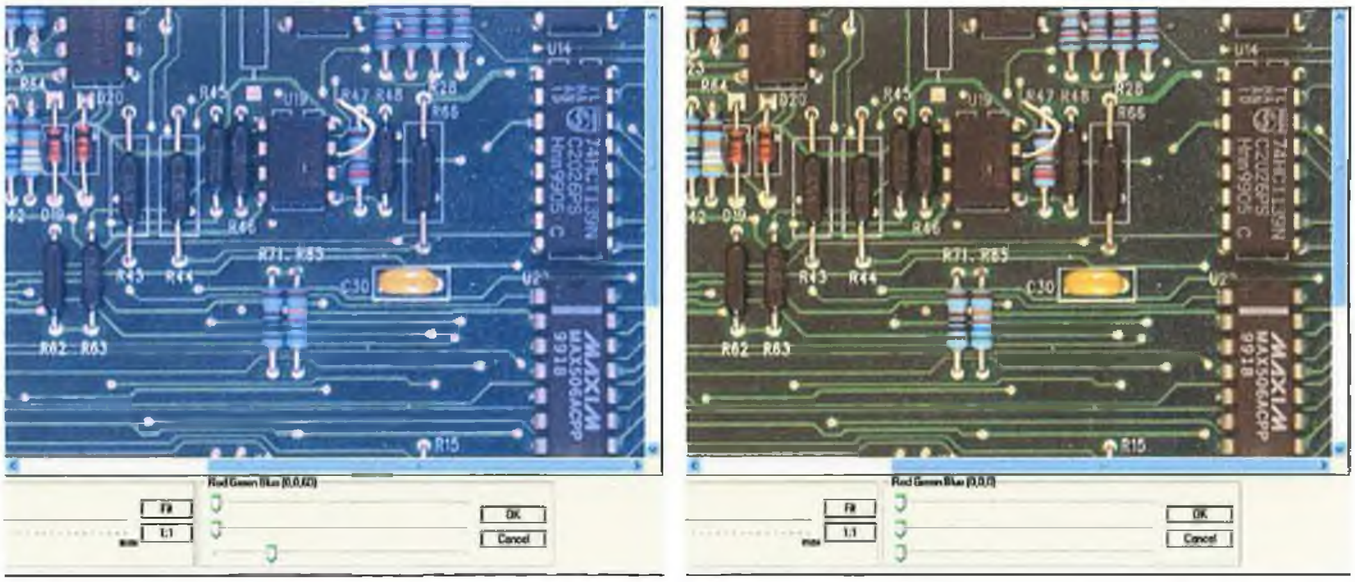

Red Green Blue (0,0,0)

Figure 5.6.5.14: RGB filter

The range of each slider is from *0* to *255*, by default the sliders are set to *0*. The following images show the RGB filter set to *R=0, G=0, B=60* applied to an image:



a) RGB channels – before filter b) RGB channels – after filter (*R=0, G=0, B=60*)

Figure 5.6.5.15: RGB adjustment

b) ***Color Depth...*** – adjusts the number of *bits per pixel* (bpp) used by a picture.

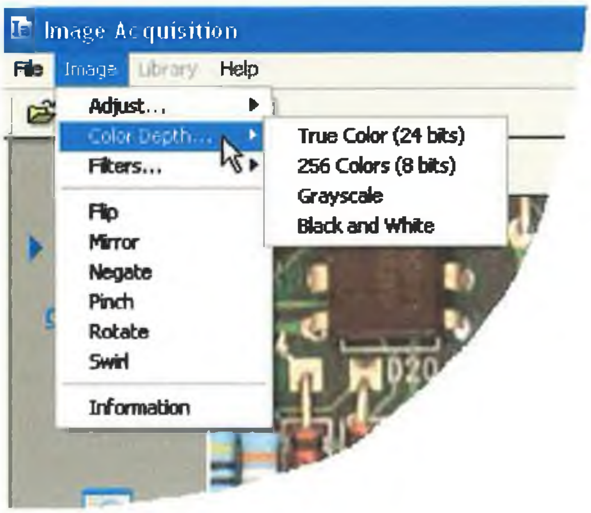


Figure 5.6.5.16: *Color Depth...* filters group

- *Color Depth... → True Color (24 bits)* – Will resample the current image to a 24 bits per pixel image. If the image is already on 24 bits mode the filter has no effect. In *True Color* image, each pixel occupies 24 bits on memory, 8 bits for each of the *Red, Green* and *Blue* channels ($8 \times 3 = 24$). It means that each pixel will have $2^8 = 256$ shades of red, green and blue which gives a total of $256^3 = 16\ 777\ 216$ number of possible colors per pixel. The following picture shows an image in true color mode:

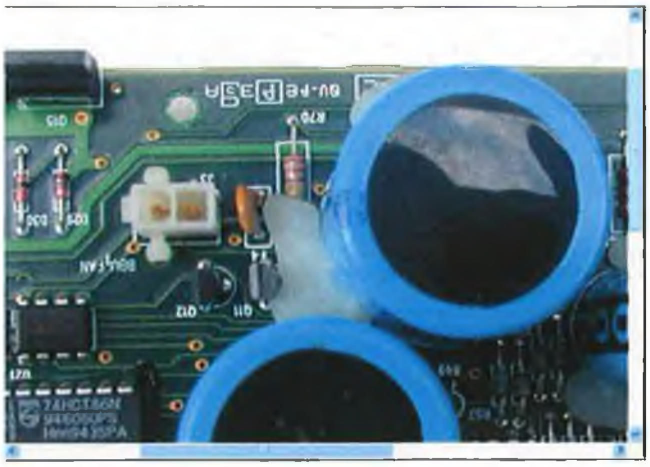


Figure 5.6.5.17: A true color image

- *Color Depth... → 256 Colors (8 bits)* – Will resample the current image to *8* bits per pixel image. A pixel with a bit depth of *8* has 2^8, or *256*, possible colors. These colors can be any *256* of the *16* millions colors of a true color image. That's why *color maps* are defined in order to build sets of colors. The program uses the system color map which is predefined by the operating system. The following picture shows an image in *256* colors mode:

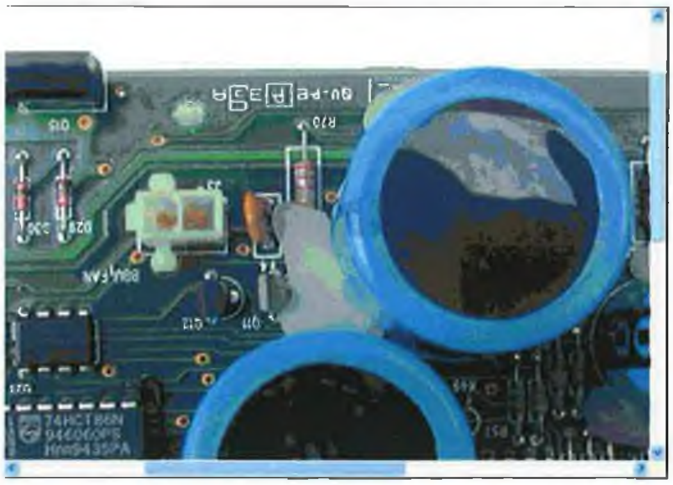


Figure 5.6.5.18: A 256 colors image

- *Color Depth... → Grayscale* – Will resample the current image to a grayscale image. Grayscale images are similar to *256* color images, but it uses a color map with *256* shades of gray. The following picture shows an image in grayscale mode:

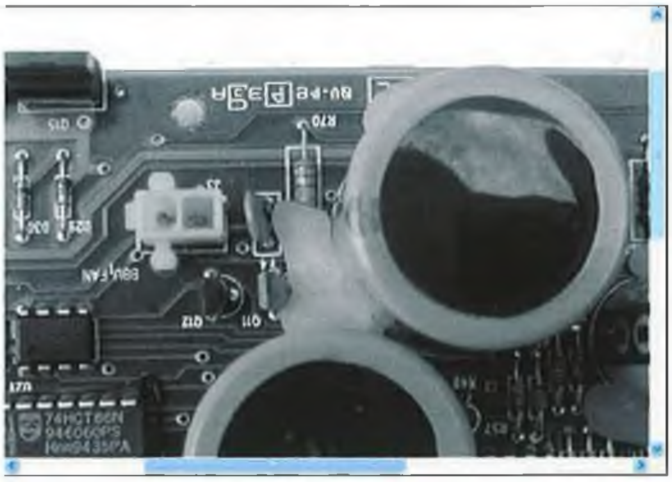


Figure 5.6.5.19: A grayscale image

- *Color Depth... → Black and White* – Will resample the current image to a black and white image. These images have only one bit per pixel where "1" is white and "0" is black. The following picture shows an image in black and white mode:

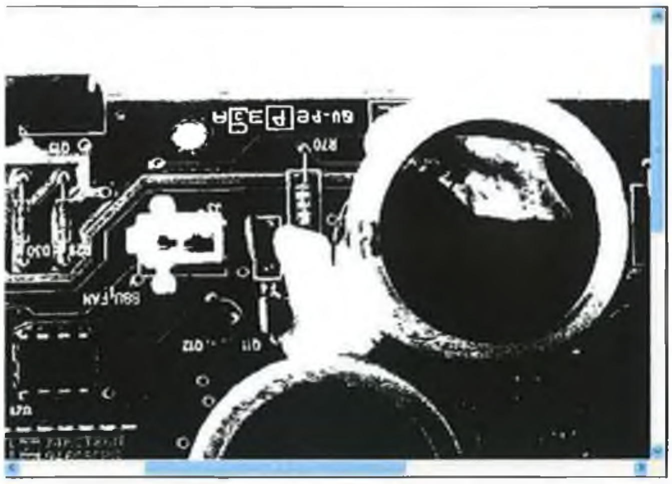


Figure 5.6.5.20: A black and white image

The changes made to the color depth are permanent and cannot be reversed. Bringing a picture from a lower bit depth to a higher one will only change the internal representation of the pixel, will not actually change its color or enhance the quality of the picture, what so ever.

There are other types of color depths defined as standards but the program doesn't use them. Other color depths are used for artistic purposes or to manage the output of the picture on a printer or plotter. It is worth to mention some other color depths widely used: 32 bits per pixel images will have the same structure as a 24 bits image regarding the number of colors but they will have an extra 8 bits for each pixel used, for transparency purposes. This raises the total number of colors to $256^4 = 4\ 294\ 967\ 296$, so a pixel may have over *4.2* billion different colors. 16 bits per pixel image is able to display $256^2 = 65536$ colors.

c) ***Filters…***

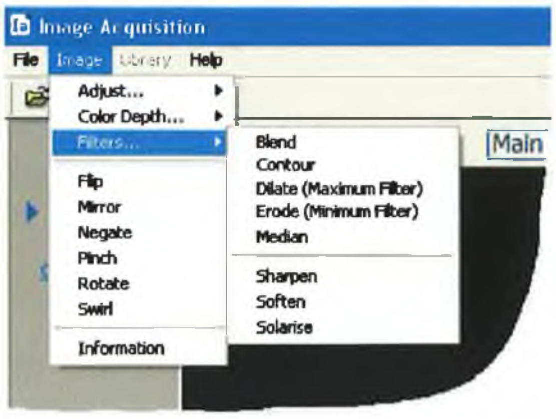


Figure 5.6.5.21: *Filters…* group

- *Filters…* → *Blend* – Will blend each pixel in the current picture with its neighboring pixels. Blend is a *linear filter* (part of the smoothing filters discussed on section 2.5) and can be used to reduce the noise in images. The filter has been introduced as an opposite to *Sharpen* filter.

