NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

Topological Median Filters for Pattern Recognition

by

Vladimir Martincevic
B. Eng

Thesis submitted to
The Faculty of Graduate Studies and Research
In partial fulfilment of
the requirements for the degree of
Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of System and Computer Engineering
Carleton University
Ottawa, Ontario

June 2004

© Copyright
2004, Vladimir Martincevic

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-97425-1

Our file Notre référence

ISBN: 0-612-97425-1

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

The undersigned recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis

Topological Median Filters for Pattern Recognition

submitted by
Vladimir Martincevic, B.Eng

in partial fulfilment of the requirements
for the degree Master of Applied Science

Professor H. M. Schwartz
Thesis Supervisor

Professor R. A. Goubran
Chair, Department of Systems and Computer Engineering
Carleton University

June 2004

Abstract

This thesis demonstrates the use of the topological median filter for improvement of pattern recognition processes. The method used in the thesis relies on the edge preservation property of the topological median filter. Topological median filters do not have the same noise removal capability of the conventional median filters, but they better preserve the edges. This edge preservation property is very suitable for image prefiltering prior to the edge detection. The results without prefiltering and with prefiltering by conventional median filters and topological median filters are compared on edge detection performance and edge accuracy. The results show the advantages of the topological median filters over conventional median filters in edge preservation and edge position accuracy. Finally, processing speed is compared for MATLAB m-files and the C code implementation of MATLAB mex-files of the topological median “light” filter.

Keywords

Topological median filter, median filter, edge detection, feature extraction, image morphology, connected components, α -connectivity.

Acknowledgments

To Tanja for support and Alex for providing me with semiconductor images.

Table of Contents

1. Introduction.....	1
2. Overview of filters, edge detection, and image feature extraction	5
2.1 Linear Filters.....	6
2.2 Nonlinear Filters	16
2.3 Morphological Image Operators	27
2.4 Edge Detection.....	43
2.5 The Hough Transform.....	56
3. Topological Median Filters	63
4. Analysis and Results	88
4.1 Edge Detection Performance	88
4.2 Edge Accuracy.....	109
4.3 Processing Speed	115
5. Conclusions.....	118
References.....	120
Appendix.....	122

List of Figures

Figure 1 Two-dimensional linear system	7
Figure 2 Moving window.....	9
Figure 3 Linear spatial filtering.....	11
Figure 4 Frequency response of the Gaussian filter	12
Figure 5 Effect of the Gaussian filter on the Lena image.....	13
Figure 6 Effect of the Gaussian filter on a noisy Lena image	14
Figure 7 Effect of the Gaussian filter on the SEM image.....	15
Figure 8 Effect of the median filter on the Lena image.....	18
Figure 9 Effect of the median filter on a noisy Lena image	19
Figure 10 Effect of the Gaussian filter on the SEM image.....	20
Figure 11 Effect of the minimum filter on the Lena image.....	21
Figure 12 Effect of the minimum filter on the SEM image.....	22
Figure 13 Effect of the maximum filter on the Lena image	23
Figure 14 Effect of the maximum filter on the SEM image	24
Figure 15 Effect of the min max and max min filters on the Lena image	25
Figure 16 Effect of the min max and max min filters on the SEM image	26
Figure 17 Graphical representation of the set operations.....	29
Figure 18 Relationship between the set and logical operations	30
Figure 19 Example of dilation.....	31
Figure 20 Example of erosion	32
Figure 21 Examples of binary dilation and erosion.....	33
Figure 22 Example of grayscale dilation.....	36
Figure 23 Example of grayscale erosion	37
Figure 24 Binary and grayscale opening	38
Figure 25 Binary and grayscale closing	39
Figure 26 Structuring elements for detection of connected pixels	41
Figure 27 Dilate-and-mask algorithm for 4-connected pixels.....	41
Figure 28 Dilate-and-mask algorithm for 8-connected pixels.....	41
Figure 29 Edge detection block diagram	44
Figure 30 Edge detection test images	47
Figure 31 Effect of the Prewitt operator.....	48
Figure 32 Effect of the Sobel operator	49
Figure 33 The second derivative of a signal.....	50
Figure 34 Effect of the Laplacian operator	53
Figure 35 Marr-Hildreth method	53
Figure 36 Canny method.....	55
Figure 37 Space mapping: x-y and m-b	58

Figure 38 Division of the parameter space	58
Figure 39 Representation of a line by polar coordinates.....	59
Figure 40 Space mapping: x-y and polar parameter.....	60
Figure 41 Line location by the Hough transform	61
Figure 42 Generalized Hough transform model.....	62
Figure 43 Example of $E + p$ neighborhood	64
Figure 44 Paths connecting pixels p and q and their variations along the path.....	72
Figure 45 Connected pixels by thresholding	74
Figure 46 Conn-maps example	82
Figure 47 CMF and TMF response on a step edge.....	86
Figure 48 CMF and TMF response on a bar code test image.....	87
Figure 49 Step Edge Reference.....	89
Figure 50 Noisy step edge test images	90
Figure 51 The edge detected by the Sobel method without prefiltering	91
Figure 52 The edge detected by the Sobel method with median prefiltering.....	92
Figure 53 The edge detected by the Sobel method with TDL prefiltering	93
Figure 54 The edge detected by the Canny method without prefiltering.....	94
Figure 55 The edge detected by the Canny method with median prefiltering.....	95
Figure 56 The edge detected by the Canny method with TDL prefiltering	96
Figure 57 Edges on Lena image without prefiltering	99
Figure 58 Edges on Lena image with median prefiltering.....	100
Figure 59 Edges on Lena image with TLD prefiltering.....	101
Figure 60 Edges on a semiconductor image without prefiltering.....	102
Figure 61 Edges on a semiconductor image with median prefiltering.....	103
Figure 62 Edges on a semiconductor image with TDL prefiltering	104
Figure 63 Edges on a high magnification SEM image without prefiltering.....	105
Figure 64 Edges on a high magnification SEM image with median prefiltering....	106
Figure 65 Edges on a high magnification SEM image with TDL prefiltering	107
Figure 66 Test images used for the edge accuracy test.....	110
Figure 67 Edges without prefiltering	111
Figure 68 Edges with median prefiltering	112
Figure 69 Edges with TDL prefiltering.....	113

List of Tables

Table 1	Number of detected edge pixels	97
Table 2	Edges located by the Hough transform	114
Table 3	Processing speed of MATLAB implementation of <i>TL</i>	116
Table 4	Processing speed of MATLAB MEX implementation of <i>TL</i>	117

1. Introduction

This thesis investigates the use of digital imagery for the reverse engineering of integrated circuits. The objective is to use digital image processing techniques to trace the connections on integrated circuits and identify the components. Integrated circuit reverse engineering requires a huge number of images. The images are taken by a still digital camera equipped with high magnification lenses (typically x500 – x1000), or SEM (Scanning Electron Microscope). The images are taken in such a way that each image has some horizontal and vertical overlap. After all images are taken, the images are then stitched together, and divided into nonoverlapping congruent images called tiles. A mosaic represents one layer of an integrated circuit, and it is built of tiles. The mosaic can contain semiconductor structures (resistors, capacitors, diodes, MOSFET transistors, etc.) or integrated circuit interconnections. The content of the mosaic also determines its name – poly or metal. All mosaics of an integrated circuit are stacked in proper order and analyzed. The result of the analysis is the electrical schematic of the device. The reverse engineering process requires analysis of shapes (features) captured on each tile. The extraction of features is the merging of smaller shapes distributed over multiple tiles into bigger shapes, and the creation of electrical contacts among mosaics. This process is

time consuming and error prone, so the ultimate goal is to completely automate the integrated circuit reverse engineering process. The challenges that one can encounter in automating the integrated circuit reverse engineering process are: noise, inconsistent brightness and magnification, poor contrast, blur and pixel distortion. One can recall from optics that an image is in focus at the focal point. Our visual system cannot distinguish small deviations in distance from the focal point. The deviation that preserves the image sharpness is called the *depth of field*. The depth of field is very small (a few tens of nanometer for x1000 lenses), and slight room temperature drift (about 0.01°C) can cause a focal point drift that is larger than the depth of field. The drift causes blur in the images. The SEM images do not have temperature drift problems, but other problems such as edge pixel distortion, and poor or high material electron emissions are encountered. Besides these problems on a single mosaic, there is the problem of inconsistent background among mosaics. The sample preparation process, delayering and staining of a semiconductor device, gives different colours over layers of a single semiconductor device.

One way to solve the problem of automation of the reverse engineering process is to mimic the way that people are performing the process. It means that first the shapes of interest have to be recognized, then connected together, and finally interpret the meaning of the shapes. The most weight has to be put on the first step of the automated process – shape or pattern recognition. The accuracy and quality of the shape or pattern recognition process will have a significant impact on the quality of the whole automated process. Shapes in an image can be recognized by their edges. This is the reason why line detection algorithm such as the Hough transform [1], [2] and shape recognition

algorithms, pattern correlation [1], [2], the generalized Hough transform [2], [3], etc. start with edge detection. There are many different edge detection algorithms that differ in computational complexity. Some of the well known algorithms are: the highpass filter based algorithms [4], the first derivative or gradient based algorithms [1], [4], [5], the second derivative or Laplacian based algorithms [1], [4], the Marr-Hildreth algorithm [4] [5], the anisotropic diffusion based algorithms [4], and the fuzzy-logic based algorithms [6].

It is known that edges in images appear in places of fast intensity changes. The rate of an image intensity change determines the frequency of the image. It is also known that low frequencies carry intensity information, and high image frequencies carry shape information. In its nature, edge detection is a highpass filtering operation. The highpass filtering operations (e.g. derivative) are very sensitive to noise. Removing noise before edge detection is a very common method of improving edge detector performance. Image denoising is usually done with low-pass filters that can be either linear or nonlinear. Linear filters are easy to implement FIR digital filters. Nonlinear filters are more complicated than linear filters, but also provide better noise removal and edge preservation [4]. A method using nonlinear prefiltering is described in [1], [4], [7]. One sort of nonlinear filter is a median filter. There is a newly developed set of median filters [8] that use the property of pixel connectivity to its neighbours for filtering. These are topological median filters that use the fuzzy concept of α – connectivity in order to improve filter performance.

This thesis investigates the use of topological median filters for image processing and performance improvement in edge detection. Research has to be conducted to

determine if topological median filters can improve the edge detection process and if the improvement, if any, is worth the increase in complexity.

Chapter 2 gives an overview of different filters, techniques for edge detection, and shape (pattern) recognition. Chapter 3 gives an overview of topological median filters, Chapter 4 gives the performance results of topological median filters, and Chapter 5 gives conclusions and recommendations.

2. Overview of filters, edge detection, and image feature extraction

This chapter gives an overview of image processing in the spatial domain. The term spatial domain refers to the image plane itself, and all operations are performed by direct manipulation of pixels in an image. Image processing in the spatial domain is a wide research area and only the topics necessary for understanding of the work in this thesis are presented. The topics that are presented in this chapter include: linear filters, nonlinear filters, morphological operators, edge detection algorithms, and the Hough transform. The first three topics are necessary for the understanding of the morphological median filters that are described in Chapter 3. The last two topics give an overview of shape recognition.

All images in this thesis, if it is not otherwise notified, are grayscale discrete images defined on rectangular grid in 2-D space.

2.1 Linear Filters

Linear filters have an important role in digital image processing. Many techniques for modifying, improving, or representing digital images use the concept of discrete linear systems theory. This section provides an overview of discrete linear systems theory and its application to digital images. One can learn more about linear filters in [1] and [3].

The basic two-dimensional discrete-space signal is the *two-dimensional impulse function*, defined as [4]

$$\delta(m-p, n-q) = \begin{cases} 1, & m = p \text{ and } n = q \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The function in equation (1) is called the *Kronecker delta* function, the *unit sample sequence*, or the *discrete impulse function*. The response of the discrete linear system to the unit sample sequence is used to characterize discrete linear systems and is called the *impulse response* of the discrete linear system. Any discrete-space image f can be expressed in terms of the impulse function in (1) as [4]:

$$\begin{aligned}
f(m,n) &= \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} f(m-p, n-q) \delta(p, q) \\
&= \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} f(p, q) \delta(m-p, n-q)
\end{aligned} \tag{2}$$

Expression (2) shows that any discrete-space image can be written as sum of shifted and weighted unit impulses. The expression is also known as discrete-space linear convolution. The impulse response effectively describes linear space invariant systems. Let a two-dimensional system L be a process for image transformation. The process of an image transformation L can be written as [4]:

$$g(m, n) = L[f(m, n)] \tag{3}$$

The system for an image transformation is shown in Figure 1.