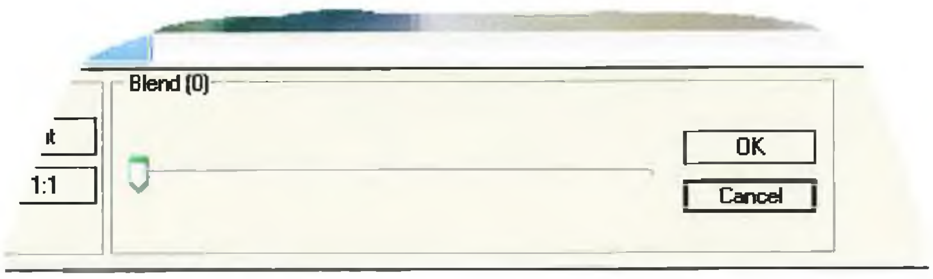


Figure 5.6.5.22: Blend filter

The slider specifies the percent weighting of the neighboring pixels. Higher values result in more blending; lower values result in less blending. Valid values are 0 through 100, the default value is 0. The effects of blend filter can be best seen on magnified images; the following images show the blend filter set to 75% applied to an image:

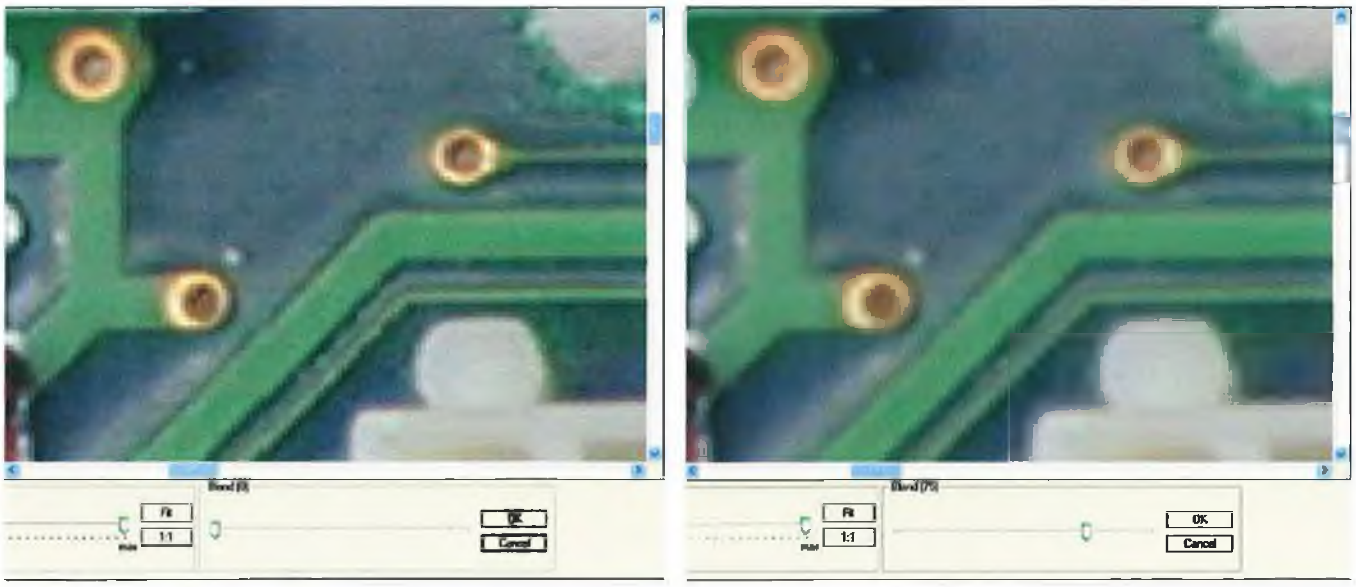

a) Blend – before filter b) Blend – after filter (75%)

Figure 5.6.5.23: Blend

- *Filters... → Contour* – Will perform an outline effect (edge detection) on the current picture. Contour has the effect of creating enhanced edges; the issue has been widely discussed on section 2.6. The following pictures show the contour filter applied to an image:

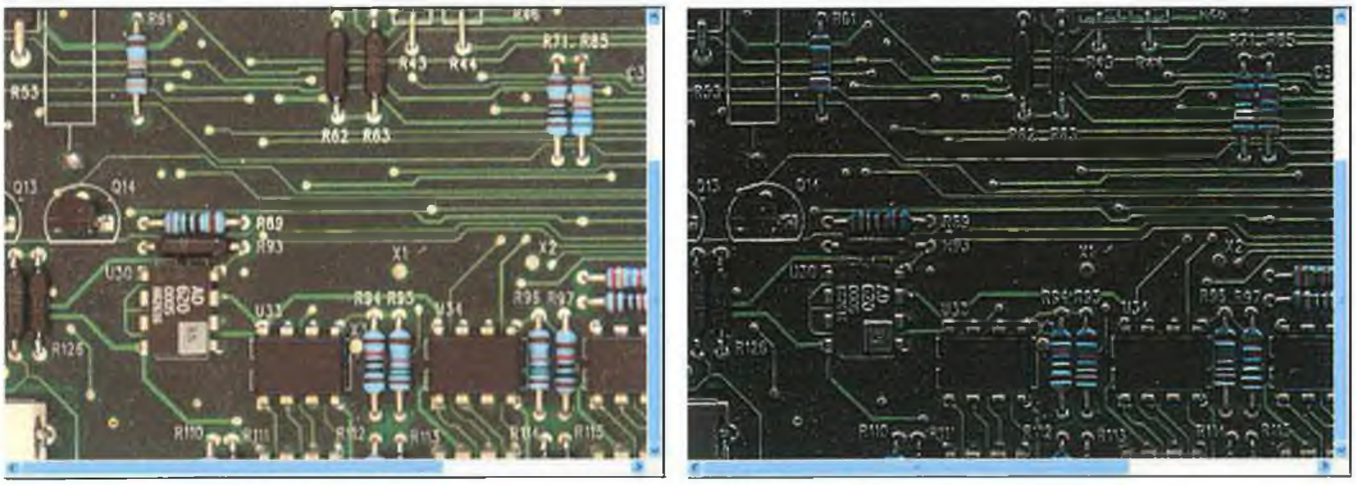
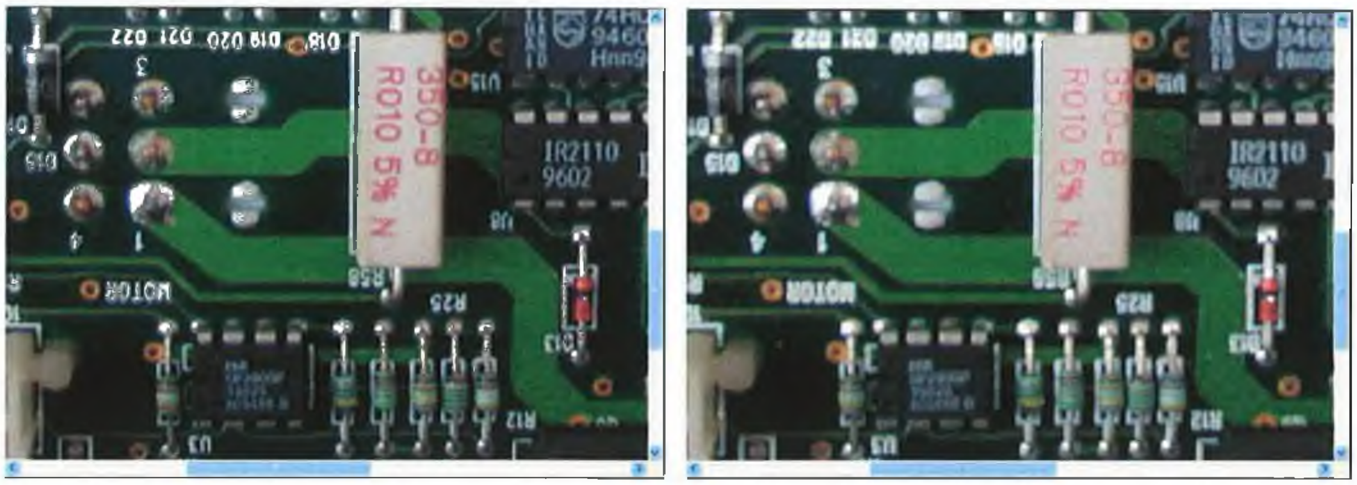

a) Contour – before filter b) Contour – after filter

Figure 5.6.5.24: Contour Filter

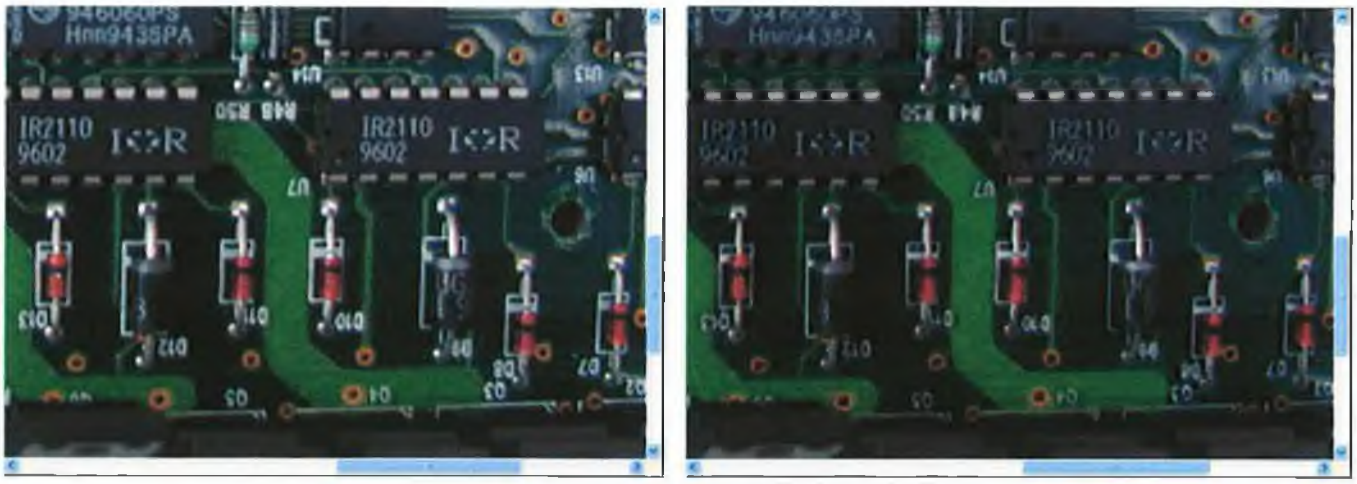
- *Filters... → Dilate (Maximum Filter)* – Will perform a dilation on the current image. The filter diffuses the current image by replacing each pixel with a randomly selected neighboring pixel. Dilation, in general, causes objects to dilate or grow in size; the opposite of this filter is *erosion* which causes objects to shrink. A detailed description of the filter is given in section 3.3. The following pictures show the dilate filter applied to an image:



a) Dilate – before filter b) Dilate – after filter

Figure 5.6.5.25: Dilate Filter

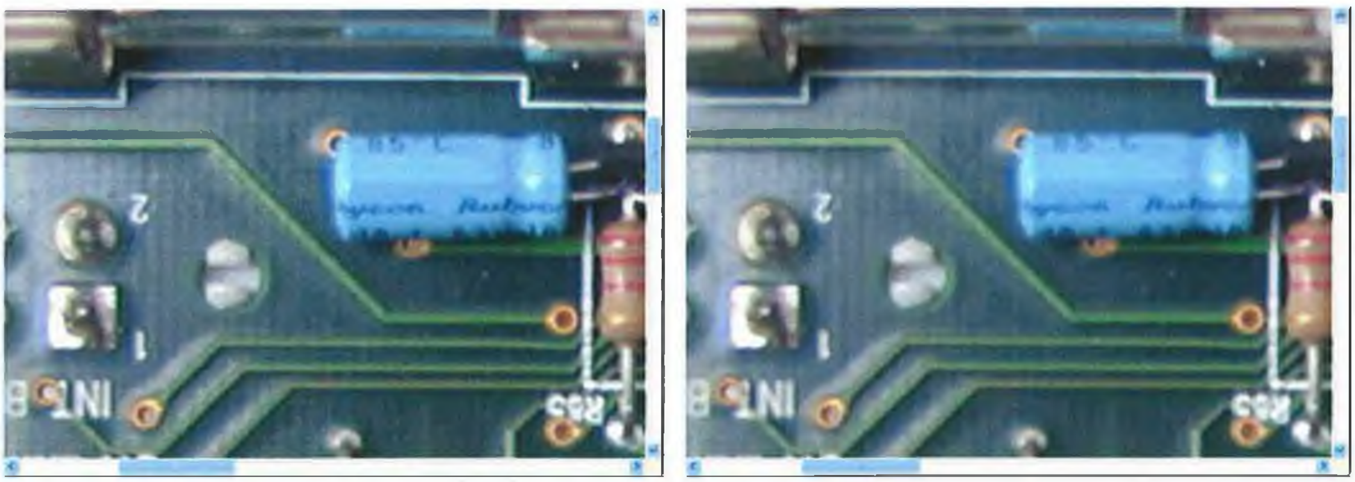
- *Filters... → Erode (Minimum Filter)* – Will perform an erosion on the current image. Erosion, in general, causes objects to shrink in size (see section 3.3); the opposite of this filter is the *dilation*, described above. The following pictures show the erode filter applied to an image:



a) Erode – before filter b) Erode – after filter

Figure 5.6.5.26: Erode Filter

- *Filters... → Median* – Filters the current picture using a 3x3 median filter. Median filtering reduces the noise in an image. The median filter divides the image into pixel groups (of 3 by 3 pixels) computing the output pixel as the median value of the brightness within the group (see section 2.5.2). The following pictures show the median filter applied to an image:



a) Median – before filter b) Median – after filter

Figure 5.6.5.27: Median Filter

- *Filters...* → *Sharpen* – The sharpen filter uses a high – pass convolution matrix filter to accentuate high frequency details in the image The filter can be used to accentuate the edges in an image.