Figure 1 Two-dimensional linear system

The system in Figure 1 is linear if it satisfies the following [4]:

For any $f_1(m, n)$, $f_2(m, n)$ such that $g_1(m, n) = L[f_1(m, n)]$, $g_2(m, n) = L[f_2(m, n)]$, and any two constants a and b , then

$$ag_1(m, n) + bg_2(m, n) = L[af_1(m, n) + bf_2(m, n)] \tag{4}$$

The discrete linear system that is the easiest for analysis is *shift invariant* discrete linear system. The system $\mathbf{L}$ is *shift invariant* if for every $f(m,n)$ that satisfies equation (3), also satisfies

$$g(m-p, n-q) = \mathbf{L}[f(m-p, n-q)] \quad (5)$$

for any p, q [4].

Applying the unit impulse sequence on a discrete linear system, the *impulse response* of the discrete linear system is obtained.

$$h(m, n) = \mathbf{L}[\delta(m, n)] \quad (6)$$

The impulse response unambiguously describes a linear shift invariant system.

From equations (2), (3), (5), and (6), follows:

$$\begin{aligned} g(m, n) &= \mathbf{L} \left[\sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} f(p, q) \delta(m-p, n-q) \right] \\ &= \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} f(p, q) \mathbf{L}[\delta(m-p, n-q)] \\ &= \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} f(p, q) h(m-p, n-q) \\ &= f(m, n) * h(m, n) \end{aligned} \quad (7)$$

Equation (7) is the two-dimensional discrete space linear convolution of input f with impulse response h . The image filtering is the task of obtaining the output g by convolving an impulse response h with a given input f . This operation can also be expressed as follows:

Let $I : \mathbb{Z}^2 \rightarrow \mathbb{Z}$ be an integer-valued image defined on tuples of integers where $\mathbb{Z}$ is the set of all integers. Let $p \in \mathbb{Z}^2$ be a pixel location. Then $I(p) \in \mathbb{Z}$. Let $E = \{q_1, \dots, q_{N-1}\}$, where $q_j \in \mathbb{Z}^2$, be a set of $N-1$ pixel location, so that $E + p = \{q_1 + p, \dots, q_{N-1} + p\}$ defines a neighborhood of p . Then $I(E + p) = \{I(q) \mid q \in E + p\} \subset I$ is the set of values of I on neighbourhood $E + p$ [8] and is called the *moving window*. The moving window is shown in Figure 2. The moving window can be moved in any position on the image as long as all pixels in the moving window are contained in the image.

Figure 2 Moving window

The size of a moving window is N . Images defined on a rectangular grid have a rectangular moving window filter with R rows and C columns, R and $C \in \mathbb{Z}^+$. The impulse response $h: \mathbb{Z}^2 \rightarrow \mathbb{R}^{R \times C}$ is a matrix with R rows and C columns. The impulse response is also called the *kernel*. The coefficients of kernel h are already calculated by one of the FIR filter design methods [1], [4], [9]. Image filtering is done in the following way. The moving window is placed in one corner of the image. The values of the moving window and kernel are scalar multiplied then added (convolved). The sum is assigned to the pixel at position p . The moving window is shifted one pixel (horizontally or vertically). A new result is obtained for the new position of pixel p . The shifting operation is repeated until all possible moving window positions are exhausted. The process is graphically depicted in Figure 3 [1]. For convenience R and C have to be odd numbers, otherwise pixel p will not be in the center of the moving window. This means that the result of the filtering will be placed out of the image grid. The whole image cannot be filtered except if the moving window consists of pixel p only. For any other size of the moving window, one has to pad an image in order to filter the whole image. There are different padding options such as pad with 0, pad with image edge, etc [1].

Image filters have some properties that will be described. An image filter is separable if the impulse response $h(x, y) = h_1(x)h_2(y)$. An image filter is isotropic if it has the same response in all directions (circularly symmetric response). An image filter is said to be smooth if there is no oscillation in the frequency response of the filter in either passband or stopband.

Figure 3 Linear spatial filtering

The Gaussian filter is one of the most popular filters for linear digital image processing. The impulse response of the Gaussian filter is given as [4]:

$$h(m,n) = \frac{1}{2\pi\sigma^2} e^{-(m^2+n^2)/2\sigma^2} \quad (8)$$

The Gaussian filter is separable because it can be expressed as product of two 1-D Gaussian filters. It is isotropic because of the same response in all directions. It is also

smooth because there are no oscillations in its frequency response. The frequency response of the Gaussian filters with different value for σ is shown in Figure 4. On the same figure all of the Gaussian filter properties can be seen.

Figure 4 Frequency response of the Gaussian filter

(a) The frequency response of 3x3 the Gaussian filter with $\sigma = 0.7$. (b) The frequency response of 9x9 the Gaussian filter with $\sigma = 1.0$. (c) The frequency response of 15x15 the Gaussian filter with $\sigma = 2.0$. (d) The frequency response of 31x31 the Gaussian filter with $\sigma = 5.0$

Design of FIR filters is not in the scope of this thesis. One can learn more about FIR filter design in [1], [4] and [9].

The effect of the Gaussian filters in Figure 4 on the Lena image is shown in Figure 5. It can be seen that the image blur is proportional to the σ parameter of the Gaussian filter. The noise removal by the Gaussian filter is shown in Figure 6.

Figure 5 Effect of the Gaussian filter on the Lena image

(a) The original Lena image. (b) Lena image filtered by Gaussian filter with $\sigma = 0.7$. (c) Lena image filtered by Gaussian filter with $\sigma = 1.0$. (d) Lena image filtered by Gaussian filter with $\sigma = 2.0$. (e) Lena image filtered by Gaussian filter with $\sigma = 5.0$

Gaussian, impulse, and multiplicative noise are added to the original Lena image

(Figure 6-a). It can be seen from Figure 6-b to Figure 6-e that the amount of the noise

removed from the image is also proportional to the Gaussian filter parameter σ . It can also be seen that there is almost no visible difference between images Figure 5-e and Figure 6-e. The more noise is removed the more blurred images appear. Inability to preserve edge sharpness is the greatest deficiency of classical linear filter based images denoising methods [4].

Figure 6 Effect of the Gaussian filter on a noisy Lena image

(a) The original Lena image with added noise. (b) Noisy Lena image filtered by Gaussian filter with $\sigma = 0.7$. (c) Noisy Lena image filtered by Gaussian filter with $\sigma = 1.0$. (d) Noisy Lena image filtered by Gaussian filter with $\sigma = 2.0$. (e) Noisy Lena image filtered by Gaussian filter with $\sigma = 5.0$.

Figure 7-a shows a SEM image of an integrated circuit structure. Figure 7-b to 7-e show the application of the Gaussian filters from Figure 4 to the SEM image. It can be seen that the last image has lost all details of the comb structure. Figure 7-e, as shown, is unsuitable for any further processing.

Figure 7 Effect of the Gaussian filter on the SEM image

(a) The original SEM image. (b) SEM image filtered by Gaussian filter with $\sigma = 0.7$. (c) SEM image filtered by Gaussian filter with $\sigma = 1.0$. (d) SEM image filtered by Gaussian filter with $\sigma = 2.0$. (e) SEM image filtered by Gaussian filter with $\sigma = 5.0$ (The image courtesy of Alexander R. LaChance)

2.2 *Nonlinear Filters*

Denoising digital images with linear filters introduces blurring. Blurring spreads the edges on the digital images. The more noise removed, the more blurred the edges in an image appear. In order to preserve edges in noisy images, one can try nonlinear filters. Responses of nonlinear filters are not as straightforward to derive as linear filters, but their edge preserving performance is better than with linear filters.

The most popular class of nonlinear filters for digital image processing is the *order-statistics* or *rank-statistics* class. The order-statistics filters are spatial filters whose response at a point is based on ordering the pixels contained in the image area encompassed by the filter [1]. The area encompassed by the filter is the moving window defined in section 2.1.

Let $\mathbf{S}$ be an ascending ordered sequence of the values $I(E + p)$ with length N . Let n be the index or position in the sequence $\mathbf{S}$, $n \in [0, N - 1]$. The response of the *order-statistics* filter is determined with respect to pixel p in the following way. Ordered sequence $\mathbf{S}$ is created from image values $I(E + p)$. The value at a predetermined index n is assigned to the pixel p , $p = \mathbf{S}[n]$.

The most popular *order-statistics* filter is the median filter. The median operator is defined as [4]

$$\text{median}(\mathbf{S}) = \begin{cases} \mathbf{S}\left(\frac{N+1}{2}\right) & \text{if } N \text{ is odd} \\ \frac{\mathbf{S}\left(\frac{N}{2}\right) + \mathbf{S}\left(\frac{N}{2} + 1\right)}{2} & \text{if } N \text{ is even} \end{cases} \quad (9)$$

Expression (9) shows that the same conditions mentioned in section 2.1 are also valid for nonlinear filters. The size of the sequence $\mathbf{S}$ has to be an odd number in order to keep the result on the image grid. There is also the image padding requirement if the whole image has to be filtered. Expression (9) also shows that for an even number length sequence the median value cannot be assigned directly from the sequence $\mathbf{S}$, but it has to be calculated. To avoid the problem of calculating the median value, another definition of the median operator will be provided. The ordered sequence $\mathbf{S}$ can be expressed as the

histogram $H_{I(E+p)}$. $H_{I(E+p)}(k)$ is the number of pixel locations in $E + p$ at which image I has value k , $k \in \{0, 1, \dots, K\}$ [8]. If the histogram is normalized with respect to the

sequence length N , then the probability density function (PDF) $PDF_{I(E+p)} = \frac{1}{N} H_{I(E+p)}$

is obtained. The median operator is defined as [8],

$$\text{median}(\mathbf{S}) = k, \text{ such that } \sum_{i=0}^{k-1} PDF_{I(E+p)}(i) < 0.5 \text{ AND } \sum_{i=k}^K PDF_{I(E+p)}(i) \geq 0.5 \quad (10)$$

The effect of the median filter on the original Lena image is shown in Figure 8.

Figure 8 Effect of the median filter on the Lena image

(a) The original Lena image. (b) The original Lena image filtered with 3x3 median filter. (c) The original Lena image filtered with 5x5 median filter. (d) The original Lena image filtered with 7x7 median filter. (e) The original Lena image filtered with 9x9 median filter.

It can be seen from Figure 8 that the last image, Figure 8-e is less blurred than the image in Figure 5-e. The effect of the median filter on the Lena image with added noise is shown in Figure 9. One can see from Figure 9 that the images after median filtering have

the same or better appearance than the images in Figure 6. It has to be noted that the median filter requires a smaller filter kernel for the similar image appearance. Figure 10 shows the SEM image filtered with median filters.

Figure 9 Effect of the median filter on a noisy Lena image

(a) The original Lena image with added noise. (b) Noisy Lena image filtered with 3x3 median filter. (c) Noisy Lena image filtered with 5x5 median filter. (d) Noisy Lena image filtered with 7x7 median filter. (e) Noisy Lena image filtered with 9x9 median filter.

Another two very often used order-statistics filters are the minimum and the maximum filters [7]. Let $p \in \Sigma$ be a pixel location and let $I : \Sigma \rightarrow \{\mathbb{Z}^+ \cup 0\}$ be an image.

Let $E \subset \Sigma$ be a set of contiguous pixel locations and let $E + p \subset \Sigma$ be a neighborhood of shape E at pixel p . The minimum filter of image I , $MIN[I; E]$, is defined as [8]:

$$MIN[I; E](p) = \min\{I(E + p)\}, \quad (11)$$

where $I(E + p)$ is the set of values of I on neighbourhood $E + p$.

Figure 10 Effect of the Gaussian filter on the SEM image

(a) The original SEM image. (b) SEM image filtered with 3x3 median filter. (c) SEM image filtered with 5x5 median filter. (d) SEM image filtered with 7x7 median filter. (e) SEM image filtered with 9x9 median filter. (The image courtesy of Alexander R. LaChance)

The minimum filter is useful for finding the darkest pixels in an image. The effect of the minimum filter on the original Lena image is shown in Figure 11. The effect of the minimum filter on the SEM image is shown in Figure 12.

Figure 11 Effect of the minimum filter on the Lena image

(a) The original Lena image. (b) The original Lena image filtered with 3x3 minimum filter. (c) The original Lena image filtered with 5x5 minimum filter. (d) The original Lena image filtered with 7x7 minimum filter. (e) The original Lena image filtered with 9x9 minimum filter.

The maximum filter of image I , $MAX[I; E]$, is defined as [8]:

$$MAX[I; E](p) = \max\{I(E + p)\} \quad (12)$$

Figure 12 Effect of the minimum filter on the SEM image

(a) The original SEM image. (b) SEM image filtered with 3x3 minimum filter. (c) SEM image filtered with 5x5 minimum filter. (d) SEM image filtered with 7x7 minimum filter. (e) SEM image filtered with 9x9 minimum filter. (The image courtesy of Alexander R. LaChance)

The maximum filter is useful for finding the brightest pixels in an image. The effect of the maximum filter on the original Lena image is shown in Figure 13. The effect of the maximum filter on the SEM image is shown in Figure 14. The min and max operators can be used together as $MIN[MAX[I; E]; E]$, or $MAX[MIN[I; E]; E]$. These two combinations achieve very effective results.