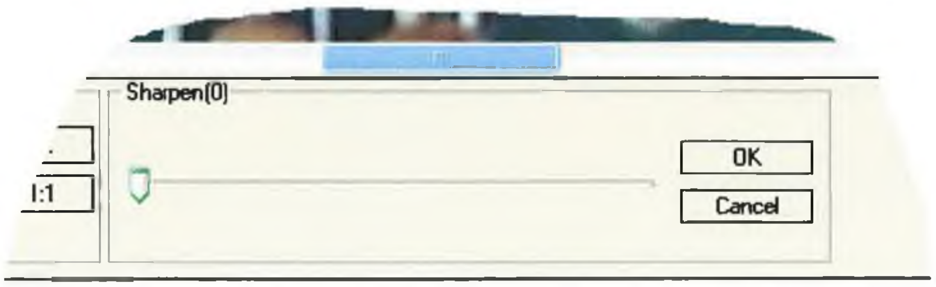
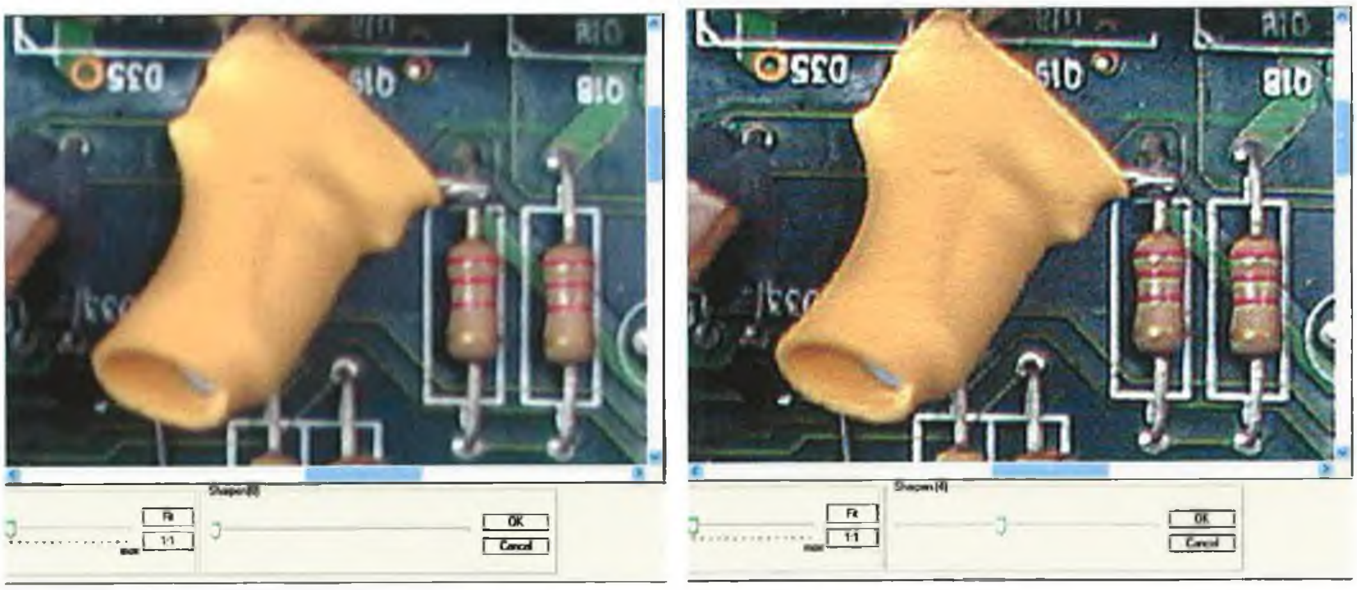


Figure 5.6.5.28: Sharpen

The range of the slider is from 0 to 10, by default the slider is set to 0. The following images show the sharpen filter set at 4 applied to an image:



a) Sharpen – before filter b) Sharpen – after filter (4 units)

Figure 5.6.5.29: Sharpen Filter

- *Filters... → Soften* – The soften filter uses a low – pass convolution matrix filter to accentuate low frequency details in the image. The filter is the opposite of the sharpen filter and is a variation of the smoothing filters described in section 2.5.

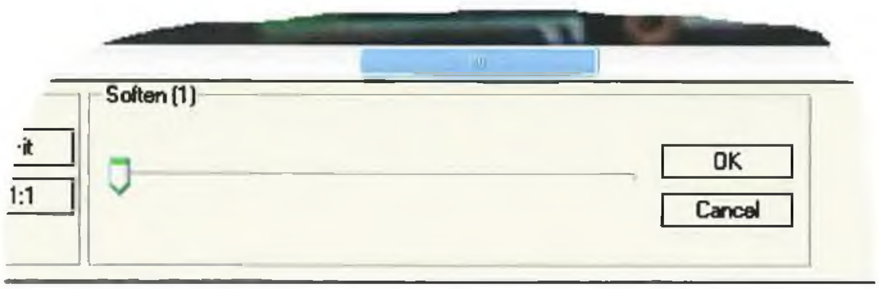
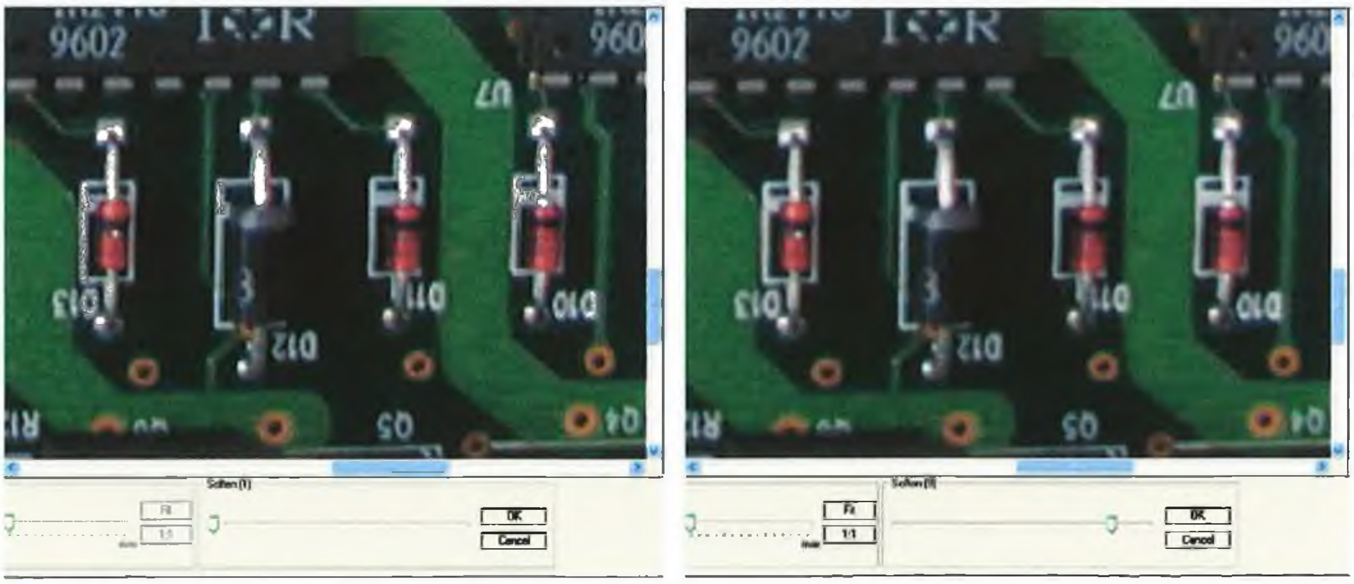


Figure 5.6.5.30: Soften

The range of the slider is from 1 to 10, by default the slider is set to 1. The following images show the sharpen filter set at 9 applied to an image:



a) Soften – before filter b) Soften – after filter (9 units)

Figure 5.6.5.31: Soften Filter

- *Filters...* → *Solarize* – The solarize filter solarizes the colors in the current picture by negating the colors beyond the threshold value. The filter can be used to accentuate the edges of the objects so an edge detection filter will have better results.

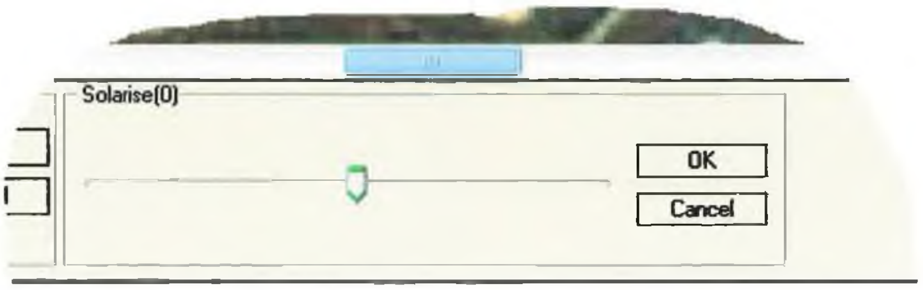


Figure 5.6.5.32: Soften filter frame

The range of the slider (which sets the threshold) is from –128 to 128, by default the slider is set to 0. The following images show the solarize filter set at 30 applied to an image:

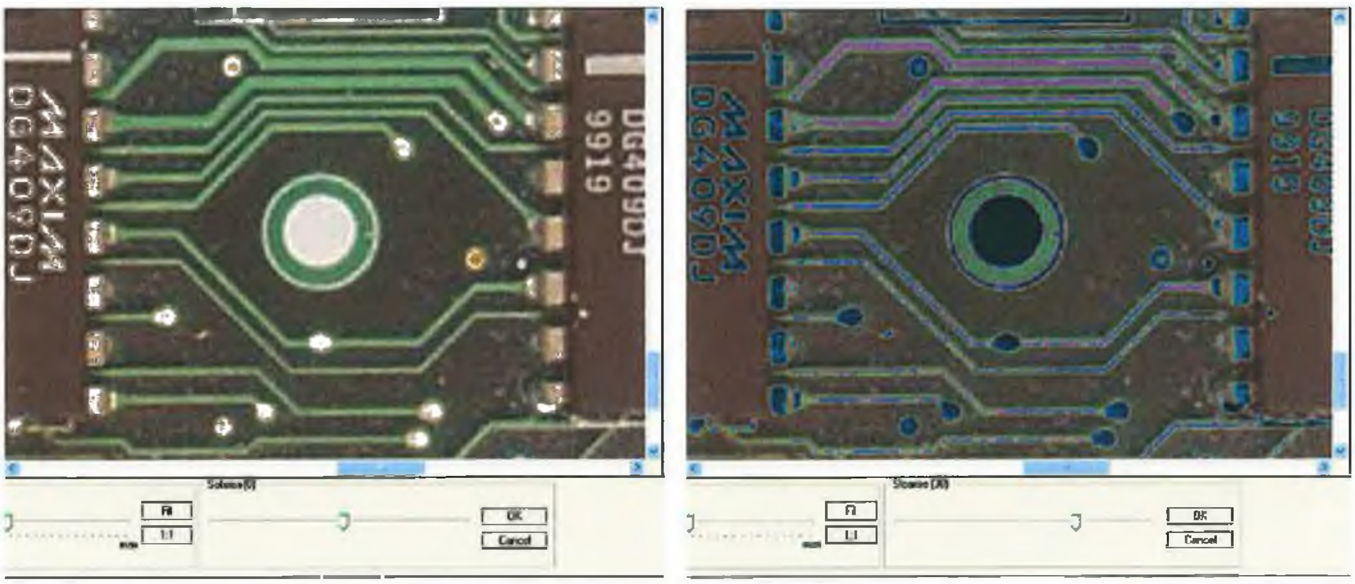

a) Solarize – before filter b) Solarize – after filter (30 units)

Figure 5.6.5.33: Solarize Filter

d) *Image layout filters...*

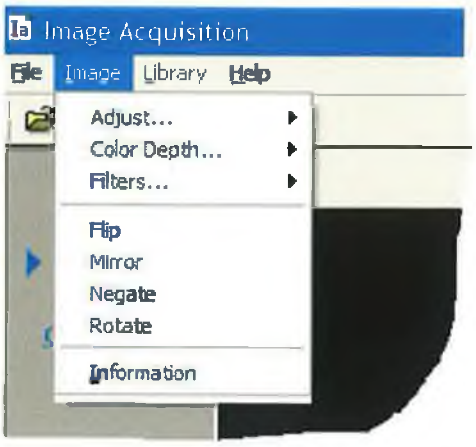
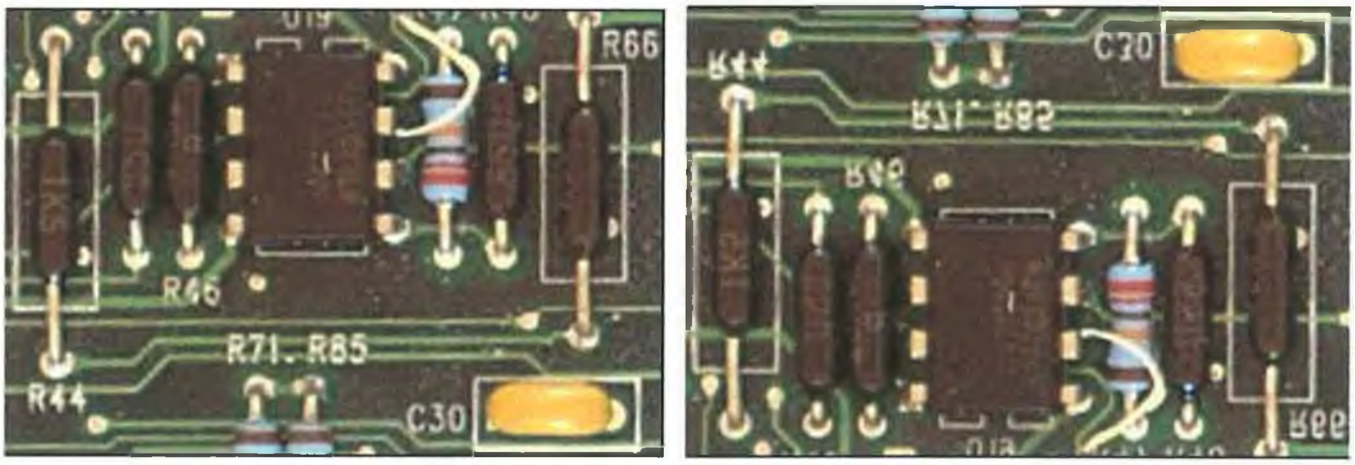


Figure 5.6.5.34: *layout filters* group

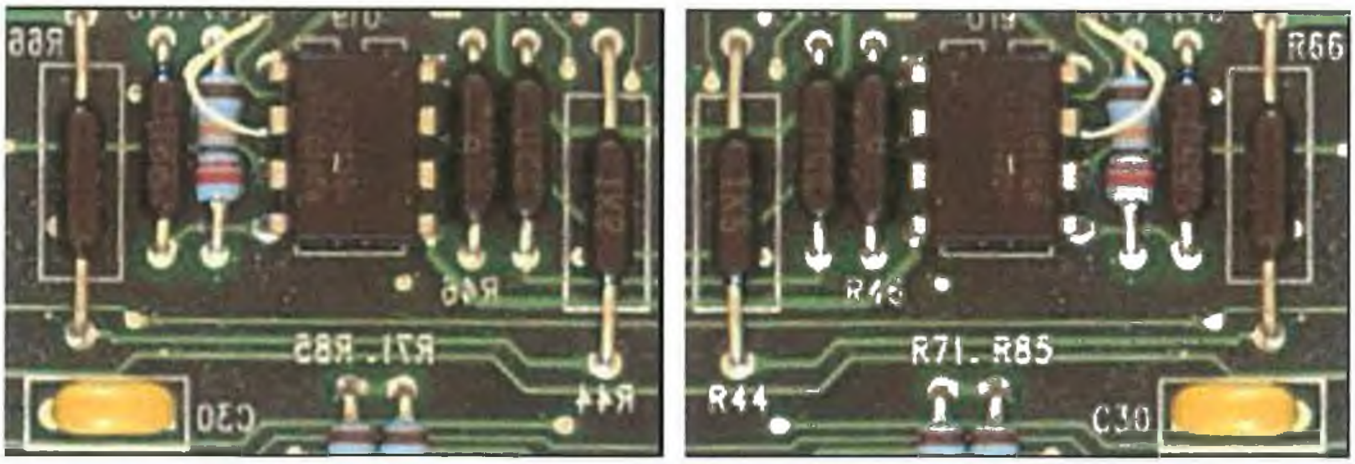
- *Image...* → *Flip* – Will vertically flip the current image. The description of the operation is given in section 2.8.3. The following images show an image before and after it was flipped:



a) Flip – before transformation b) Flip – after transformation

Figure 5.6.5.35: Flip method

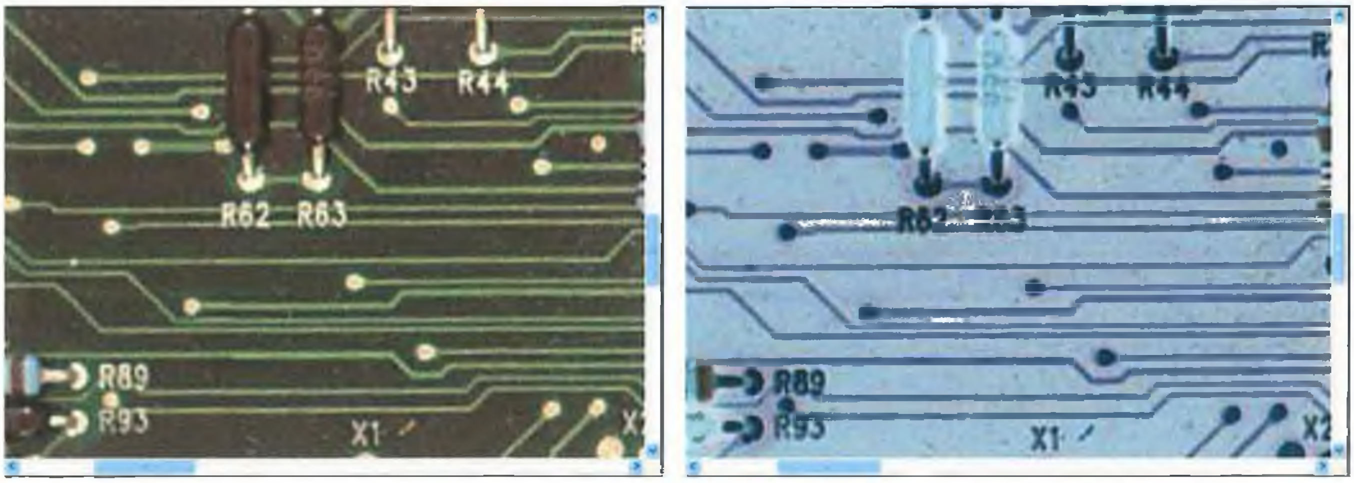
- *Image... → Mirror* – Will mirror the current image (as described in section 2.8.3. The following images show an image before and after it was mirrored:



a) Mirror – before transformation b) Mirror – after transformation

Figure 5.6.5.36: Mirror method

- *Image... → Negate* – Will create a color negative of the image. The operation is very simple: the Boolean operator *NOT* is used (as described in section 2.4.1) for the whole image. The following images show an image before and after it was negated:



a) Negate – before transformation b) Negate – after transformation

Figure 5.6.5.37: Negate method

- *Image... → Rotate* – Will rotate the current image with a certain angle.

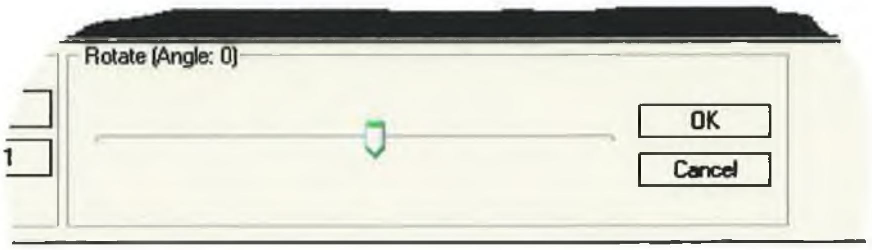
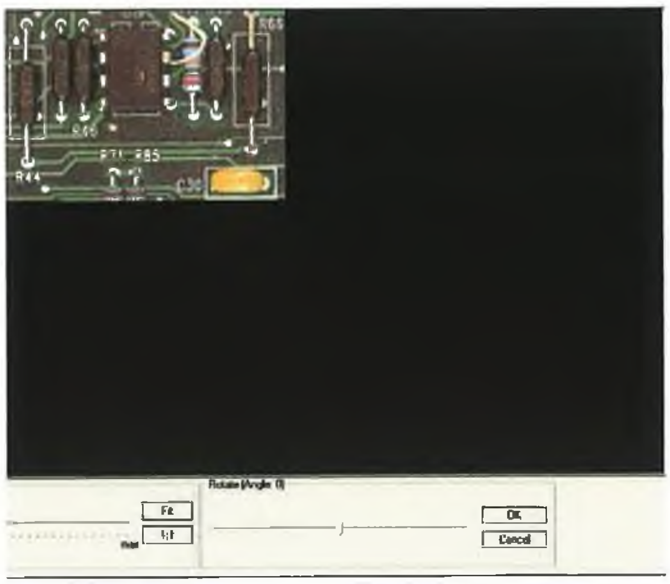
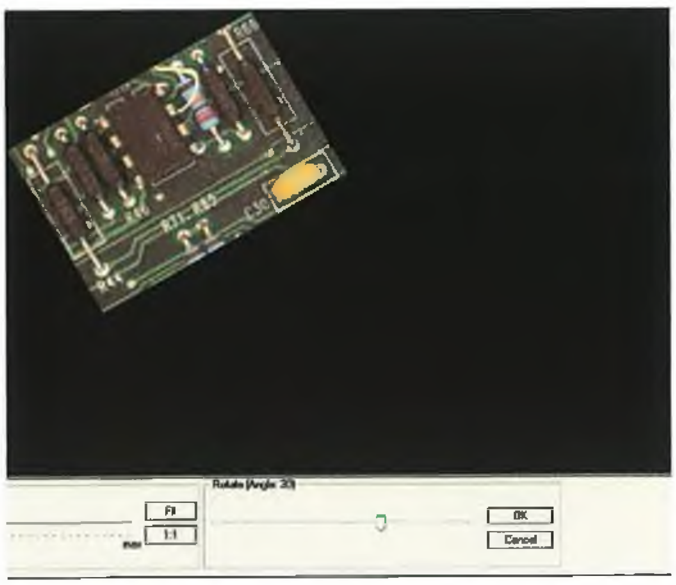


Figure 5.6.5.38: Rotate

The range of the slider is from –180 to 180, by default the slider is set to 0 and the increments are in angles of 1 degree. Positive values of the slider will rotate the image counterclockwise while negative values will rotate it clockwise. The description of the rotation operator is given in section 2.8.1. The following pictures show an image rotated at an angle of 30 degrees:



a) Rotate – before transformation



b) Rotate – after transformation (30 degrees)

Figure 5.6.5.39: Rotate method

e) Other filters

The program is capable to perform some other filters (i.e. blur, diffuse, emboss, mosaic, twist, pinch, swirl), but they have no practical utility for image recognition. These filters were introduced to extend the utility of the program and to enable the user to process other kind pictures (i.e. for artistic purposes), so he doesn't necessary need to install other software capable to process images.

The *Library* submenu contains two items: *Manual Override* and *Delete*.

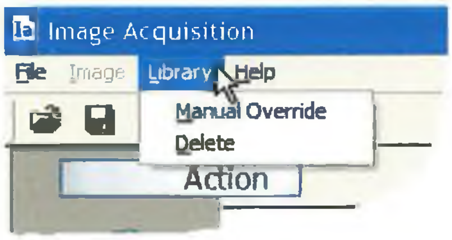


Figure 5.6.5.40: Library menu

Manual Override allows the user to modify the library manually by using an external editor (i.e. Notepad or WordPad) as the database file is in plain text format. The structure of the database will be described in the next section, *View Library*.

Delete allows the user to delete entries from the library or, by choice, to delete the whole library.

The last submenu, *Help*, contains two entries: *Contents...* and *About*.