Figure 13 Effect of the maximum filter on the Lena image

(a) The original Lena image. (b) The original Lena image filtered with 3x3 maximum filter. (c) The original Lena image filtered with 5x5 maximum filter. (d) The original Lena image filtered with 7x7 maximum filter. (e) The original Lena image filtered with 9x9 maximum filter.

The effects of $MIN[MAX[I; E]; E]$ and $MAX[MIN[I; E]; E]$ filters on the Lena image are shown in Figure 15. It can be seen for Figure 15 that $MIN[MAX[I; E]; E]$ suppresses dark details, while $MAX[MIN[I; E]; E]$ does the opposite. The effects of $MIN[MAX[I; E]; E]$ and $MAX[MIN[I; E]; E]$ filters on the SEM image are shown in Figure 16.

Figure 14 Effect of the maximum filter on the SEM image

(a) The original SEM image. (b) SEM image filtered with 3x3 maximum filter. (c) SEM image filtered with 5x5 maximum filter. (d) SEM image filtered with 7x7 maximum filter. (e) SEM image filtered with 9x9 maximum filter. (The image courtesy of Alexander R. LaChance)

Figure 16 illustrates better the dark and light detail suppression by $\text{MIN}[\text{MAX}[I; E]; E]$ and $\text{MAX}[\text{MIN}[I; E]; E]$ filter, respectively. One can see reduction of dark details in Figure 16-d, as well as reduction of light details in Figure 16-h.

Figure 15 Effect of the min max and max min filters on the Lena image

(a) The original Lena image filtered with 3x3 min max filter. (b) The original Lena image filtered with 5x5 min max filter. (c) The original Lena image filtered with 7x7 min max filter. (d) The original Lena image filtered with 9x9 min max filter. (e) The original Lena image filtered with 3x3 max min filter. (f) The original Lena image filtered with 5x5 max min filter. (g) The original Lena image filtered with 7x7 max min filter. (h) The original Lena image filtered with 9x9 min max filter.

There are also other nonlinear filters in use such as weighted median filter [4], adaptive median filter [1], alpha-trimmed mean filter [1], extremum filter [7], etc. One can learn more about them from the references.

Figure 16 Effect of the min max and max min filters on the SEM image

- (a) SEM image filtered with 3x3 min max filter. (b) SEM image filtered with 5x5 min max filter.
(c) SEM image filtered with 7x7 min max filter. (d) SEM image filtered with 9x9 min max filter.
(e) SEM image filtered with 3x3 max min filter. (f) SEM image filtered with 5x5 max min filter.
(g) SEM image filtered with 7x7 max min filter. (h) SEM image filtered with 9x9 max min filter.
(The image courtesy of Alexander R. LaChance)

2.3 Morphological Image Operators

The word *morphology* commonly denotes a branch of biology dealing with the form and structure of animals and plants [1]. In image processing the same word is used in the context of *mathematical morphology*. Mathematical morphology is used as a tool for extracting image components, such as shapes, boundaries, skeletons, etc., and it is based on set theory. Morphology can be applied on binary (black and white), grayscale and colour images, but the definitions of morphological operations change with the type of processed images. A set is defined in $\mathbb{Z}^2$ space for binary images, $\mathbb{Z}^3$ space for grayscale images, and in higher dimensional spaces for colour images. This section will describe morphology on binary and grayscale images only.

Let A be a set defined in $\mathbb{Z}^n$. If n-tuple $a = (a_1, a_2, \dots, a_n)$ is an element of A , it is written

$$a \in A,$$

similarly if a is not element of A , it is written

$$a \notin A.$$

The *union* of two sets A and B is the set of elements (pixels) that belong to either A or B [1]:

$$A \cup B = \{c \mid (c \in A) \vee (c \in B)\} \quad (13)$$

The *intersection* of two sets A and B is the set of elements (pixels) that belong to both A and B [1]:

$$A \cap B = \{c \mid (c \in A) \wedge (c \in B)\} \quad (14)$$

The *complement* of the set A is the set of elements (pixels) not belonging to A [1]:

$$A^c = \{c \mid c \notin A\} \quad (15)$$

The *difference* between the sets A and B is the set of elements (pixels) that belong to A and do not belong to B [1]:

$$A - B = \{c \mid (c \in A) \wedge (c \notin B)\} \quad (16)$$

The next two definitions are not generally found in set theory, but they are extensively used in morphology. The *translation* of the set A by the point x , denoted $(A)_x$, is defined as [1]:

$$(A)_x = \{c \mid c = a + x, a \in A\} \quad (17)$$

In plain words, to all coordinates a that belong to the set A will be added the value of the coordinate x .

The *reflection* of the set A , denoted $\hat{A}$, is defined as [1]:

$$\hat{A} = \{c \mid c = -a, a \in A\} \quad (18)$$

The reflections change the sign of all coordinates.

Graphical representation of the union, intersection, difference, complement, translation and reflection is shown in Figure 17 [1].

Figure 17 Graphical representation of the set operations [1]

(a) Two sets A and B . (b) The union of A and B . (c) The intersection of A and B . (d) The difference between A and B . (e) The complement of A . (f) Translation of A by x . (g) Reflection of B .

The set operations union, intersection, and complement for binary images can be replaced with logical operations, OR, AND, and NOT, respectively. The logical operations on binary images are shown in Figure 18 [1].

Figure 18 Relationship between the set and logical operations [1]

The logic operations on binary images (black represents binary 1's)

Two fundamental morphological operations are *dilation* and *erosion*. The dilation and erosion definitions will be first given for binary images.

Dilation of the set A by the set B is [1]:

$$A \oplus B = \{c \mid c = a + b, a \in A, b \in B\} \quad (19)$$

where A represents the image being operated on, and B is a set of pixels, a shape, that operates on the pixels of A in order to produce the result. The set B is called the structuring element, and its shape defines the nature of the specific operation. The dilation can also be expressed as [5]

$$A \oplus B = \bigcup_{b \in B} (A)_b \quad (20)$$

The graphical representation of dilation is shown in Figure 19. The **X** in sets A and B denotes the origin, and black squares in sets A and B represent binary 1's. The dilation of set A by structuring element B is obtained by placing the origin of the structuring element

Figure 19 Example of dilation

B over every element of the set A that is binary 1. The resulting image is obtained as the union (OR) of such placed structuring elements. Erosion of the set A by the set B is [5]:

$$A \ominus B = \{c \mid (B)_c \subseteq A\} \quad (21)$$

In other words, it is the set of pixels c such that the structuring element B , translated by c is contained in A . The graphical representation of erosion is shown in Figure 20.

Figure 20 Example of erosion

The **X** in sets A and B denotes the origin, and black squares in sets A and B represent binary 1's. The erosion of set A by structuring element B is obtained by placing the structuring element over pixels in the set A such that the structuring element B is contained in the set A . The positions of the origin of the structuring element are the result of erosion as shown in Figure 20. A binary image, structuring elements, and dilation and erosion of the binary image by structuring elements are shown in Figure 21. It has to be noted that in Figure 21 logical 1's are represented black, and logical 0's are represented white. It is important to realize that dilation and erosion are not inverse operations.

Figure 21 Examples of binary dilation and erosion

(a) Binary test image (black represents binary 1's). (b) Disc structuring element with diameter of 15 pixels. (c) Square structuring element with width of 9 pixels. (d) Square structuring element with width of 15 pixels. (e) Binary image dilated with disc (b). (f) Binary image dilated with square (c). (g) Binary image dilated with square (d). (h) Binary image eroded with disc (b). (i) Binary image eroded with square (c). (j) Binary image eroded with square (d).

However they are dual operations and the connection between them is given as [1]:

$$(A \ominus B)^c = A^c \oplus \hat{B} \quad (22)$$

It says that the complement of erosion is the same as dilation of the complement image by the reflected structuring element. The proof of the duality can be found in [1]. In binary image processing, dilation is used for bridging gaps that are smaller than the structuring element. Erosion is used for making gaps and thinning lines.

The dilation and erosion can be extended to grayscale images. It has to be noted that grayscale morphology operates on functions ($f : \mathbb{Z}^2 \rightarrow \mathbb{Z}$ or $f : \mathbb{Z}^2 \rightarrow \mathbb{R}$) rather than sets. The functions $f(x, y)$ and $b(x, y)$ are the input image and the structuring element, respectively. Dilation of f by b is defined as [1]:

$$(f \oplus b)(s, t) = \max\{f(s - x, t - y) + b(x, y) \mid (s - x), (t - y) \in D_f, (x, y) \in D_b\} \quad (23)$$

where D_f and D_b are the domains of f and b , respectively. The condition that $(s - x)$ and $(t - y)$ have to be in the domain of f , and x and y have to be in the domain of b , is analogous to the condition of “be element of the set” in expression (19). Erosion of f by b is defined as [1]:

$$(f \ominus b)(s, t) = \min\{f(s + x, t + y) - b(x, y) \mid (s + x), (t + y) \in D_f, (x, y) \in D_b\} \quad (24)$$

where D_f and D_b are the domains of f and b , respectively.

The grayscale dilation (erosion) can be computed in the following way [5]:

- 1) Position the origin of the structuring element over the first pixel of the image being dilated (eroded).
- 2) Compute the sum (difference) of each corresponding pair of pixel values in the structuring element and the image.
- 3) Find the maximum (minimum) value of all these sums (differences), and set the corresponding pixel in the output image to this value.
- 4) Repeat the process for each pixel in the image being dilated (eroded).

The effect of dilation on grayscale images is shown in Figure 22. Grayscale dilation has a similar effect as binary dilation. It widens “light” features in a grayscale image. It is used for removal of small “dark” features that are smaller than the structuring element. Grayscale dilation also increases the intensity level of the resulting image. These effects can be noticed in Figure 22-b, c, e, and f.

The effect of erosion on grayscale images is shown in Figure 23. Grayscale erosion has a similar effect as binary erosion. It widens “dark” features in a grayscale image. It is used for removal of small “light” features that are smaller than the structuring element. Grayscale erosion also decreases the intensity level of the resulting image. These effects can be noticed in Figure 23-b, c, e, and f.

The application of erosion immediately followed by dilation using the same structuring element is referred to as an opening operation. The opening operation is written as [1]:

$$A \circ B = (A \ominus B) \oplus B \quad (25)$$

The effect of opening on binary and grayscale images is shown in Figure 24.

Figure 22 Example of grayscale dilation

(a) The original Lena image. (b) The Lena image dilated by disc with diameter of 9. (c) The Lena image dilated by square with width of 5. (d) The SEM image. (e) The SEM image dilated by disc with diameter of 9. (f) The SEM image dilated by square with width of 5. (The SEM image courtesy of Alexander R. LaChance)

From Figure 24 it can be seen that opening eliminates protrusions on the binary image.

On grayscale images, opening removes narrow “light” features of an image.

Figure 23 Example of grayscale erosion

(a) The original Lena image. (b) The Lena image eroded by disc with diameter of 9. (c) The Lena image eroded by square with width of 5. (d) The SEM image. (e) The SEM image eroded by disc with diameter of 9. (f) The SEM image eroded by square with width of 5. (The SEM image courtesy of Alexander R. LaChance)

Similar to opening, the application of dilation immediately followed by erosion using the same structuring element is referred to as a closing operation. The closing operation is written as [1]:

$$A \bullet B = (A \oplus B) \ominus B \quad (26)$$

Figure 24 Binary and grayscale opening

(a) The binary image (black represents binary 1's). (b) The binary image opened by disc with diameter of 15. (c) The binary image opened by square with width of 15. (d) The Lena image. (e) The Lena image opened by disc with diameter of 9. (f) The Lena image opened by square with width of 5. (g) The SEM image. (h) The SEM image opened by disc with diameter of 9. (f) The SEM image opened by square with width of 5. (The SEM image courtesy of Alexander R. LaChance)

The effect of closing on binary and grayscale images is shown in Figure 25.

Figure 25 Binary and grayscale closing

(a) The binary image (black represents binary 1's). (b) The binary image closed by disc with diameter of 15. (c) The binary image closed by square with width of 15. (d) The Lena image. (e) The Lena image closed by disc with diameter of 9. (f) The Lena image closed by square with width of 5. (g) The SEM image. (h) The SEM image closed by disc with diameter of 9. (i) The SEM image closed by square with width of 5. (The SEM image courtesy of Alexander R. LaChance)

From Figure 25 it can be seen that closing eliminates small gaps on the binary image.

On grayscale images, closing removes narrow “dark” features of an image. Both, opening and closing smooth the image, but the smoothing effect is focused on different features of an image.

A well-known morphological operation is the extraction of connected components. Detection and extraction of connected components is described in [1], [10], and [11]. The extraction of connected components is a very important algorithm for this thesis, and its application will be explained in section 3.

A simple method for extracting connected components is the dilate-and-mask algorithm [1] and [10].