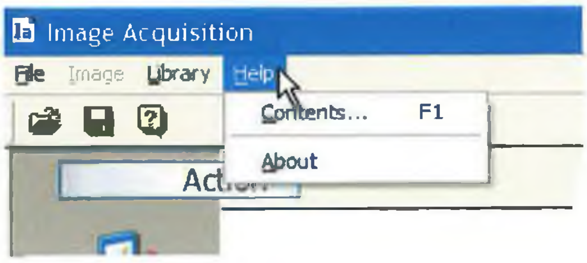


Figure 5.6.5.41: Help menu

The *Contents...* tab will bring the application help window on the screen. The help system of the application is a detailed description of the software, similar with the contents of this chapter, but also has index and search capabilities. The *About* item will bring the application *About box* which usually contains information about the version, the creator of the program or copyright notices. This is a screen shoot of the about window:

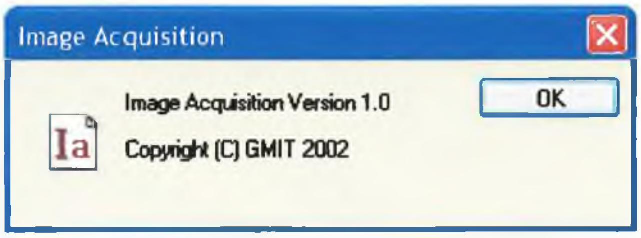


Image Acquisition

Image Acquisition Version 1.0

Copyright (C) GMIT 2002

Figure 5.6.5.42: *About* window

5.6.6 *Tool bar*

The application toolbar is a row, column, or block of on – screen buttons or icons. When clicked, these buttons or icons activate certain functions, or tasks, of the program. *Image Acquisition* toolbar contains buttons which can access tasks like *open, save* or *help* with only one mouse click. The functionality of the buttons on the toolbar can be easily changed in order to meet the user preferences. The following picture shows the toolbar of the application:

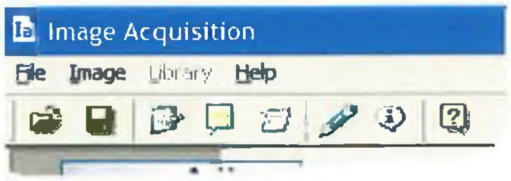


Figure 5.6.6: The toolbar of the application

The meaning of each icon from left to right is:

1. Open – performs the same action like **File** → **Open** menu
2. Save – performs the same action like **File** → **Save** menu
3. Switch the program in *Select Components* mode
4. Switch the program in *View Library* mode
5. Switch the program in *Analyze* mode
6. Library – manual override (same as **Library** → **Manual Override** menu)
7. Image information (same action like **Image** → **Information** menu)
8. Help (same as **Help** → **Contents** menu)

5.6.7 Status bar

The status bar is the area from the bottom of the screen where is displayed information about the actions performed by the program.

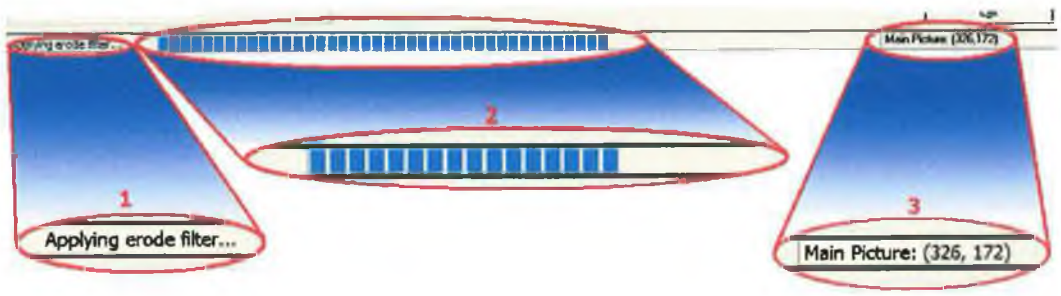


Figure 5.6.7: The *status bar*

The status bar is split in three parts:

i) The name of the action performed

ii) The progress of the current action

iii) The position of the mouse

The picture above shows the status bar where each of the three parts is magnified for a better view.

The first and second part of the status bar are updated each time an action is performed in the program, i.e. when loading or saving a file, a filter or a transformation is applied, a selection is made.

The third part is only updated when the mouse pointer moves over the main picture or over the selection. The text is changed accordingly (*Main Picture* or *Selection* followed by the position of the mouse). The precision of the numbers shown is one pixel on each x and y axis. A certain pixel will have the same position regardless of the zoom factor.

5.6.8 Control Panel

The control panel is the area of the screen where the user can choose which mode the program is running; *Select Components*, *View Library* and *Analyze*. By clicking on the associated icon the mode is changed.

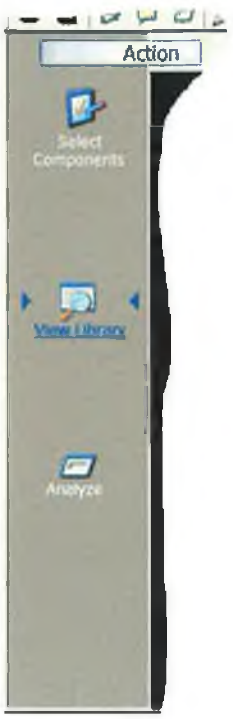


Figure 5.6.8: The *control panel* of the application

The selected mode is highlighted by two blue arrows and an underlined text (*view library* on figure 5.6.8).

5.6.9 Zoom panel

The zoom panel allows the user to magnify or shrink pictures. Changing the magnification factor will not affect in any way the picture, just the mode it is displayed. The zoom issue has been widely discussed on section 2.8.2.

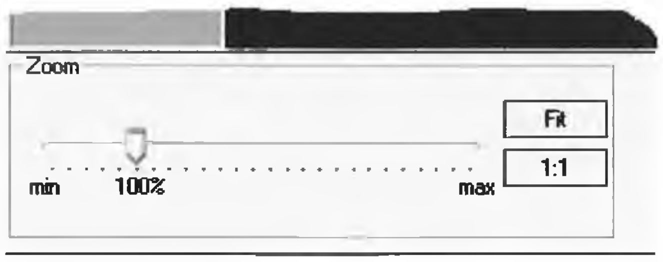


Figure 5.6.9: *Zoom panel*

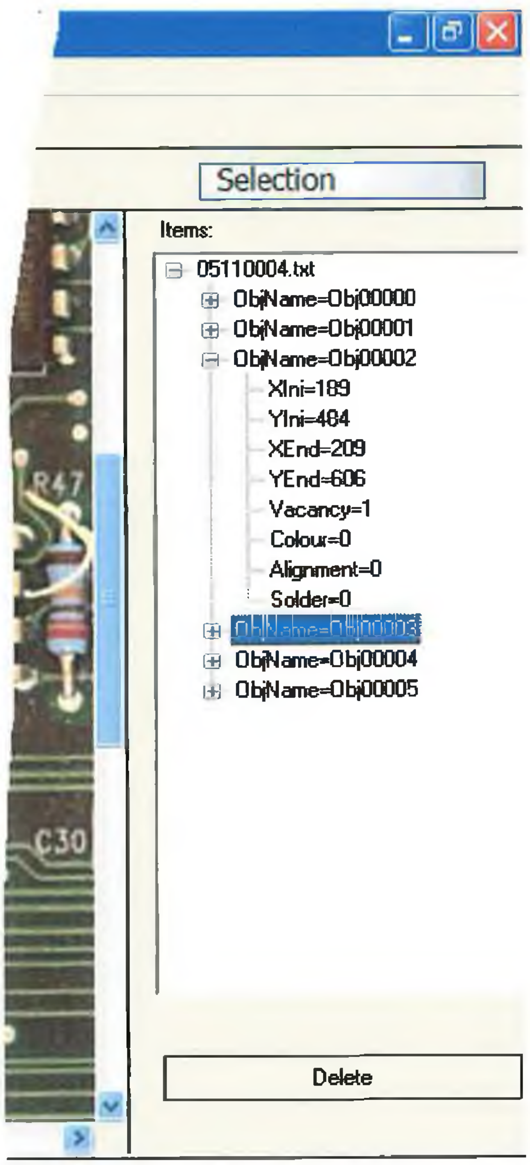
The ***min*** value of the slider will display the picture at about one quarter of its original size. The ***max*** value of the slider will show the image magnified about seven times. When the slider is set to ***100%*** (the default value) the image is displayed on its original size.

The ***Fit*** button will adjust the zoom factor so that the image fits into the image holder (both width and height); the slider will automatically position accordingly. The ***1:1*** button will reset the zoom factor to 1 (no change in size); the slider will automatically be set to the ***100%*** position.

5.7 *View Library*

View Library

When application is running in *View Library* mode, the user can visualize, modify or delete components from the library. The application main window is the same as in *Select components* mode, the difference is the right panel, which is different. The user cannot add more components to the library or perform filters on the main picture when the program runs in *View Library* mode. Figure 5.7.1 shows the right panel when the application is running in *View Library* mode:



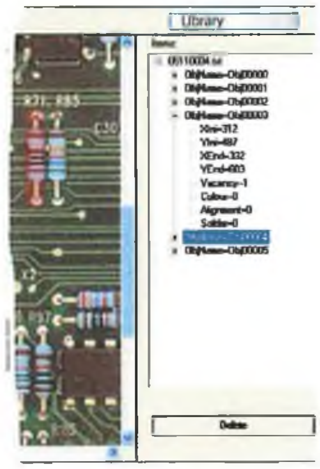
```
⊟ 05110004.txt
    ⊞ ObjName=Obj00000
    ⊞ ObjName=Obj00001
    ⊟ ObjName=Obj00002
        XIni=189
        YIni=484
        XEnd=209
        YEnd=606
        Vacancy=1
        Colour=0
        Alignment=0
        Solder=0
    ⊞ ObjName=Obj00003
    ⊞ ObjName=Obj00004
    ⊞ ObjName=Obj00005
```

Figure 5.7.1: Application in *View library* mode

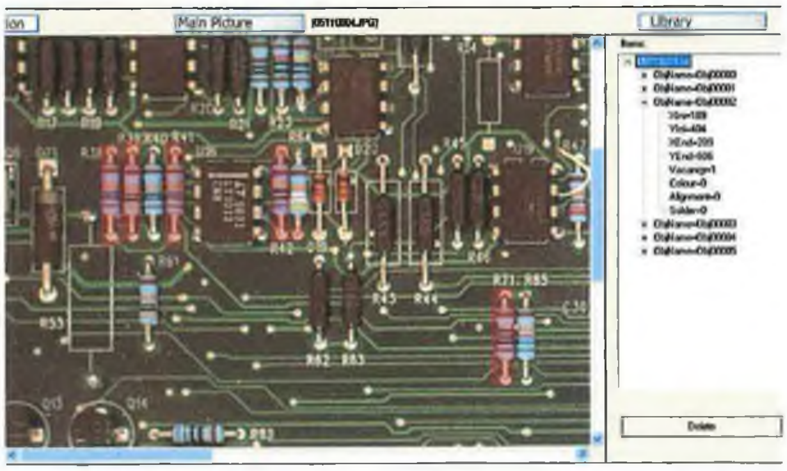
The *tree control** [MSDN, 2002] presented on figure 5.7.1 contains all the objects in the library. The tree is structured on three levels:

- The first level (the *root* level) has only one object: the name of the library.
- The second level (the *object* level) is a *child* of the first level and contains all the selections made by the user. Although it can contain maximum one hundred thousands elements it is expected that a total of no more than one or two hundreds of objects will be defined by the user. Each field has the name of the object as defined earlier in this chapter (section 5.6.3).
- The third level (the *detail* level) is a *child* of the second level and contains details (like position and the defects that will be analyzed) regarding each object in the library. A defect that is equal to zero means "the defect will not be checked at analysis time", while a value of one means that it will be checked.

When the user clicks on a second or third level object, that object will be highlighted on the main picture (figure 5.7.2 – a). If the user clicks on the root all the objects in the library will be highlighted (figure 5.7.2 – b).



a)



b)

Figure 5.7.2: a) One component selected; b) all components selected

* The name *tree control* and its proprieties (*root* and *children*) are used and implemented in document as they have been defined in Microsoft Developer Network (MSDN), Visual C++/Controls section.

A component will be removed from the library by pressing the ***Delete*** button when a second of third level object is selected. If the root is selected the whole library will be deleted. In this case a confirmation dialog will be displayed.

Beside the possibility to edit the library within the program (as described above), it is also possible to edit the library using an external text editor (such as Notepad, WordPad, Word, etc.) because the library is saved in text format. In order to do that the user has to use the command *Manual Override* from the main menu (as described in section 5.6.5). However, caution must be taken as, if invalid values are introduced in the fields, the results may be undesirable. The listing (fragment) below describes the structure of the library.