Let $Y \subseteq A$ represent a connected component in the set A . Let $p \in Y$ be a pixel p in the connected component Y . The following iterative expression gives all elements of Y [1]:

$$X_k = (X_{k-1} \oplus B) \cap A, \quad k = 1, 2, 3, \dots \quad (27)$$

where $X_0 = p$, and B is a structuring element. If $X_k = X_{k-1}$, the algorithm converges and $Y = X_k$. The structuring elements for 4-connected and 8-connected pixels are shown in Figure 26. The progress of the dilate-and-mask algorithm for 4-connected pixels is shown in Figure 27. The grey represents binary 1's in the binary image. The black pixel in Figure 27-a is the start point p . The 4-connected region grown from pixel p is shown in Figure 27-c. The progress of the dilate-and-mask algorithm for 8-connected pixels is shown in Figure 28. It has to be noted that the algorithm always converges, because the

Figure 26 Structuring elements for detection of connected pixels

(a) The structuring element for detection of 4-connected pixels (grey represents logical 1's). (b) The structuring element for detection of 8-connected pixels.

Figure 27 Dilate-and-mask algorithm for 4-connected pixels

(a) The Binary image (grey represents binary 1's) with the start point p (black). (b) The first dilation by 4-connected structuring element shown in black. (c) The 4-connected region that contains pixel p shown in black.

Figure 28 Dilate-and-mask algorithm for 8-connected pixels

(a) The Binary image (grey represents binary 1's) with the start point p (black). (b) The first dilation by 8-connected structuring element shown in black. (c) The 8-connected region that contains pixel p shown in black.

image contains a finite number of pixels.

Another method of implementing the extraction of connected components is the use of the algorithm for labeling connected components in a binary image [4]. The algorithm assigns different numbers to different connected regions. The dilate-and-mask algorithm is the choice of the authors of [8] and it is used for the extraction of connected components in section 3. The algorithm is iterative and has to copy data, but can finish without parsing the whole image. On the other hand, the labeling algorithm does not need to copy data, but cannot finish before the last pixel is labeled.

One can learn more about other morphological operators from the references [1], [4], [5], [10], and [11].

2.4 Edge Detection

Edge detection is one of the most commonly used operations in image analysis. The reason for this is that edges outline an object. Edges are boundaries between an object and the background, and also indicate the boundaries between overlapping objects [5]. Edge detection is usually the first step for image segmentation and pattern recognition. Edges are sudden changes in image intensity, and usually edges are modeled with a step function. The problem with this model is that edges are not perfect step functions in real images, but have a gradual transition between two intensity levels. In addition to the edge transition, there is inherent noise that makes edge detection even more difficult. A good edge detector has to find all edges without finding false edges, and the distance between detected edges and real edges should be as small as possible.

The block diagram of an edge detector is shown in Figure 29. The first step of an edge detection process is the edge map extraction. The edge map extraction is the gathering of the pixels that are potential edge candidates. After extraction, the weak edge candidates are removed by thresholding. The thresholded edge map is thinned in order to get edges. The edge thinning is the process that removes all edge candidates that are not local maximums in the edge map. This last step is necessary if one pixel thick edges are required. Some algorithms omit the last two steps, while some of them reverse the order

of the last two steps.

Figure 29 Edge detection block diagram

The edge detectors are generally distinguished by the method of locating edges. The simplest method for edge location is a highpass filter [4]. The highpass filter is usually a Gaussian highpass filter. This edge detection method is very sensitive to noise and will not be discussed further in this thesis.

The gradient-based edge detection methods [1], [4], and [5] use first or second order derivatives of the signal for the edge location. Let $p = (p_x, p_y)$ be a pixel position, and $I(p)$ the intensity of the image at position p . Then the image gradient at position p is [4]

$$\nabla I(p) = \left(\frac{\Delta I}{\Delta p_x}, \frac{\Delta I}{\Delta p_y} \right) \quad (28)$$

It has to be noted that the gradient in expression (28) is a function of discrete arguments. The simplest way to calculate the gradient is the first difference. The gradient based on the first difference method is defined as [4]:

$$\begin{aligned}\nabla_x I(p) &= I(p_x, p_y) - I(p_{x-1}, p_y) \\ \nabla_y I(p) &= I(p_x, p_y) - I(p_x, p_{y-1})\end{aligned}\tag{29}$$

The intensity and direction of the gradient is determined as

$$|\nabla I(p)| = \sqrt{(\nabla I_x)^2 + (\nabla I_y)^2} \quad \text{and} \quad \angle(\nabla I(p)) = \tan^{-1}\left(\frac{\nabla I_y}{\nabla I_x}\right)$$

respectively. The first differences can be expressed as a filter kernel with impulse response $h_1(p)$ and $h_2(p)$ [4]

$$h_1(p) = \begin{bmatrix} 0 & 0 \\ -\mathbf{1} & \mathbf{1} \end{bmatrix} \quad h_2(p) = \begin{bmatrix} 1 & 0 \\ -\mathbf{1} & 0 \end{bmatrix}$$

Boldface elements indicate origin position [4].

These two kernels have the problem that the center point of the horizontal and vertical gradients does not lie in the same position, and it is off the grid. To overcome this problem the central differences are used for gradient estimation. The gradient based on the central difference method is defined as [4]

$$\begin{aligned}\nabla_x I(p) &= \frac{1}{2} (I(p_{x+1}, p_y) - I(p_{x-1}, p_y)) \\ \nabla_y I(p) &= \frac{1}{2} (I(p_x, p_{y+1}) - I(p_x, p_{y-1}))\end{aligned}\tag{30}$$

The scaling value of 0.5 is because of the distance between two pixels. This gradient magnitude threshold value can be always adjusted for the scaling factor [4]. For simplicity, the scaling factor is omitted in the remainder of the text. The central differences can be expressed as filter kernels with impulse response $h_1(p)$ and $h_2(p)$ [4]:

$$h_1(p) = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad h_2(p) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

Although the edge position problem is solved with the latest operator, noise is an inherent problem of all derivative operators. The idea to detect edges in one direction and smooth in the orthogonal direction produces two well-known operators: Prewitt and Sobel. The filter kernels of the Prewitt operator are [4]:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

The Sobel operator, one of the most widely used gradient edge detectors, has the following filter kernels [4]:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The Sobel operator is usually a better choice because its lowpass filter produces better smoothing. The images in Figure 30 will be used as test for the Prewitt and Sobel operators.

Figure 30 Edge detection test images

(a) The original Lena image. (b) Noisy Lena image. (c) The SEM image. (The SEM image courtesy of Alexander R. LaChance)

The effect of the Prewitt and Sobel operators are shown in Figure 31 and Figure 32, respectively. Although the Sobel operator is a better choice than the Prewitt operator the significant difference between Figure 31 and Figure 32 is not noticeable.

Figure 31 Effect of the Prewitt operator

(a) Figure 30-a processed by horizontal, vertical and both Prewitt operators (top to bottom). (b) Figure 30-b processed by horizontal, vertical and both Prewitt operators (top to bottom). (c) Figure 30-c processed by horizontal, vertical and both Prewitt operators (top to bottom). (Figure 30-c courtesy of Alexander R. LaChance)

Figure 32 Effect of the Sobel operator

(a) Figure 30-a processed by horizontal, vertical and both Sobel operators (top to bottom). (b) Figure 30-b processed by horizontal, vertical and both Sobel operators (top to bottom). (c) Figure 30-c processed by horizontal, vertical and both Sobel operators (top to bottom). (Figure 30-c courtesy of Alexander R. LaChance)

Another popular edge detection method is based on the second order derivative. The method is often called the Laplacian. The zero crossings of Laplacian operators represent maximums of the first derivative. The edges are represented as zeros in the second derivative [4]. This approach has two problems. The first problem is false zero crossing when the first derivative reaches its minimum. The false zero crossing is shown in Figure 33. A wide area in where the second derivative is zero can be noticed in Figure 33-c. The second problem is sensitivity to noise. The Laplacian is even more sensitive to noise

Figure 33 The second derivative of a signal

(a) Image signal in one direction. (b) The first derivative of the signal. (c) The second derivative of the signal.

than the gradient. It has to be noted that the Laplacian is a scalar and requires only one

filter kernel for edge detection. The discrete Laplacian operator ∇^2 is given as [4]:

$$\nabla^2 I(p) = \nabla \nabla I(p) = \nabla_x (\nabla_x I(p)) + \nabla_y (\nabla_y I(p)) \quad (31)$$

The simplest Laplacian operator can be derived as follows. First a discrete gradient operator is defined as the first difference [4].

$$\begin{aligned} \nabla_x I(p) &= I(p_{x+1}, p_y) - I(p_x, p_y) \\ \nabla_y I(p) &= I(p_x, p_{y+1}) - I(p_x, p_y) \end{aligned} \quad (32)$$

The second derivative is built by applying the first difference a second time.

$$\begin{aligned} \nabla_x (\nabla_x I(p)) &= \nabla_x I(p_x, p_y) - \nabla_x I(p_{x-1}, p_y) \\ \nabla_y (\nabla_y I(p)) &= \nabla_y I(p_x, p_y) - \nabla_y I(p_x, p_{y-1}) \end{aligned} \quad (33)$$

Combining equation (32) and (33) produces

$$\begin{aligned} \nabla_x (\nabla_x I(p)) &= I(p_{x+1}, p_y) - 2I(p_x, p_y) + I(p_{x-1}, p_y) \\ \nabla_y (\nabla_y I(p)) &= I(p_x, p_{y+1}) - 2I(p_x, p_y) + I(p_x, p_{y-1}) \end{aligned} \quad (34)$$

The definition of the Laplacian follows as:

$$\begin{aligned}
\nabla^2 I(p) &= \nabla \nabla I(p) = \nabla_x (\nabla_x I(p)) + \nabla_y (\nabla_y I(p)) \\
&= I(p_{x+1}, p_y) - 2I(p_x, p_y) + I(p_{x-1}, p_y) + I(p_x, p_{y+1}) - 2I(p_x, p_y) + I(p_x, p_{y-1}) \\
&= I(p_{x+1}, p_y) + I(p_{x-1}, p_y) + I(p_x, p_{y+1}) + I(p_x, p_{y-1}) - 4I(p_x, p_y)
\end{aligned} \tag{35}$$

The filter kernel of the Laplacian operator is [4]

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The edge detection algorithm based on the Laplacian operator does not require thresholding and edge thinning, although it requires checking for false edges. The algorithm must also address problem of edges outside the grid. If two adjacent edge candidates have different signs, the edged is placed somewhere between these edge candidates. In this case one of the edge candidates has to be assigned as the edge. The effect of the Laplacian operators is shown in Figure 34. The algorithm shown in Figure 34 is the plain Laplacian filter without false edges suppression. The importance of the false edges suppression for the Laplacian edge detector is obvious.

The noise sensitivity of the Laplacian operator is solved with filtering an image with the lowpass Gaussian filter. The Gaussian filter smoothes an image prior to edge detection with the Laplacian. This improved version of the Laplacian based edge detection is called the Laplacian of Gaussian (LoG) or Marr-Hildreth method. Complete implementation of the method is given in [5]. The results of the application of

Marr-Hildreth method in Figure 30 are shown in Figure 35. The improvement in comparison to the Prewitt, Sobel, and Laplacian methods can be noticed.

Figure 34 Effect of the Laplacian operator

(a) Figure 30-a processed by Laplacian operator. (b) Figure 30-b processed by Laplacian operator. (c) Figure 30-c processed by Laplacian operator. (Figure 30-c courtesy of Alexander R. LaChance)

Figure 35 Marr-Hildreth method

(a) Figure 30-a processed by Marr-Hildreth method. (b) Figure 30-b processed by Marr-Hildreth method. (c) Figure 30-c processed by Marr-Hildreth method. (Figure 30-c courtesy of Alexander R. LaChance)

An edge detector that further improves edge detection in the presence of noise is the Canny edge detector. Canny's method tries to accomplish three simultaneous goals: low rate of detection errors, good edge localization, and only a single detection response per edge. In his work Canny determined the optimal 1-D edge detector for the step edge and showed that its impulse response can be approximated by the derivative of a Gaussian [5]. Canny's method also uses nonmaximum suppression. The nonmaximum suppression is the method for extracting the pixels that are the strongest edge candidates. The magnitude of the gradient at an edge pixel should be greater than the magnitude of the gradient of the pixels on each side of the edge. The method has to eliminate the pixels on an edge side. Another difference between the Canny method and other edge detection methods described here is that the Canny method uses two thresholds. Complete implementation of the Canny edge detector is given in [5]. The results of the application of Canny method on Figure 30 are shown in Figure 36. If Figure 35 and Figure 36 are compared, it can be seen that the Canny method is better than Marr-Hildreth method for images (a) and (c), but gives very poor result for image (b). Moreover, the Sobel and Prewitt methods give better results on that image. This requires some explanations. According to Canny's work [12], the method is optimized for edge detection in the presences of white noise. Besides the white noise, the image in Figure 30-b purposely contains other types of noise. The other images also contain some other types of noise, but the white noise is dominant, and that is not case with the image in Figure 30-b. This is the explanation why the image in Figure 30-b presents a particular problem for the Canny method.