- *File 05110004.TXT:*

```
[Header]
Image=05110004.TIF
Path=D:\Test Pictures\

[Objects]
ObjName=Obj00000
XIni=114
YIni=486
XEnd=136
YEnd=606
Vacancy=1
Color=0
Alignment=1
Solder=0

ObjName=Obj00001
XIni=139
YIni=482
...
```

The library has the same name (on the hard disk) with the picture associated with it, only the extension is changed to "*.TXT*". The library starts with a header which makes the link between the library and the *main picture.* The header contains the name of the file and the path where it can be found. Then, it continues with an enumeration of all the objects, their position in the main picture and the types of defects that have to be analyzed.

5.8 *Analyze*

The program running in *Analyze* mode enables the user to analyze *target pictures* based on the libraries previously built. This time the main picture holder (section 5.6.1) is empty, as the *target picture* has to be loaded in its place. Figure 5.8.1 shows the right panel when the application is running in *Analyze* mode:

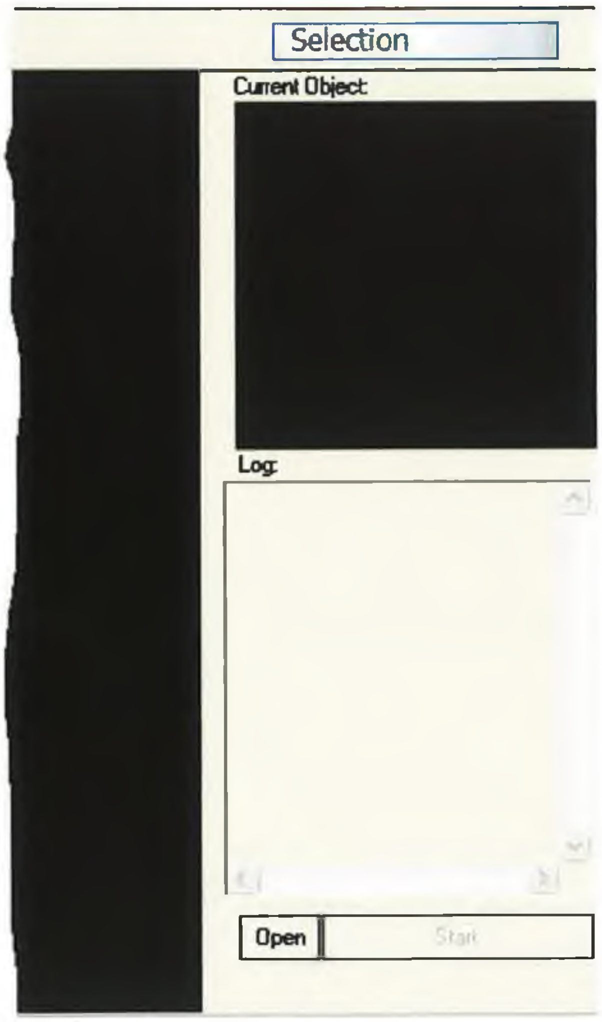


Figure 5.8.1: Application running in *Analyze* mode

The meaning of the interface is:

- ***Open*** button – Is used by the user to open a *target picture*. The procedure is identical to the one used to open the *main picture*. This time the picture is loaded in the *main picture holder* (section 5.6.1). Once the picture is loaded, the ***Start*** button becomes enabled and the program is ready to initiate the processing.
- ***Start*** button – when pressed, the application will initiate the processing of the *target picture*. In the same time it will change its label to ***Stop***. This allows the user to interrupt the analysis at any time.
- ***Current object*** – will display during the analysis the picture of the component that is processed.
- ***Log*** – lists the progress of the analysis, including all the objects that have been analyzed, the results and statistics. At the end of the process this list will be saved on disk in text format. The resulting file will have the same name with the *target picture* plus the characters "*–log*" and the extension "*.txt*".

The following listing is a sample of a log file generated by image acquisition:

```
Image Acquisition Log for file 05110004.JPG
Log started on 27/5/2002 at 11:13:38

1. Obj00000
Coords: (253,74,272,94)
Looking for: Vacancy, Alignment defects
Vacancy Trust Factor: 96.15%
Alignment Trust Factor: 88.78%

2. Obj00001
Coords: (148,75,167,98)
Looking for: Vacancy defects
Vacancy Trust Factor: 98.67%

3. Obj00002

.
.
.

13 objects has been analyzed
All objects in the library has been successfully
processed
Log ended at 11:13:41
```

From the listing have been excluded 11 objects (the dots) as they follow the same pattern with the first ones. The log states in the header the name of the *target picture* (in this case “05110004.JPG”), the date and time when the processing started. The body of the log file enumerates all the objects that have been analyzed and returns a *trust factor* for each of the defects that have been investigated. The trust factor is expressed in percents from 0 to 100. A trust factor of 0% means that the component is certain to have the defect, while a trust factor of 100% denotes that the component is certain not to have the defect. Overall, the following intervals can be considered for the trust factor:

• 0% – 40%:	the component has the defect
• 40% – 60%:	uncertain
• 60% – 100%:	the component is defect free

The log file ends with some statistics about the components analyzed:

- An overall counting of the objects analyzed;
- If necessary, will state how many components haven’t been analyzed, otherwise (like the case listed above) will affirm that all the objects have been successfully processed.
- The cause why the log file ended: all objects analyzed; interrupted by user (by pressing ***Stop*** button); or the program has been aborted (external cause).
- The time when the log file ended.

In case the user decides to run a second analysis on the same target picture, the old log file is not overwritten, but appended.

5.9 *Guidelines on improving the recognition accuracy*

This section will present some examples of how the components should be selected in order to maximize the accuracy of the recognition process. First of all, as a rule, a selection shouldn't contain more than a component. Next figure will show some examples of selections:

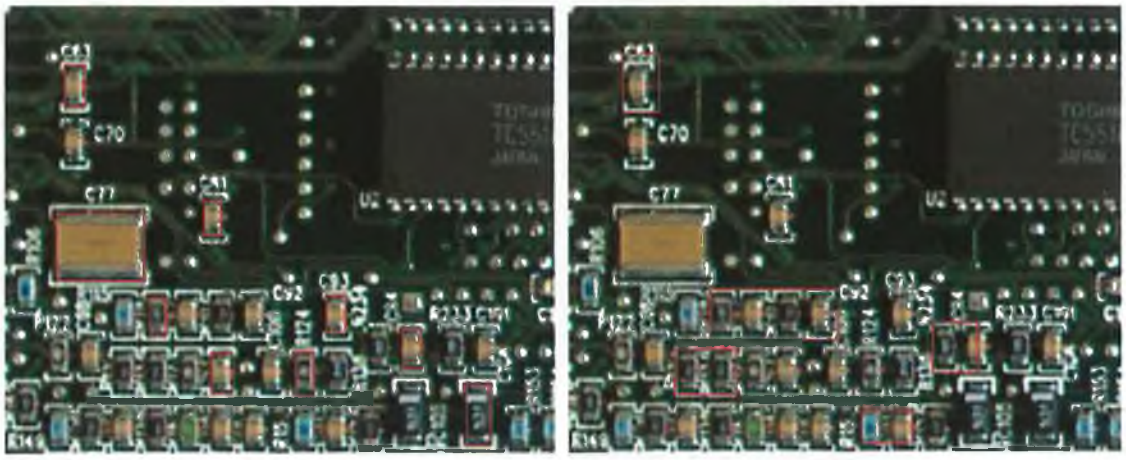
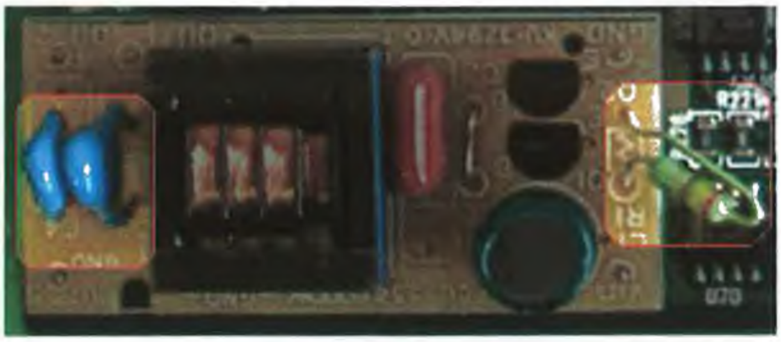

a) Valid selections b) Invalid selections

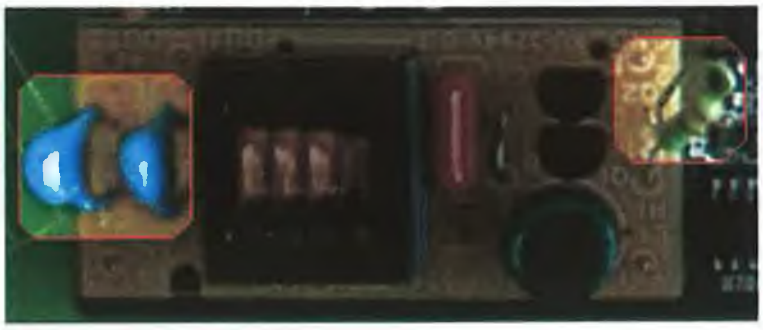
Figure 5.9.1: Selections

The red rectangle around the components in figure 5.9.1 defines a selection. In figure 5.9.1a are presented components that are selected correctly. It can be seen that the rectangle is positioned as precise as possible around the object. Figure 5.9.1b presents some wrong examples of components selection. Some of the rectangles are not properly fitted to match the size of the components, this way much of the noise contained in the background is added and recognition may be compromised. Some other cases, a group of components are selected, which can lead to unpredictable results as each object has to be treated as an individual. Using picture magnification, as described in section 5.6.9, the selection process can be much easier and accurate. Again, it has to be mentioned, that even if the selection of the components phase may look slow or too thorough, this step has to be performed only once for a certain model of board.

There is another class of components that have to be excluded from the recognition process: the components that are not embedded on the board (i.e. bounded with flexible wires). Two examples are presented in figure 5.9.2:



a) First board



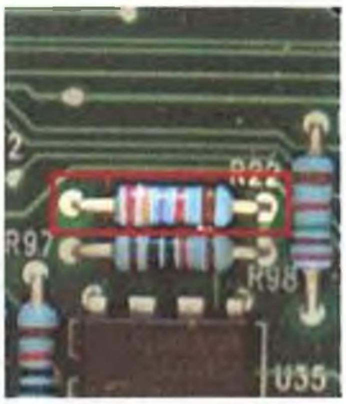
b) Second board

Figure 5.9.2: "Flexible" components

As it can be seen from the figures above, on two different boards, the components change not only position, but also the shape (due to perspective distortions). In this case, it is not possible to include them in the recognition process, as the camera and the boards are always in the same position. Next section will describe each type of defect in detail.

5.9.1 Vacancy

This type of defect checking verifies if a component is present or not on the board. Next figures show a classic example and some of the conclusions after the analysis using two filters.


a) Valid component

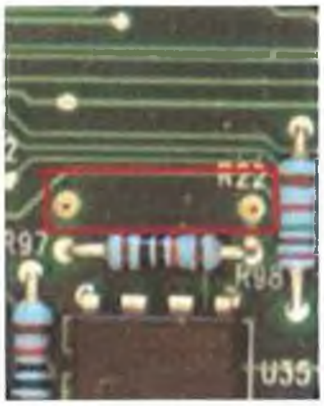
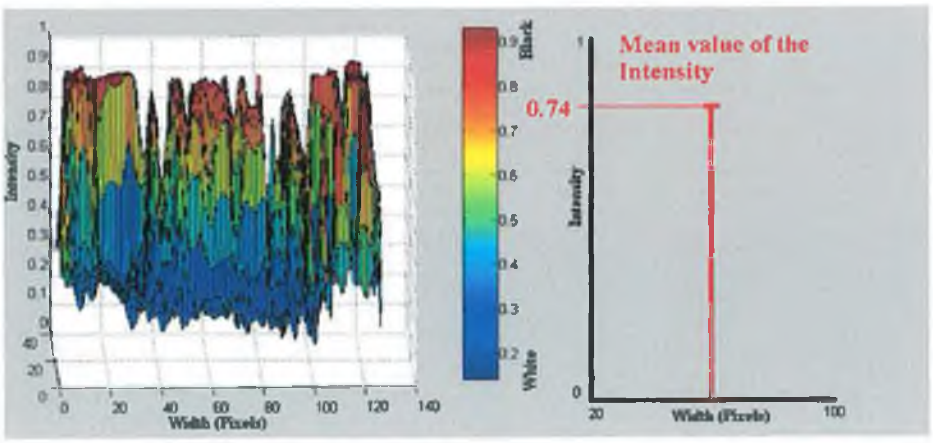
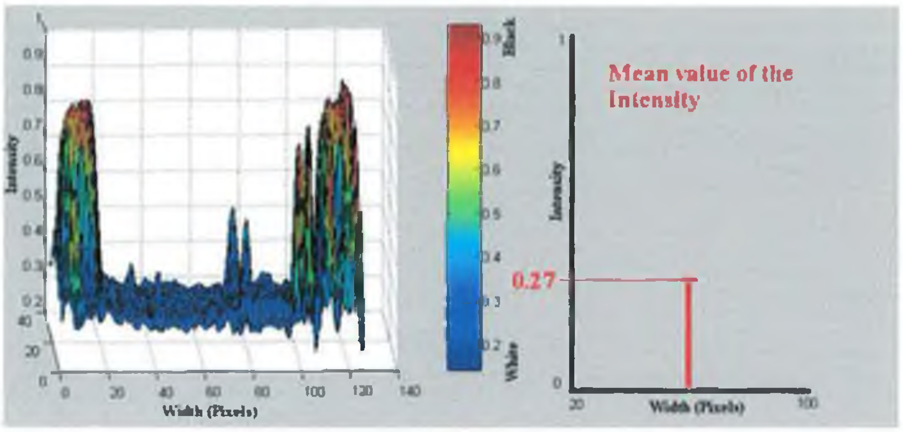

b) Missing component

Figure 5.9.1.1 Vacancy

As one might observe, although the rectangle is quite tight around the resistor, not much attention have been paid to the pins that bounds the component on the board. As the analysis will show it doesn't matter so much in this case if the user selects or not the pins. Figure 5.9.1.2 shows a light intensity comparison between the two cases:



a) Valid Component



b) Missing component

Figure 5.9.1.2: Intensity comparison

As the graph demonstrates, there is a noticeable difference (0.47 on a scale from 0 to 1) between the light intensity levels in the case studied. Figure 5.9.1.3 presents the same case, this time studied with a normalized cross – correlation filter (as described in section 2.7).

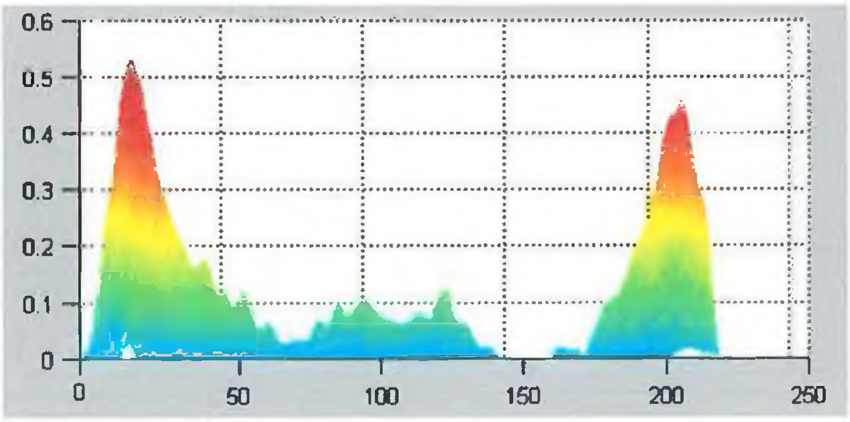


Figure 5.9.1.3: Normalized cross – correlation filter missing component comparison

The graph in figure above is related with a vacant component. The two local maximums correspond with the two circular areas made of solder, where the component was supposed to be planted. If the component was present the maximum should be in the middle of the graph, as shown in figure 5.9.1.4.

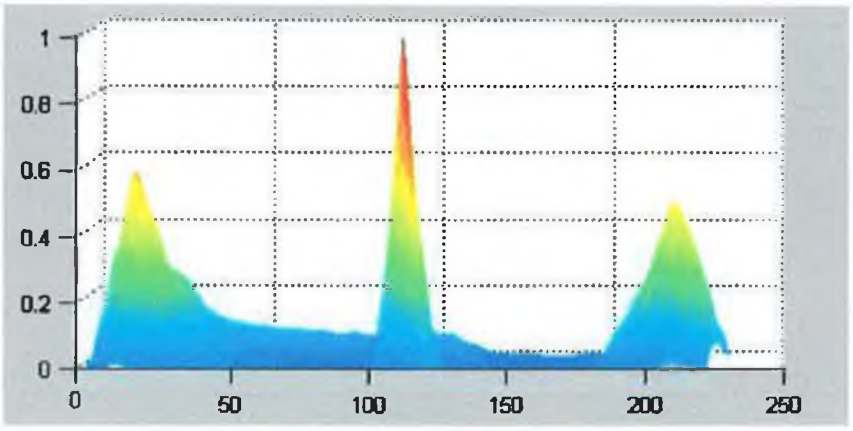


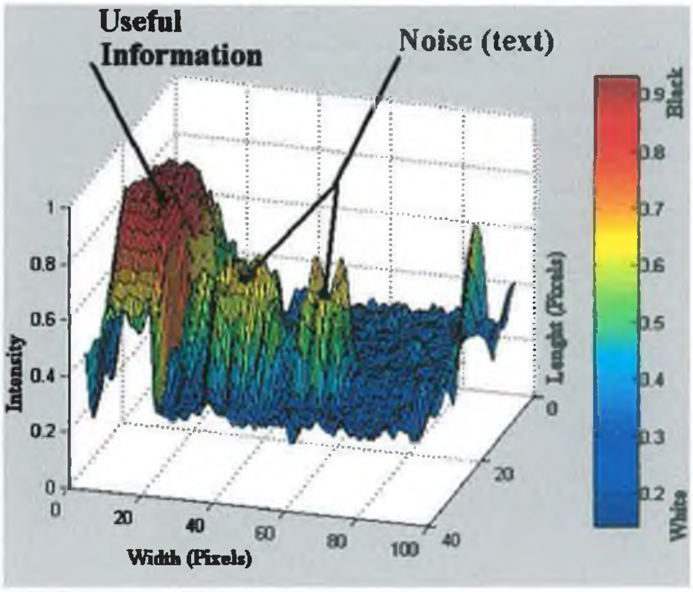
Figure 5.9.1.4: Normalized cross – correlation filter
valid component

5.9.2 Alignment

The alignment type of defects verifies if a component is correctly aligned on the board. From a physicist point of view, for certain electronic components alignment is crucial (i.e. capacitors), but for other components alignment is not important (i.e. resistors). However, the problem is somehow simplified if we think of how the components are placed on the board. The machine can only place the components in the correct way, upside down, or fail to insert the component (case studied on the previous section). Due to the construction of the robots that place the components, it is impossible to place them, for instance, with a deviation of 45 degrees from the original position. The following figures present an example of misaligned capacitor:

a) Sample capacitor

b) Misaligned capacitor



c) Intensity analysis

Figure 5.9.2: Misaligned components

The graph shows the alignment of the capacitor can be predicted by the analysis on the light intensity levels. The white strip on the capacitor (that marks the polarity) has values of intensity close to one (maximum) while the noise (the text printed on the capacitor) has its maximum at 0.5.

5.9.3 Solder Defects

The solder defects analyze those components that are welded incorrectly or the welding may cause a short – circuit. Some examples of selections will be presented together with the processing that is performed.

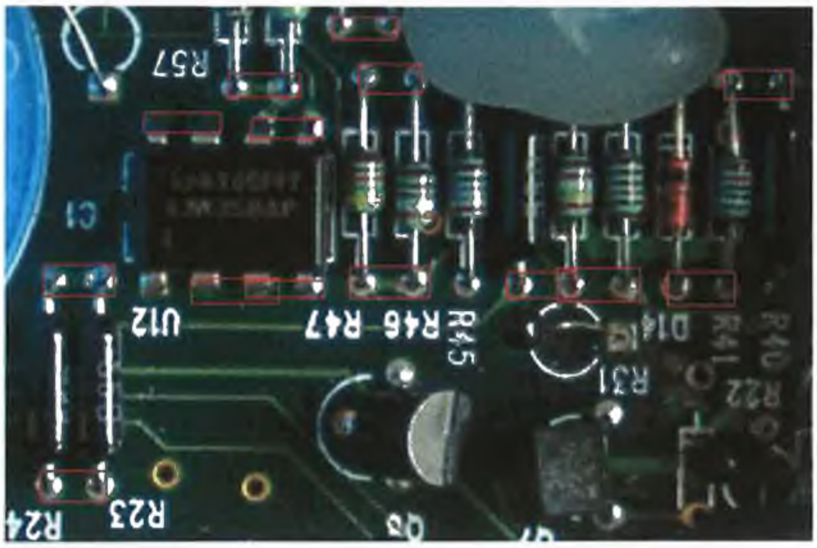
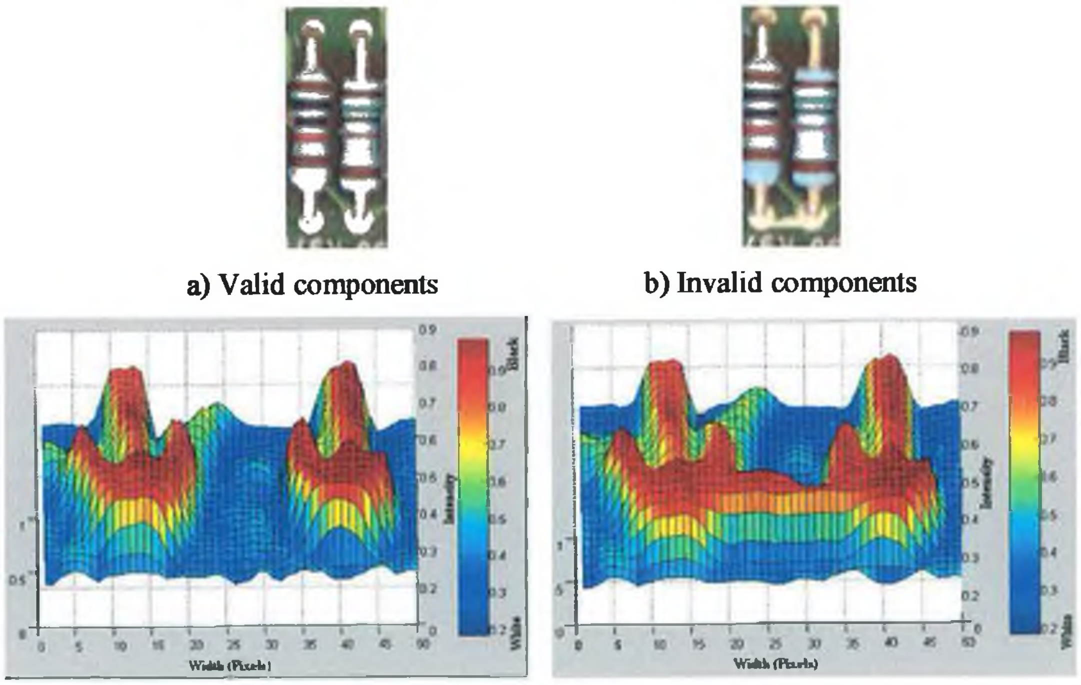


Figure 5.9.3.1: Solder defects selection

As it can be noticed, the selections aim for the end of the components, where the pins are welded on the board. Overlays are allowed, as each selection is treated as an individual, there is no link with the previous one. Next figures present the processing in intensity domain.



a) Valid components b) Invalid components

c) Valid components diagram d) Invalid components diagram

Figure 5.9.3.2: Solder defects

The correct join has a light intensity of 0.2 between the pins (on a band of about 15 pixels width), while the pins that are in short – circuit the light intensity is close to the maximum. A big advantage is the fact that the pins are metallic, thus the light is strongly reflected, making the contrast stronger and, as a result, the recognition process more accurate.

5.9.4 Color Code

The color code analysis studies those objects that are color – coded and try to confirm if the right component is used. Although, the number of rings on a resistor can be counted, establishing exactly what color is used is quite difficult as the components are very small; a ring on a resistor is only 3 – 4 pixels width. A magnified resistor is shown in figure 5.9.4 together with the analysis in HSV (*Hue, Saturation, Value*) spectrum.