Figure 36 Canny method

(a) Figure 30-a processed by Canny method. (b) Figure 30-b processed by Canny method. (c) Figure 30-c processed by Canny method. (Figure 30-c courtesy of Alexander R. LaChance)

These edge detection methods are the most popular ones. There are some other methods that have to be mentioned. These are Kirch [5], Shen-Castan (Infinite Symmetric Exponential Filter) [5], morphological gradient [1] and [4], anisotropic diffusion [4], etc. More information on these edge detectors can be found in the references.

2.5 The Hough Transform

The Hough transform is an effective way for detecting and localizing geometric primitives such as lines and circles in an image. The Hough transform requires binary images. The usual way of converting grayscale images to binary ones is by thresholding. The thresholding method is not suitable for the Hough transform, because of the amount of data for processing by the Hough transform. The increase of the data amount decreases the success rate of finding required geometric primitives. The method of choice for converting grayscale images to binary ones that are suitable for the Hough transform is edge detection [1] and [2]. The edges give enough data for detecting geometric primitives, while suppressing other irrelevant data. The geometric primitives can be expressed as [1]

$$g(\mathbf{v}, \mathbf{c}) = 0 \tag{36}$$

The arguments of equation (36) are vectors of coordinates and coefficients respectively. The Hough transform can easily detect geometric primitives that can be expressed by equation (36). The principle of the Hough transform is mapping of a geometric primitive from x - y into a single point in parameter space. The point position determines the

location of the geometric primitive, while the intensity of the point determines the strength of the detected geometric primitive. The principle of the Hough transform will be described on the simplest example, a straight line.

A line in $\mathbb{R}^2$ can be expressed as [1]:

$$y = mx + b, \text{ or } y - mx - b = 0 \quad (37)$$

where m and b are the slope and the y-intercept respectively. Any straight line can be characterized by m and b . There is also an infinite number of lines that pass through point (x, y) . All of these lines have unique parameters m and b , and the equation of a line can be also expressed as

$$b = y - mx \quad (38)$$

The m - b space is also called the parameter or Hough space. The line can be defined in x - y or m - b space. Every line is uniquely defined with two points. Figure 37 shows the line defined with two points in x - y space and the same points mapped in the parameter or m - b space.

Mapping these two points in parameter space will map two different points on the same line in x - y space to the single point in the parameter space. This mapping gives the principle of the Hough transform. In plain words, the Hough transform maps points that satisfy equation (36) into a single point in the Hough or parameter space. The algorithm

only has to count the number of these mappings in order to determine position and orientation of the line. The parameter space is divided into so-called accumulator cells, as shown in Figure 38.

Figure 37 Space mapping: x-y and m-b

(a) A line defined by two points. (b) The points mapped into the parameter space.

The cells are used for counting of the mapping occurrences. The number of the cells determines the resolution of the Hough transform. Each point in x-y space is mapped

Figure 38 Division of the parameter space

into parameter space. The result of mapping is rounded to the nearest cell in parameter space and that cell is incremented. The cells with count above an assigned threshold are considered as line occurrences. The coordinates of the cells are parameters of the detected lines. This method of line detection and localization has a problem in that the m parameter can have an infinite value. To avoid an infinite value for the slope, the line is expressed in polar coordinates. Equation (37) can be written as [1]

$$x \cos \theta + y \sin \theta = \rho \quad (39)$$

Figure 39 illustrates the geometrical interpretation of expression (39). The range for θ and ρ parameters are $\pm 90^\circ$ and D , respectively, where D is diagonal length of the image. The illustration of the parameter space and detected lines is shown in Figure 40. The Hough transform is very robust, and can tolerate high levels of noise.

Figure 39 Representation of a line by polar coordinates

Figure 40 Space mapping: x-y and polar parameter

(a) A line defined by two points. (b) The points mapped into the polar coordinate parameter space.

Quantization of θ - ρ space into the accumulator cells is an important design decision.

Increasing the number of cells can reduce the quantization problem, but a substantial amount of memory can be required for the Hough transform. Reduction in memory can be achieved by using only the cells of interest and omitting other cells [2]. An example of the line detection by the Hough transform is shown in Figure 41.

Detection and localization of circles and ellipses is similar to the method explained for lines. The only difference is that the accumulator cells are 3-D with three parameters in the parameter space [1].

The Hough transform can also be applied on shapes that cannot be expressed by equation (36). The Hough transform that is intended for finding and locating 2-D arbitrary shapes is called the generalized Hough transform. The first step of the generalized Hough transform is the model definition [2]. The model is a shape that has to be detected and localized, and it is represented by the outline or boundary of the shape.

Figure 41 Line location by the Hough transform

(a) The original grayscale image. (b) The edge map obtained by Sobel method. (c) The image of the Hough space accumulator cells. (d) the lines obtained from the Hough space peaks. (Figure 41-a courtesy of Alexander R. LaChance)

A model is shown in Figure 42. For each point on the model's boundary the angle between horizontal axis and tangent on the boundary is calculated. The reference point of the model is chosen and vectors for all boundary points are computed.

Figure 42 Generalized Hough transform model

The tangent $\theta(X_m)$ and the vector intensity $\rho(X_m)$ for one model point X_m

A lookup table containing angles and reference vectors is created. During recognition the table is indexed by discrete values of the angle. The table is called the *R-table* or the *reference table*. The recognition process is similar to the previously explained Hough transform. The 2-D accumulator is initialized. Every point in the image is placed in the R-table and the reference point candidates are calculated. The accumulator cells of the previously calculated reference point candidates are incremented. The cell with the highest count corresponds to the reference point of the shape. Rotation and scaling can also be incorporated in this method. It has to be noticed that the generalized Hough transform is very demanding on memory and it is computationally intensive. One can learn more about the generalized Hough transform and improvements to the algorithm in [1], [2], and [3].

3. Topological Median Filters

The topological median filter is a new type of median filter for images. The topological median filter implements some existing ideas and some new ideas on fuzzy connectedness to improve, over a conventional median filter, the extraction of edges in noise. The concept of α – connectivity is defined and used to create an algorithm for computing the degree of connectedness of a pixel to all the other pixels in an arbitrary neighborhood [8]. This section gives a brief overview of the terminology necessary for understanding topological median filters, as published in [8]. Image features and the median filter are discussed first, then digital and fuzzy topology, results on fuzzy connectedness, algorithms, definition of topological median filters and finally statistical properties of the topological median filters.

Let $I : \mathbb{Z}^n \rightarrow \mathbb{Z}$ be an integer-valued image defined on n-tuples of integers where $\mathbb{Z}$ is the set of all integers. Let $p \in \mathbb{Z}^n$ be a pixel location. Then $I(p) \in \mathbb{Z}$. Let $E = \{q_1, \dots, q_m\}$, where $q_j \in \mathbb{Z}^n$, be a set of m pixel locations, so that $E + p = \{q_1 + p, \dots, q_m + p\}$ defines a neighborhood of p . In 2-D space $n = 2$. Thus, p is a pixel location, $I(p)$ is a pixel value, $E + p$ is a neighborhood of p , and

$I(E + p) = \{I(q) \mid q \in E + p\}$ is the set of values of I on neighborhood $E + p$ [8]. One example of an $E + p$ neighborhood is shown in Figure 43.

Figure 43 Example of $E + p$ neighborhood

The concept of a *path* is also necessary for the topological median filter. A path ρ is defined within a set of pixel locations, E , as a set $\rho = \{p = p_0, p_1, \dots, p_m = q\} \subset E \subset \mathbb{Z}^n$ such that the distance between p_k and p_{k+1} is no greater than $\sqrt{n}$. In 2-D space $n = 2$, and pixels on the path must be 8-neighbours of each other. The topological median filters use only contiguous neighbourhoods.

For any given p , the values, $I(E + p)$, can be collected in a histogram,

$H_{I(E+p)} : \mathbb{Z} \rightarrow \mathbb{Z}$. $H_{I(E+p)}(k)$ is the number of pixel locations in $E + p$ at which I has value k . If the histogram is normalized, it forms a sample density function [8]

$$\tilde{P}_{I(E+p)} = \frac{1}{m} H_{I(E+p)} \quad (40)$$

which is an estimate of the probability density function (PDF) of I on $E + p$, $P_{I(E+p)}$.

The median filter is defined as [8]:

$$\begin{aligned} \text{med}\{I\}(p) = \text{med } E + p\{I\} = \text{median of } \tilde{P}_{I(E+p)} = k \text{ such that} \\ \sum_{i=0}^{k-1} \tilde{P}_{I(E+p)}(i) < 0.5 \text{ AND } \sum_{i=k}^K \tilde{P}_{I(E+p)}(i) \geq 0.5 \end{aligned} \quad (41)$$

where $K = \max\{I(q) \mid q \in E + p\}$.

In plain words, probabilities of an image intensity value occurrence are summed starting from the minimum intensity value. Expression (41) states that the value at which the sum of probabilities is greater or equal to 50% is the median value. The equivalent statement is that the value at the center of the ordered sequence of values from $E + p$ is the median value.

Given a noisy image I , the goal is to recover the underlying noise-free image J .

Assume that $I = J + N$, where N is a noise image such that each $N(p)$ is an independent, identically distributed (IID) random variable with (unknown) PDF, P_N . J is presumed to be deterministic but each pixel value, $I(p)$, in the noisy image is a random variable with PDF $P_{I(p)}(k) = P_N(k - J(p))$. That is, $P_{I(p)}$ is P_N with mean shifted by $J(p)$. It is assumed that an image feature is a contiguous region in $\mathbb{Z}^n$ on which J has

approximately constant intensity or approximately linear intensity variation in all directions. The feature boundaries are intensity edges. Let $\Phi\{p\}$ designate the feature in J to which pixel p belongs. Define $\#\{\cdot\}$ to be the cardinality operator or in plain words the size operator. Then for given a set B of pixel locations, $\#\{B\}$ is the number of pixel locations in B . Consider $\Phi\{p\} \cap \{E + p\}$, the subset of the neighborhood of p that contains pixels from the feature to which p belongs. If $\{E + p\}$ is small enough that

$$\#\{\Phi\{p\} \cap \{E + p\}\} > \frac{1}{2} \#\{E + p\} \quad (42)$$

but is large enough that $\tilde{P}_{I(E+p)}(k)$, the sample density of I , approximates $J(p) + P_N(k)$ then $\text{med}_{E+p}\{I\}(p)$ is a reasonable estimate of $J(p)$ [8]. This is especially true if $P_N(k)$ has zero mean and has significant weight at values of $|k|$ larger than the variation of J over $\Phi\{p\}$. If one could determine $\Phi\{p\} \cap \{E + p\}$, the set of pixels in the neighborhood that are in the same feature as the center pixel, then one could use only those pixels to estimate the true value of $J(p)$ by computing $\text{med}_{\{E+p\} \cap \Phi(p)}\{I\}$. Topological connectedness can be used to estimate $\Phi\{p\} \cap \{E + p\}$. Restricting the estimate of the median to only part of the neighborhood may cause the sample size to be too small to get a good estimate [8].

The notion of connectedness is a fundamental concept of digital topology. Digital topology has straightforward application to binary images where regions, edges and

object boundaries can be defined and located with precision. By modeling an intensity image as a fuzzy set, digital topology is extended from binary to intensity images [8]. A brief overview of fuzzy set theory necessary for understanding connectedness of intensity images follows.

In contrast to the set theory where an element has discrete membership in the set (0 or 1), an element in fuzzy set theory has continuous membership in the set in an interval $[0, 1]$. An intensity (grayscale) image can be modeled as a fuzzy set by mapping the normalized grey level of a pixel to a fuzzy set degree of membership (DOM).

Let Σ be a set of points. A generic element of Σ is denoted by p . Let $E \subset \Sigma$ with $p \in E$.

This relationship can be represented by the characteristic function, $\mu_E(p)$, such that

$$\mu_E(p) = \begin{cases} 1, & \text{if } p \in E \\ 0, & \text{if } p \notin E \end{cases} \quad (43)$$

A characteristic function can also take any value in the interval $[0, 1]$ [8]. In that case

$\mu_E(p)$, becomes a fuzzy membership function, and $\mu_E(p)$ indicates the DOM of p in E .

The statement that p is not in E is written as $\mu_E(p) = 0$. If p is fully in E , then

$\mu_E(p) = 1$. If $0 < \mu_E(p) < 1$, then p has partial membership in E . A subset E of a

universal set Σ is a fuzzy set if for each $p \in \Sigma$, $\mu_E(p)$ has a value in the closed interval

$[0, 1]$ [8]. A fuzzy set, E , is *empty* if and only if $\mu_E(p) = 0$ for every $p \in \Sigma$ [8]. Two

fuzzy sets, E and, F are said to be *equal* if and only if $\mu_E(p) = \mu_F(p)$ for every $p \in \Sigma$

[8]. The *complement* of a fuzzy set E is denoted by $\bar{E}$ and its membership function is

defined by $\mu_{\bar{E}}(p) = 1 - \mu_E(p)$ for every $p \in \Sigma$ [8]. A fuzzy set E is a *subset* of F ($E \subset F$) if and only if $\mu_E(p) \leq \mu_F(p)$ for every $p \in \Sigma$ [8]. The *union* of fuzzy sets E and F is G whose membership function is [8]

$$\mu_G(p) = \max\{\mu_E(p), \mu_F(p)\} \quad (44)$$

for all $p \in \Sigma$.

The intersection of fuzzy sets E and F is H whose membership function is [8]:

$$\mu_H(p) = \min\{\mu_E(p), \mu_F(p)\} \quad (45)$$

for all $p \in \Sigma$.

All of the connectivity definitions that follow are applicable on either a 4-connected or an 8-connected topology on a finite, rectangular pixel grid. Σ is defined as the set of pixel locations from an image $I : \Sigma \rightarrow \{\mathbb{Z}^+ \cup 0\}$. Σ is an R row by C column contiguous subset of $\mathbb{Z}^2$. R and C are positive integers and $\mathbb{Z}^+$ is the set of all positive integers. I is the mapping from a specific pixel location to a specific intensity value. $I(p)$ is the value of the image at pixel $p \in \Sigma$ and $I(\Sigma)$ represents the entire matrix of intensities. Depending on the context, I will often be used to mean $I(\Sigma)$.

Definition 1 [8]: Degree of membership in an image. Let $I : \Sigma \rightarrow \{\mathbb{Z}^+ \cup 0\}$ be an image.

Assume $I(p) \neq 0$ for at least one $p \in \Sigma$. Let $K = \max\{I(p) \mid p \in \Sigma\}$. The Degree Of

Membership (DOM) of p in I is defined as

$$\mu_I(p) = \frac{I(p)}{K} \quad (46)$$

The DOM, $\mu_I(p)$, of a pixel is its normalized intensity. $\mu_I(p)$ can be considered to be the DOM of p in the set of “bright” pixels. Similarly, the degree of membership of p in the set of “dark” pixels can be defined as

$$\mu_{\bar{I}}(p) = 1 - \mu_I(p) \quad (47)$$

$E + p$ is used to denote a specific neighborhood in the image. That is, $E \subset \mathbb{Z}^2$, is a set of contiguous pixel locations in a specific geometric configuration. $E + p \subset \Sigma$ is the neighborhood of shape E located at pixel $p \in \Sigma$. The DOM of pixel p in the image is represented as $\mu_I(p) = \mu_{I(\Sigma)}(p)$, or in the neighborhood as $\mu_{I(E+p)}(p)$. Since the DOM of a pixel is its normalized intensity, $\mu_{I(E+p)}(p) = \mu_I(p)$ for all $E + p$. Thus the DOM of pixel p can be written $\mu_I(p)$ in all contexts [8]. The gray level or intensity value $I(p)$ of a pixel p is often referred as its DOM. Since I has a specific, maximum intensity value, the gray level and the DOM are logically equivalent [8].

Definition 2 [8]: Path strength. Let $\rho = \{p = p_0, p_1, \dots, p_n = q\} \subset E \subset \Sigma$ be any path in E between two pixels, p and q . The strength of ρ , $s_E(\rho)$, is defined as the minimum DOM among the pixels on the path,

$$s_E(\rho) = \min_{0 \leq i \leq n} \{\mu_I(p_i)\} \quad (48)$$

In plain words “a path is as strong as its weakest link.” In any real digital image, there are a finite number of paths between any two pixels. To determine the degree of connectedness between two pixels all possible paths that link the pixels must be analyzed.

Definition 3 [8]: Degree of connectedness of two pixels. The Degree Of Connectedness (DOC) of p and q , $c_E(p, q)$, is defined as

$$c_E(p, q) \equiv \max_{\rho \subset E} \{s_E(\rho)\}, \quad (49)$$

for all paths $\rho \subset E$ between p and q . The degree of connectedness between two pixels is the strength of the strongest path between any two pixels. Connectivity can also be defined for a set of more than two pixels.

Definition 4 [8]: Degree of connectedness of a set. For any subset $E \subseteq \Sigma$, the DOC of E is defined as

$$c_E(E) = \min_{p, q \in E} \{c_E(p, q)\} \quad (50)$$

The DOC of a set of pixels is given by the smallest DOC in the set between two of its members. This leads to a definition of connectedness.

Definition 5 [8]: Connectedness of two pixels. Two pixels p and q are said to be connected in E if and only if there exists a path

$$\rho^c = \{p = p_0, p_1, \dots, p_n = q\} \quad (51)$$

where $p_i \in E$, for $0 \leq i \leq n$, such that for every pixel, $p_i \in \rho^c$,

$$\mu_I(p_i) \geq \min\{\mu_I(p), \mu_I(q)\} \quad (52)$$

Two pixels are connected in E if there is a path in $E \subseteq \Sigma$ between them such that each pixel on the path has a DOM not smaller than that of the lesser of the two end pixels.

Equivalently, p and q are said to be connected if [8]

$$c_E(p, q) = \min\{\mu_I(p), \mu_I(q)\} \quad (53)$$

In other words, two pixels are connected if the minimum DOM along the strongest path that connects them is attained by one of the pixels itself [8]. Paths that connect pixels p and q are shown in Figure 44. .

According to Definition 2 the “dash dot “ and “solid” line paths have the same strength although the “dash dot “ line parts have greater DOM than the “solid” line path. The path marked with the “dotted” line has the smallest path strength. The paths marked with “dash dot “ and “solid” line are the only paths that satisfy Definition 5. The

existence of one of these paths is sufficient to prove connectivity between p and q . The “dotted” line path does not satisfy the connectivity criteria because the strength of the right path is less than the DOM of two end pixels.

Figure 44 Paths connecting pixels p and q and their variations along the path

The following properties are used in derivation of α – connectivity [8]:

- 1) The following are all quantities in the closed interval $[0,1]$: the DOM, $\mu_I(p)$, of pixel p ; the strength, $s_E(\rho)$, of path $\rho(p,q)$ from p to q ; and the DOC, $c_E(p,q)$ of p and q .

- 2) DOC is reflexive: $c_E(p, p) = \mu_I(p)$.

- 3) DOC is symmetric: $c_E(p, q) = c_E(q, p)$.

- 4) DOC of two pixels cannot be greater than the smaller of the two's DOM:

$$c_E(p, q) \leq \min\{\mu_I(p), \mu_I(q)\}.$$

By definition, the degree of connectedness between any two pixels is computed by evaluating all the paths connecting these pixels. Traversing all possible paths is a computationally expensive task, especially for large subsets. The concept of α – connectedness is developed for improving computational efficiency of the DOM [8]. It also clarifies the relationship between DOC in grayscale images and connectivity in binary images.

Definition 6 [8]: α -connectedness. Pixel p is said to be α -connected to pixel q in E if the DOC between p and q in E is equal to the DOM of p in I . That is, p is α -connected to q if $c_E(p, q) = \mu_I(p)$.

In other words, p is α -connected to q , if p is on path ρ , the path of greatest strength among all the paths from p to q , and if among all the pixels in ρ , p has the smallest degree of membership [8].

An equivalent definition is [8]: Let B_t be the binary image created by thresholding I at gray level t . That is, let

$$B_t(p) = \begin{cases} 1, & \text{if } I(p) \geq t \\ 0, & \text{if } I(p) < t \end{cases} \quad (54)$$

Then, pixel p is α -connected to pixel q in E if p is connected to q in $B_{I(p)} \cap E$. That is, if p and q are in the same connected component in window E of binary image $B_{I(p)}$ created by thresholding I at the gray level, $I(p)$, then p is α -connected to pixel q in E .

This is illustrated in Figure 45. It can be seen from Figure 45-a that all pixels are connected to the pixel **r**. Similarly, Figure 45-b, c, and d show the connected pixels at levels of pixels **q**, **p**, and **s**, respectively.

It follows from the definition that the DOMs of pixels on a path, ρ , that α -connects p to q are such that $\mu_I(p_i) \geq \mu_I(p)$. If ρ does not α -connect the points but instead traverses a valley in DOM space between p and q , such that $s_E(\rho) = \mu_I(r)$, then r is α -connected to both p and q . Conversely, if r is α -connected to p along path $\rho(p, r)$ and to q along $\rho(r, q)$ then the strength of the concatenated path from p to q is $s_E(\rho(p, q)) = \mu_I(r)$.

Clearly, α -connectivity is reflexive since $c_E(p, p) = \mu_I(p)$ means that p is α -connected to itself. It is transitive since if p is α -connected to q that is α -connected to r , then p must be α -connected to r . It is not symmetric since p could be α -connected to q where $\mu_I(q) > \mu_I(p)$ that implies that q is not α -connected to p .

Figure 45 Connected pixels by thresholding

(a) The image from Figure 44 thresholded at the level of the pixel **r**. (b) The image from Figure 44 thresholded at the level of the pixel **q**. (c) The image from Figure 44 thresholded at the level of the pixel **p**. (d) The image from Figure 44 thresholded at the level of the pixel **s**.

By Definition 5 it is obvious that if one of p or q is α -connected to the other, then p and q are connected. On the contrary, if p and q are connected, then either p is α -connected to q or q is α -connected to p . This property suggests a way of finding the connected components in a set of pixels: if the α -connected pixels can be found, then the connectivity among those pixels is established.

Theorem 1 [8]: DOC of two pixels in a set.

Let $I : \Sigma \rightarrow \{\mathbb{Z}^+ \cup 0\}$ be an image, where Σ is an R row by C column contiguous subset of $\mathbb{Z}^2$ and $\mathbb{Z}^+$ is the set of positive integers. Let $K = \max\{I(p) \mid p \in \Sigma\}$. Let E be a finite subset of Σ . Let p and q be distinct pixels in E . Let $\{r_i\}_{i=0}^{M-1} \subset E$ be the set of pixels that are α -connected in E both to p and to q . The degree of connectedness between p and q is t/K , where t is the largest threshold that can be applied to I such that p and q are in the same connected component of $B_t \cap E$. That is,

$$c_E(p, q) = \max\{t \mid (p \in B_t \cap E) \text{ AND } (q \in B_t \cap E)\} \quad (55)$$

Moreover,

$$t = K \cdot \max\{\mu_I(r_i)\}_{i=0}^{M-1} \quad (56)$$

Proof [8]: Let $\rho_{pi}(r_i, p)$ and $\rho_{qi}(r_i, q)$ be paths over which r_i is α -connected to p and q .

Let $\rho_i(p, q)$ be the concatenation of $\rho_{pi}(r_i, p)$ and $\rho_{qi}(r_i, q)$. Then

$$s_E(\rho_i) = \mu_I(r_i) \quad (57)$$

Let $\hat{r} = \max_i \{r_i\}$. It follows from the definition of DOC that

$$c_E(p, q) \geq \mu_I(\hat{r}) \quad (58)$$

Moreover, by property 4 of the DOC $c_E(p, q) \leq \min\{\mu_I(p), \mu_I(q)\}$.

If there were a path, $\rho_x(p, q)$, such that

$$\min\{\mu_I(p), \mu_I(q)\} > s_E(\rho_x) > \mu_I(\hat{r}) \quad (59)$$

then there would be a pixel $r_x \in \rho_x$ such that $\mu_E(r_x) < \mu_E(s)$ for all other pixels $s \in \rho_x$.

This implies that r_x is α -connected to both p and q so that $r_x = r_i$ for some i . Therefore,

$$c_E(p, q) = \mu_I(\hat{r}) \quad (60)$$

Let $t_i = K \cdot \mu_I(r_i)$. Then any pixel $s \in \rho_i(p, q)$ must have DOM such that $K \cdot \mu_I(s) \geq t_i$.

Thus, B_{t_i} is a connected binary component that contains r_i, p and q . Let $\hat{t} = K \cdot \hat{r}$. Then

$\hat{t}$ is the largest number t for which B_t is a connected binary component that contains

both p and q .

The consequence of the result is that once α –connectedness is computed for a set of pixels, the degree of connectedness between the pixels can be evaluated without considering all the possible paths. Determination of connectedness among a group of pixels is reduced to the determination of α -connectedness.

Algorithm 1 computes $c_E(p, q)$, the DOC in E of p and q . The pixel values from I do not need to be normalized. They can be used directly through the substitution,

$\mu_I(p) = I(p)$. The efficiency of the algorithm can be increased if the first pass threshold level is set to $t_{\max} = \min\{\mu_I(p), \mu_I(q)\}$. If p and q are binary-connected at threshold $t_{\max}$, then they are connected in grayscale with DOC equal to $t_{\max}$. If p and q are not binary-connected at that threshold, then thresholds smaller than $t_{\max}$ must be checked.

Algorithm 1 [8] Computation of the DOC between two pixels.

Let $E = \{p_i\}_{i=0}^{n-1}$, such that $p = p_0$ and $q = p_{n-1}$, be a set of pixels from Σ .

Let U , be a set of pixels that is empty at first.