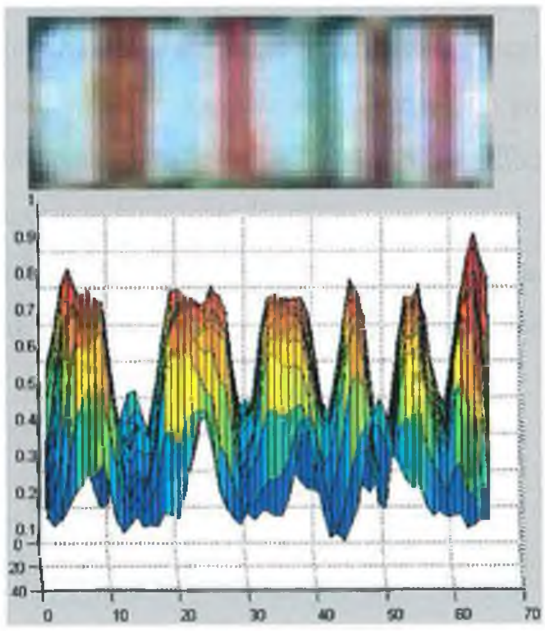


Figure 5.9.4: Color code on a resistor

6 Conclusions and Further Developments

6.1 Thesis summary

The aim of the research was to develop a semi – automatic, self – learning pattern recognition system capable of detecting defects such as component vacancy, component misalignment, component orientation, component error, and component weld. The research was conducted in three directions: image acquisition, image filtering/recognition and software development.

Image acquisition studied the process of forming and digitizing images and some fundamental aspects regarding the human visual perception. Analogies have been made between the human eye system and how images are formed on cameras. Although not strictly correct, the analogy between machine vision and human vision is currently the best model available for researchers. Then, the attention has focused on image processing systems and the way its components interact. The importance of choosing the right camera and illumination system for a certain type of problem has been highlighted.

Probably the most important step towards image recognition is image filtering. The filters are used to correct and enhance images in order to prepare them for recognition. Also, a combination of certain filters might be regarded as object recognition (i.e. the application presented in section 3.12). That is why a lot of attention has been paid to image filtering. The filters can be divided in several categories starting with the fundamental process of the formation of analogue and digital images when convolution is employed to correct defects that may appear in an optical system. The histogram operations are important when one wishes to compare two images acquired in different circumstances; the operations based on mathematics provide a fast and easy way to manipulate images; smoothing filters are important as they provide a convenient way to reduce the noise in images; the derivative based operations are the background for more advanced filters such as edge detection; the geometric operations allow the pictures to be rotated, scaled or translated in order to apply other filters or are simply an aid on having a better view on the image; the cross – correlation based algorithms are successfully used to detect patterns on a search window (template matching). The mathematical morphology gives highly optimized methods of processing and analyzing shapes in binary images. The morphological processing of the grey scale or binary images became in the recent time a powerful instrument for the applications that use segmentation, image compression or image recognition.

The third direction in which the research was conducted was software development. First of all, a number of possible hardware platforms have been investigated. A personal computer has proven to be the most suitable platform for software development due to its relatively low price compared to other solutions, but also due to the advances in terms of computing power, which bring the desktop computers that much closer to a workstation or a server. As an operating system Windows from Microsoft was chosen for its popularity among personal computer users but, most important, for the wide range of applications developed for this environment and the software support for most hardware devices which, for instance, makes it easier to choose a certain camera or scanner. All was left to do was to choose a development environment in order to build the application. C++ by far looked the most attractive solution due to it's capability to develop any application in a very professional way (a very fast code compared to others compilers). Matlab came as help, because it includes an image acquisition toolbox with a large variety of functions dedicated to image processing and also provide a very good integration with C++ language through its compiler toolbox. At the end the software application was built using C++ for the user interface and Matlab for filter design and implementation. The images were manipulated using both of the compilers.

Next sections will present further developments to the project and the final conclusions of the thesis.

6.2 Further Developments

The research carried out has a great scope for future developments. A number of issues have been recognized during the development phase:

- The illumination system can be improved so that the light emitted by the bulbs is controlled by the computer. This will eliminate the need to correct contrast or brightness by filters.
- Using a conveyor belt to transport the boards to the illumination system will increase the speed of the image acquisition process.
- A higher resolution camera can make the recognition of small components more reliable. Also, a digital camera that supports TWAIN cross – platform interface will speed the acquisition process. In this way, the picture will be imported straight into the application (TWAIN interface is already implemented in the program), eliminating the need to save first the picture on the disk and then to import it into the program.
- Using neural networks to learn patterns of most common components will improve the process of selection when the program runs in *select components* mode. The libraries will be able to exchange information so, when a new library is created, the program will automatically select the components that exist in other libraries.
- The functionality of the program can be extended so a larger number of defects may be analyzed and detected. For instance, using OCR (optical character recognition) processing, the text printed on the components can be analyzed and, this way, it can be established exactly if the correct component is used or not.

6.3 Final Remarks

The importance of computer image processing is continuously growing in a world were automation is replacing more and more human activities. Part of the fields where image processing was successfully implemented has been discussed at the beginning of this thesis. A large number of algorithms (some very simple yet ingenious, some very complex requiring years of research) have been developed in order to achieve these goals. Although some of the algorithms are common to most applications, they are not classified on a scientific base, meaning that there isn't a set of rules to tell which one to use in a given situation.

The application developed during the research is customised in order to meet the requirements of the industrial partner, Tyco Healthcare/Mallinckrodt, and incorporates the most common algorithms used in image processing. The application is a reliable tool for defects checking of printed circuit boards. It provides a fully integrated framework able to analyze components by a series of filters such as noise reduction, edge detection, geometrical operations or pattern matching. In the same framework, databases of components are built and the results of the analysis are stored on disk, in log files. Furthermore, the ability of the system to control the image acquisition process increases the functionality of the program.

References

- Ballard, D. and Brown, C. – *Computer Vision*, Prentice-Hall, 1982

- Batchelor, B.G., Hodgson D.C. - *Automated Visual Inspection*, IFS (Publications) Ltd., UK, 1985

- Blahut, R. E. – *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, Reading, 1985

- Brigham, E.O. – *The Fast Fourier Transform and Its Applications*, Prentice Hall International Editions, Englewood Cliffs, NJ, 1988

- Castleman, K.R., - *Digital Image Processing*, Second ed. Englewood Cliffs, New Jersey: Prentice-Hall., 1996

- Chapman, A. - *The Pit and the Pendulum: George Biddell Airy and His Measurements of Gravity*, Astronomy Now 5, p. 18-20, 1991

- Cho S., Haralick R. Yi S. – *Improvement of Kitter and Illingworth's Minimum Error Thresholding*, Pattern Recognition, p. 609-617, 1989

- Davies, E – *Machine Vision: Theory, Algorithms and Practicalities*, Academic Press, p. 149 – 161, 1990.

- Duda R. O., Hart P. E., *Pattern Classification and Scene Analysis*, New York: Wiley, 1973.

- Faugeras, Olivier - *Three-Dimensional Computer Vision: A Geometric Viewpoint,* M.I.T. Press, 1993.

- Frisby, J.P. - *Seeing: Illusion, Brain and Mind*, Oxford University Press. Walton Street, Oxford, 1979.

- Giardina C.R., Dougherty E.R. – *Morphological Methods in Image and Signal Processing*, Prentice Hall, Englewood Hills, NJ, 1988

- Gonzalez, R.C. and R.E. Woods, - *Digital Image Processing,* Reading, Massachusetts, Addison-Wesley, 1992.

- Goodman, J.W. - *Introduction to Fourier Optics,* McGraw-Hill Physical and Quantum Electronics Series, New York: McGraw-Hill, 1968

- Gui, V., Lacrama D. and Pescaru D. - *Image processing*, Timisoara, "Politehnica" Press, ISDN: 973-9389-26-0, 1999

- Haralick, Robert M., and Linda G. Shapiro - *Computer and Robot Vision,* Volume I, Addison-Wesley, 1992.

- Horn, Berthold, Klaus Paul - *Robot Vision,* MIT Press, 1986.

- http://iacl.ece.jhu.edu/projects/ - *The Image Analysis and Communications Lab – Active Contours*

- http://sun16.cecs.missouri.edu/ – *Computational Intelligence Research Laboratory*

- Hubel, D. H. and T. N. Wiesel - *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*, Journal of Physiology, London, Vol. 160, 106-154, 1962

- Jain, A – *Fundamentals of Digital Image Processing*, Prentice-Hall, 1986

- JPEG 8-R8 ISO/IEC JTC1 Committee Draft. *Digital Compression and Coding of Continuous – Tone Still Images*. Technical report, International Organization for Standardization, August 1990.

- Kitter J., Illingworth J. – *Minimum Error Thresholding*, Pattern Recognition, p. 41-47, 1986

- Langdon, G.G. – *An Introduction to Arithmetic Coding*, IBM J. Res. Dev.,Vol 28, No. 2, p. 135-149, March 1984

- Lantuéjoul, C. – *Sur le modèle de Johnson-Mehl generalize*. Technical report, Centre de Morphologie Mathématique, Fontainebleau, France, 1977

- Lindeberg, T. – *Scale-Space Theory in Computer Vision*, Kuwer, Dordrecht, 1994

- Maragos P. A., Schafer R.W. – *Morphological Skeleton Representation and Coding of Binary Images*, IEEE Trans. On Acoustics, Speech And Signal Processing, 34(5) p.1228-1244, 1986

- Marr, D. - *Vision,* Freeman, 1982.

- Matlab, - *Image processing toolbox*, User manual, Version 3.0, The Mathworks Inc, section 10-11, April 2001.

- Meyer, F. and S. Beucher, - *Morphological Segmentation*, J. Visual Comm. Image Rep., 1990. 1(1): p. 21-46

- Minkowski, H. – *Volumen und Oberflaeche.* Math. Ann., Vol. 57, p. 447-495, 1903.

- MPEG, ISO/IEC JTC1/SC29/WG11. *First Draft of MPEG-4 Requirements.* Technical Report N0711, International Organization for Standardization, March 1994.

- MSDN (Microsoft Developer Network) for Visual Studio .NET, – *Using CTreeCtrl*, 1987-2001

- Nalwa, Vishvjit S. - *A Guided Tour of Computer Vision.* Addison-Wesley Publishing Company, 1993.

- Oppenheim, A.V. Schaffer R. W. – *Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1975

- Otsu, N - *A Threshold Selection Method from Grey-level Histograms*, IEEE Trans on Systems, Man and Cybernetics, p. 62-66, 1979

- Pasco, R. - *Source Coding Algorithms for Fast Data Compression*, Ph. D. Thesis, Dept. Of Electronic Engineering Stanford University, Stanford, 1976

- Pratt, W. K. – *Digital Image Processing*, John Wiley and Sons, New York, 1991

- Rissanen, J.J. - *Arithmetic Codings as Number Representations*, Acta Politech. Scand. Math. 31, p. 44-51, December 1976.

- Russ, J.C., - *The Image Processing Handbook.* Second ed, Boca Raton, Florida: CRC Press, 1995.

- Sahoo, P.K., Soltani S, Wong A.K., Chen Y.C. – *Survey of Thresholding Techniques*, Computer Vision, Graphics and Image Processing 41, 1988

- Selesnick, I. W., C. S. Burrus - *Exchange Algorithms that Complement the Parks-McClellan Algorithm for Linear Phase FIR Filter Design*, IEEE Trans. on Circuits and Systems, Part II, vol. 44, no. 2, p. 137-143, 1997.

- Serra, J. - *Image Analysis and Mathematical Morphology,* Vol. 1, Academic Press, 1982.

- Serra, J. - *Image Analysis and Mathematical Morphology,* Vol. 2, Academic Press, 1988.

- Strasser E., Schrom G., Wimmer K., Selberherr S. - *Accurate Simulation of Pattern Transfer Processes Using Minkowski Operations.* IEICE Transactions on Electronics, p. 91-99, 1994.

- Vincent L., Soile P. – *Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations,* IEEE Trans. On Pattern Analysis And Machine Intelligence vol. 13, No.6, p. 583-598, 1991

- Young, I.T. and L.J. Van Vliet, - *Recursive Implementation of the Gaussian Filter*, 1995.

- Young, R. A. - *The Gaussian Derivative Model For Machine Vision: Visual Cortex Simulation*, Journal of the Optical Society of America, July 1986

- Xu, C., J.L. Prince, - *Gradient Vector Flow: A New External Force for Snakes*, Proc. IEEE Conf. on Computer Vision Pattern Recognition (CVPR), Los Alamitos: Comp. Soc. Press, pp. 66-71, June 1997

- Xu, C., J.L. Prince, - *Generalized Gradient Vector Flow External Forces for Active Contours*," Signal Processing - An International Journal, 71(2), p. 131-139, December 1998.