Let $p \sim q$ denote that p is connected to q in the binary sense.

for all $p_i \in E$, **such that** $\mu_I(p_i) \leq \min\{\mu_I(p), \mu_I(q)\}$ **threshold** E **at level** $\mu_I(p_i)$

if $p_i \sim p$ **and** $p_i \sim q$ **then**

put p_i **in** U

fi

rof

$c_E(p, q) = \max\{\mu_I(p_i) \mid p_i \in U\}$

The binary connectivity of p and q , such that $p, q \in E + p$, can be determined by thresholding $E + p$ at $\min\{I(p), I(q)\}$ and using a fast, morphological, binary connectedness algorithm such as the dilate-and-mask algorithm described in section 2.3.

Finding the DOC between a pixel and all the others in a neighborhood by the repeating use of Algorithm 1 is not very efficient. Algorithm 2 is developed to improve the efficiency of repeated use of Algorithm 1. One has to recall that for a given set $E = \{p_i\}_{i=0}^{n-1}$ of pixels the DOC, $c_E(p_0, p_i)$, between p_0 and another pixel, p_i , is the maximum threshold at which p_i is binary-connected to p_0 in E . Property 4 of the DOC states that $c_E(p, q) \leq \min\{\mu_I(p), \mu_I(q)\}$. Algorithm 2 finds the DOC between one pixel and all the others binary-connected to it at successive intensity levels. It can be modified to calculate all the connectedness relations by incorporating the connected component labeling algorithm in the procedure. Algorithm 2 can be executed in parallel because the operations at each threshold do not require the results from the others.

Algorithm 2 [8] Computation of the DOC between one pixel and the others in a set.

Let $E = \{p_i\}_{i=0}^{n-1}$ (where $p = p_0$) be a neighborhood of p in Σ .
Let $T[E; \mu_I(p_i)]$ be the set of pixels in E that have DOM greater than or equal to $\mu_I(p_i)$. Let $p \sim q$ denote that p is connected to q in the binary sense.

set $c_E(p, p_i) = 0$ for all $p_i \in E$.

set $c_E(p, p) = \mu_I(p)$.

for all $p_i \in E$ such that $\mu_I(p_i) \leq \mu_I(p)$ **threshold** E **at level**
 $\mu_I(p_i)$ **to form** $T[E; \mu_I(p_i)]$.

for each $p_j \in E$ **such that** $p_j \sim p$ **in** $T[E; \mu_I(p_i)]$

if $c_E(p, p_j) \leq \mu_I(p_i)$ **then**

$c_E(p, p_j) = \mu_I(p_i)$

fi

rof

rof

If p is taken as the center pixel in an $(2m+1) \times (2n+1)$ rectangular window ($m > 0, n > 0$), then a DOC map of the window indicates how the center pixel is connected to the others. Such a map is called a *connectivity map* or a *conn-map*. The conn-maps are created as follows.

Let p be a pixel location, let $E + p \subset \Sigma$ be a neighborhood at pixel p , let $I(p)$ be the value of image at pixel p and let $I(E + p)$ be the image of the entire neighborhood.

Similarly $M(p)$ and $M(E + p)$ are defined as values of the conn-map. In each window $E + p$, the intensity, $I(p)$, of pixel p is preserved by the conn-map; that is,

$M(p) = I(p)$. The intensities of other pixels, p_i , in the conn-map are determined by the α -connectedness relation between p and p_i . Let $A + p$ be the subset of pixels in $E + p$ that are α -connected to p , let $B + p$ be the subset of pixels in $E + p$ to which p is α -connected, and let $C + p$ be the subset of pixels in $E + p$ that are not connected to p . That is [8]

$$\begin{aligned} A + p &= \{p_i \in E + p \mid p_i \text{ is } \alpha\text{-connected to } p \text{ in } E + p\} \\ B + p &= \{q_j \in E + p \mid p \text{ is } \alpha\text{-connected to } q_j \text{ in } E + p\} \\ C + p &= \{r_k \in E + p \mid r_k \notin A \text{ AND } r_k \notin B\} \end{aligned} \quad (61)$$

Pixels, p_i , from set $A + p$ are such that $I(p_i) \leq I(p)$. They are either beside p on a constant plateau, below p on a ramp, or on the other side of a peak from p . Pixels, q_j , from set $B + p$ are such that $I(q_j) > I(p)$. They are either above p on a ramp, or on the other side of a peak from p . Pixels from set $C + p$ can have any intensity value, but form

brightness peaks or plateaus in I that are separated from p by a dark line or valley in $I(E + p)$.

The conn-map is formed by three rules [8]. Pixels $p_i \in A + p$, retain their original intensities, $I(p_i)$, in the conn-map. Pixels $q_j \in B + p$, take gray level $I(p)$ in the conn-map. Pixels $r_k \in C + p$, take the DOC value of r_k to p . The conn-map can be written [8]:

$$\begin{aligned} M(p_i) &= I(p_i) \text{ for all } p_i \in A + p, \\ M(q_j) &= I(p) \text{ for all } q_j \in B + p, \text{ and} \\ M(r_k) &= K \cdot c_E(r_k, p) \text{ for all } r_k \in C + p \end{aligned} \tag{62}$$

All these situations can be seen in Figure 46. Each 5×5 window is outlined by a large white square. Center pixel p is outlined by a small white square. Pixels that are α -connected to p are labeled with a white dot. Any pixel to which p is α -connected is labeled with a white square. Pixels in peaks disconnected from p are labeled with a white plus “+”. Pixels inside the windows that are not labeled lie on paths of constant gray level equal to that of p .

Based on connectivity map, two basic operations, $TL[I; E]$ - the topological median of light-connected pixels in I and $TD[I; E]$ - the topological median of dark-connected pixels in I , will be defined.

Definition 7 [8]: Simple topological median filters. Let $p \in \Sigma$ be a pixel location and let $I: \Sigma \rightarrow \{\mathbb{Z}^+ \cup 0\}$ be an image. Let $E \subset \Sigma$ be a set of contiguous pixel locations and let $E + p \subset \Sigma$ be a neighborhood of shape E at pixel p . The median of the light-connected

pixels in I , $TL[I; E]$, is defined by

$$TL[I; E](p) = \text{med}_{q \in E+p} \{c_{E+p}(p, q)\} \quad (63)$$

for all pixels $p \in \Sigma$ such that $E + p \subset \Sigma$. $c_{E+p}(p, q)$ is the DOC between pixels p and q within $E + p$.

The median of the dark-connected pixels in I , $TD[I; E]$, is defined by

$$TD[I; E](p) = 1 - \text{med}_{q \in \overline{E+p}} \{\overline{c_{E+p}}(p, q)\} \quad (64)$$

where $\overline{E + p}$ and $\bar{I}$ mean photographic negative: $\bar{I}(p) = 1 - I(p)$, $I(p) \in [0, 1]$.

TL operates directly on image I , whereas the domain of TD is the “negative” (in the photographic sense) image, $\bar{I}$. The sample probability density function of neighborhood $E + p$, $\tilde{P}_{I(E+p)}$, is defined to be the normalized histogram of the values of I on $E + p$ (unmodified values inside the white window in Figure 46-b to z). $\tilde{P}_{I(E+p)}$ will also be called a neighborhood probability density function (NPDF). Similarly, a connectivity map probability density function (CPDF), $\tilde{P}_{M(E+p)}$ is defined as the normalized histogram of the values of M on $E + p$ (the values inside the white window as shown in Figure 46-b to z). The conventional median filter, $\text{med}_{E+p}\{I\}$, is the median of $\tilde{P}_{I(E+p)}$ [8]. The basic topological median filter $TL[I; E]$ is the median of $\tilde{P}_{M(E+p)}$ [8]. In order

to understand the differences between $\text{med}_{E+p}\{I\}$ and $TL[I; E]$, the differences between

$\tilde{P}_{I(E+p)}$ and $\tilde{P}_{M(E+p)}$ are examined.

Figure 46 Conn-maps example

(a) The test image. (b) to (z) examples of conn-maps for the test image. Pixels α -connected to p are labelled with a dot. Pixels to which p is α -connected are labelled with a square. Pixels disconnected from p are labelled with “+”.

Pixels from $A + p$ have the same value in both I and M and cannot change the value of the median from the NPDF to the CPDF (pixels marked with white dots in Figure 46-b to z). Pixels from $B + p$ have all the same value, $I(p)$, in the CPDF (pixels marked with white squares in Figure 46-b to z). Since those values are all greater in the NPDF than in the CPDF, the CPDF median can be lower, but not less than $I(p)$. If p is in an intensity ramp and $E + p = \{A + p\} \cup \{B + p\}$, then two possibilities exist [8]:

- 1) If $\text{med}_{E+p}\{I\} \leq I(p)$ then $\text{med}_{E+p}\{M\} = \text{med}_{E+p}\{I\}$
- 2) If $\text{med}_{E+p}\{I\} > I(p)$ then $\text{med}_{E+p}\{M\} = I(p)$.

If $C + p$ is nonempty, then $E + p$ has some areas that are disconnected from p (pixels marked with “+” in Figure 46-b to z). At those pixels, $\{r_k\}$,

$M(p) = K \cdot c_E(p, r_k) < I(p)$. Since $\{r_k\} \not\subset A + p$, it must be true that

$M(p) = K \cdot c_E(p, r_k) < \min\{I(p), I(r_k)\}$. Any probability mass shifted from positions in $\tilde{P}_{I(E+p)}$ to new positions in $\tilde{P}_{M(E+p)}$ ends up lower in the density function than $I(p)$

[8]. If the disconnected areas are peaks that are darker than $I(p)$, then

$TL[I; E](p) \leq \text{med}_{E+p}\{I\}$. Moreover, if a disconnected peak is brighter than $I(p)$ its

probability mass is transferred to bins in $\tilde{P}_{M(E+p)}$ smaller than $I(p)$ [8]. The probability

mass from a disconnected peak in $E + p$ is shifted to below $I(p)$ unless that peak

already happens to have intensities less than $I(p)$. All this shows that for all images I ,

$TL[I; E](p) \leq I(p)$, that $TL[I; E]$ is antiextensive [8]. Antiextensivity is the property

of an operation that the output of the operation is contained in the input (the output values

of the filter must be contained in the range of the input values). Similarly, $TD[I; E]$ is extensive [8]. Extensivity is the property of an operation that the output of the operation contains the input (the output values of the filter must contain the input values). These two properties are important because they bound the output of the filters.

Proposition 1 [8]: Antiextensivity. Operator is TL antiextensive, i. e., $TL[I; E] \subseteq I$. That is, $TL[I; E](p) \leq I(p)$ for all $p \in \Sigma$.

Proof [8]: By property 4, for every pixel q in $E + p$, the DOC between p and q is such that $c_{E+p}(p, q) \leq \min\{\mu_I(p), \mu_I(q)\} \leq \mu_I(p)$.

Thus, $\text{med}_{q \in E+p} \{c_{E+p}(p, q)\} \leq \mu_I(p)$. Therefore, $TL[I; E](p) \leq I(p)$.

Proposition 2 [8]: Extensivity. Operator TD is extensive, i. e., $TD[I; E] \supseteq I$. That is, $TD[I; E](p) \geq I(p)$ for all $p \in \Sigma$.

Proof [8]: The DOC between a pixel p , and any pixel $q \in \overline{E + p}$, cannot be larger than $\min\{\mu_{\bar{I}}(p), \mu_{\bar{I}}(q)\}$. Moreover, $c_{\overline{E+p}}(p, q) \leq \min\{\mu_{\bar{I}}(p), \mu_{\bar{I}}(q)\} \leq \mu_{\bar{I}}(p) = 1 - \mu_I(p)$, for any pixel q in $\overline{E + p}$. It follows that $\mu_I(p) \leq 1 - \max_{q \in \overline{E+p}} \{c_{\overline{E+p}}(p, q)\}$. The following inequality always holds: $1 - \text{med}[c_{\overline{E+p}}(p, q)] \geq 1 - \max_{q \in \overline{E+p}} \{c_{\overline{E+p}}(p, q)\}$. Therefore, $TD[I; E](p) \geq I(p)$.

These two properties ensure that $I - TL[I; E]$ and $TD[I; E] - I$ are nonnegative. These difference images contain all the features from I that are not retained by $TL[I; E]$ and $TD[I; E]$ [8].

Through its antiextensivity, the $TL[I; E]$ filter is a quasi-erosion [8]. For some

images I and J , $TL[I; E] \cap TL[J; E] \neq TL[(I \cap J); E]$. On the average $TL[I; E]$ is darker than I . The skewing of the CPDF toward the dark end causes the topological median filter $TL[I; E]$ to produce a sharper response to a step-edge in noise than does a conventional median filter. As TL is a quasi-erosion, so TD is a quasi-dilation [8]. The similarities of TL and TD with erosion and dilation suggest the construction of composite operations: a quasi-opening, $TDL[I; E] = TD[TL[I; E]; E]$ and a quasi-closing, $TLD[I; E] = TL[TD[I; E]; E]$ [8]. Neither TDL nor TLD are increasing so TDL is not a true opening and TLD is not a true closing.

Definition 8 [8]: Composite topological median filters. The composite operators, TDL and TLD , are defined

$$TDL[I; E] = TD[TL[I; E]; E] \quad (65)$$

$$TLD[I; E] = TL[TD[I; E]; E] \quad (66)$$

These four topological median filters are sometimes called, “degree of connectedness filters” because they operate on the DOC map of each neighborhood in the image.

The responses of all four of the topological median filters (TMF) are identical to the response of a conventional median filter (CMF) to an isolated noise-free step-edge.

Figure 47 shows the responses of TMF and CMF on the noise-free and noisy step-edge.

It can be seen in Figure 47 that the responses differ in the presence of small features that are larger than impulses but less than 50% of the window area. All such features would be erased by a CMF, while TL and TD can preserve dark or light small features,

respectively. Figure 48 shows the effects of TMFs and CMF on both light and dark narrow bands of intensities. CMF removes the narrow feature while the appropriate TMF of the same size does not.

The type of TMF used is important. It can be seen that *TD* preserves light stripes and *TL* preserves dark stripes. *TLD*, *TDL*, and CMF remove the stripe of either polarity.

Figure 47 CMF and TMF response on a step edge

(a) Noise-free step edge. (b) 7x7 median filter response on the step edge. (c) 7x7 *TL* filter response on the step edge. (d) 7x7 *TD* filter response on the step edge. (e) 7x7 *TDL* filter response on the step edge. (f) 7x7 *TLD* filter response on the step edge. (g) Noisy step edge. (h) 7x7 median filter response on the noisy step edge. (i) 7x7 *TL* filter response on the noisy step edge. (j) 7x7 *TD* filter response on the noisy step edge. (k) 7x7 *TDL* filter response on the noisy step edge. (l) 7x7 *TLD* filter response on the noisy step edge.

If an image with bimodal noise is filtered with a *TL* or a *TD* then noise of one mode is reduced while the other tends to be preserved. The *TLD* and *TDL* filters are more symmetric, although they do introduce a bias as well. *TLD* and *TDL* are better edge preserving filters than the CMF under the condition that they are appropriately used. For an image with light features on a dark background the *TD* or *TLD* has to be used. For an

Figure 48 CMF and TMF response on a bar code test image.

image with dark features on a light background the *TL* or *TDL* has to be used. If TMFs are not used properly, the results may be worse than for a CMF in large transform windows. TMFs introduce a bias in mean image intensity. The bias is proportional to the standard deviation of the noise by a factor of about 0.35 [8]. *TD* and *TLD* introduce a positive bias, while *TL* and *TDL* introduce a negative one. Conventional median filters have better noise reduction than TMFs by a factor of approximately 1.25 [8], but TMFs are better in maintaining edge sharpness and location.

4. Analysis and Results

The topological median filters were implemented in C and run in MATLAB. The complete code is provided in the Appendix. The goal was to evaluate topological median filters on edge detection performance, accuracy of the edges, and image processing speed.

4.1 Edge Detection Performance

Topological median filter performance was compared with the conventional median filter. The reference step edge with detected reference edges is shown in Figure 49. Test images shown in Figure 50 are the image in Figure 49-a with added Gaussian noise. The noise was added by MATLAB function *imnoise()* that produces truncated Gaussian noise. The added noise has variance of 0.005, 0.01, 0.05, and 0.1. The images are 128×128

pixels. The first 128×64 pixels are black and the rest of the pixels are white. MATLAB built in edge detection methods with default parameters were used if not otherwise stated.

Figure 49 Step Edge Reference

(a) Step edge without noise. (b) The reference edge detected by the Sobel method. (c) The reference edge detected by the Canny method.

First, on the step edge image without noise the edge was detected by the Sobel and Canny methods. This is the reference edge. The test measures how many edge pixels can be detected on the place of the reference edge after application of the Sobel and Canny edge detectors on images from Figure 50. The edge detection without prefiltering, with prefiltering by 5×5 median, and with prefiltering by 5×5 *TDL* were performed on images from Figure 50. The results of edge detection by the Sobel method without prefiltering, the Sobel method with prefiltering by the 5×5 median filter, and the Sobel method with prefiltering by 5×5 *TDL* filter are shown in Figure 51, Figure 52, and Figure 53, respectively. The results of edge detection by the Canny method without prefiltering, the Canny method with prefiltering by the 5×5 median filter, and the Canny

Figure 50 Noisy step edge test images

(a) Step edge with added Gaussian noise ($m = 0, \sigma^2 = 0.005$). (b) Step edge with added Gaussian noise ($m = 0, \sigma^2 = 0.01$). (c) Step edge with added Gaussian noise ($m = 0, \sigma^2 = 0.05$). (d) Step edge with added Gaussian noise ($m = 0, \sigma^2 = 0.1$).

method with prefiltering by 5×5 *TDL* filter are shown in Figure 54, Figure 55, and Figure 56, respectively.

Figure 51 The edge detected by the Sobel method without prefiltering

(a) Step edge detected by the Sobel method without prefiltering ($\sigma^2 = 0.005$). (b) Step edge detected by the Sobel method without prefiltering ($\sigma^2 = 0.01$). (c) Step edge detected by the Sobel method without prefiltering ($\sigma^2 = 0.05$). (d) Step edge detected by the Sobel method without prefiltering ($\sigma^2 = 0.1$).

Figure 52 The edge detected by the Sobel method with median prefiltering

(a) Step edge detected by the Sobel method with 5×5 median prefiltering ($\sigma^2 = 0.005$). (b) Step edge detected by the Sobel method with 5×5 median prefiltering ($\sigma^2 = 0.01$). (c) Step edge detected by the Sobel method with 5×5 median prefiltering ($\sigma^2 = 0.05$). (d) Step edge detected by the Sobel method with 5×5 median prefiltering ($\sigma^2 = 0.1$).

Figure 53 The edge detected by the Sobel method with TDL prefiltering

(a) Step edge detected by the Sobel method with 5×5 TDL prefiltering ($\sigma^2 = 0.005$). (b) Step edge detected by the Sobel method with 5×5 TDL prefiltering ($\sigma^2 = 0.01$). (c) Step edge detected by the Sobel method with 5×5 TDL prefiltering ($\sigma^2 = 0.05$). (d) Step edge detected by the Sobel method with 5×5 TDL prefiltering ($\sigma^2 = 0.1$).

Figure 54 The edge detected by the Canny method without prefiltering

Step edge detected by the Canny method without prefiltering ($\sigma^2 = 0.005$). (b) Step edge detected by the Canny method without prefiltering ($\sigma^2 = 0.01$). (c) Step edge detected by the Canny method without prefiltering ($\sigma^2 = 0.05$). (d) Step edge detected by the Canny method without prefiltering ($\sigma^2 = 0.1$).

Figure 55 The edge detected by the Canny method with median prefiltering

(a) Step edge detected by the Canny method with 5×5 median prefiltering ($\sigma^2 = 0.005$). (b) Step edge detected by the Canny method with 5×5 median prefiltering ($\sigma^2 = 0.01$). (c) Step edge detected by the Canny method with 5×5 median prefiltering ($\sigma^2 = 0.05$). (d) Step edge detected by the Canny method with 5×5 median prefiltering ($\sigma^2 = 0.1$).

Figure 56 The edge detected by the Canny method with TDL prefiltering

(a) Step edge detected by the Canny method with 5×5 TDL prefiltering ($\sigma^2 = 0.005$). (b) Step edge detected by the Canny method with 5×5 TDL prefiltering ($\sigma^2 = 0.01$). (c) Step edge detected by the Canny method with 5×5 TDL prefiltering ($\sigma^2 = 0.05$). (d) Step edge detected by the Canny method with 5×5 TDL prefiltering ($\sigma^2 = 0.1$).

Table 1 Number of detected edge pixels

	σ^2 (variance)	without prefiltering		MED prefiltering		TDL prefiltering	
		# pixels on the proper edge position	Success Rate	# pixels on the proper edge position	Success Rate	# pixels on the proper edge position	Success Rate
Sobel	0.005	67	53.2%	63	50%	95	75.4%
	0.01	57	45.2%	63	50%	80	63.5%
	0.05	53	42%	69	54.8%	82	65.1%
	0.1	56	44.4%	58	46%	87	69.1%
Canny	0.005	70	55.6%	60	47.6%	104	82.5%
	0.01	77	61.1%	61	48.4%	117	92.9%
	0.05	67	53.2%	68	54%	113	89.7%
	0.1	69	54.8%	60	47.6%	111	88.1%

It can be seen from Figure 51, Figure 52, Figure 53, Figure 54, Figure 55, and Figure 56 that prefiltering reduces noise in the edge map images. The prefiltering also improves the visual appearance of the edge. It also can be seen that the median filter removes noise better than the topological median filter and that the edge on images with TDL prefiltering visually appears better than the same edge on the median prefiltered images. It also has to be noted that the Canny method reveals more noise than the Sobel method. Figure 56 shows the biasing of TLD filters towards small dark features on images (noise appears only on the “light” side of the original image).

The quantitative results of the comparison are shown in Table 1. It has to be noted that MATLAB does not pad the images prior to edge detection and the resulting images are reduced in size (126×126 pixels for this example). The maximum number of pixels found on the edge is 126. This value was obtained by the Sobel and Canny method and can be seen in Figure 49-b and Figure 49-c, respectively. The column “# pixels on the proper edge position” in Table 1 shows the number of pixels found on the reference edge after edge detection. The column “Success Rate” in Table 1 shows the percentage of successfully located edge pixels.

The results in Table 1 show that the TLD prefiltering increases the number of the edge pixels that match the reference edge about 20%, on average, for the Sobel edge detection method and about 30% on average for the Canny edge detection method in comparison to the edge maps without prefiltering. At the same time the median prefiltering does not show significant improvement in comparison to the edge detection methods without prefiltering.

The edge detection on the Lena image without prefiltering, with prefiltering by 3×3 median, and with prefiltering by 3×3 *TDL* is shown in Figure 57, Figure 58, and Figure 59, respectively.

Figure 57 Edges on Lena image without prefiltering

(a) The original Lena image. (b) Edges detected by the Sobel method. (c) Edges detected by the Marr-Hildreth method. (d) Edges detected by the Canny method.

Figure 58 Edges on Lena image with median prefiltering

(a) The original Lena image. (b) Edges detected by the Sobel method with 3×3 median prefiltering. (c) Edges detected by the Marr-Hildreth method with 3×3 median prefiltering. (d) Edges detected by the Canny method with 3×3 median prefiltering.

Figure 59 Edges on Lena image with TLD prefiltering

(a) The original Lena image. (b) Edges detected by the Sobel method with 3×3 TLD prefiltering. (c) Edges detected by the Marr-Hildreth method with 3×3 TLD prefiltering. (d) Edges detected by the Canny method with 3×3 TLD prefiltering.

The edge detection on a semiconductor image without prefiltering, with prefiltering by 3×3 median, and with prefiltering by 3×3 TDL is shown in Figure 60,

Figure 61, and Figure 62, respectively.

Figure 60 Edges on a semiconductor image without prefiltering

(a) A semiconductor image. (b) Edges detected by the Sobel method. (c) Edges detected by the Marr-Hildreth method. (d) Edges detected by the Canny method. (The semiconductor image courtesy of Alexander R. LaChance)

Figure 61 Edges on a semiconductor image with median prefiltering

(a) A semiconductor image. (b) Edges detected by the Sobel method with 3×3 median prefiltering. (c) Edges detected by the Marr-Hildreth method with 3×3 median prefiltering. (d) Edges detected by the Canny method with 3×3 median prefiltering. (The semiconductor image courtesy of Alexander R. LaChance)

Figure 62 Edges on a semiconductor image with TDL prefiltering

(a) A semiconductor image. (b) Edges detected by the Sobel method with 3×3 TLD prefiltering. (c) Edges detected by the Marr-Hildreth method with 3×3 TLD prefiltering. (d) Edges detected by the Canny method with 3×3 TLD prefiltering. (The semiconductor image courtesy of Alexander R. LaChance)

The edge detection on a high magnification SEM image without prefiltering, with prefiltering by 9×9 median, and with prefiltering by 9×9 TDL is shown in Figure 63,

Figure 64, and Figure 65, respectively.

Figure 63 Edges on a high magnification SEM image without prefiltering

(a) A high magnification SEM image. (b) Edges detected by the Sobel method. (c) Edges detected by the Marr-Hildreth method. (d) Edges detected by the Canny method. (The SEM image courtesy of Alexander R. LaChance)

Figure 64 Edges on a high magnification SEM image with median prefiltering

(a) A high magnification SEM image. (b) Edges detected by the Sobel method with 3×3 median prefiltering. (c) Edges detected by the Marr-Hildreth method with 3×3 median prefiltering. (d) Edges detected by the Canny method with 3×3 median prefiltering. (The SEM image courtesy of Alexander R. LaChance)

Figure 65 Edges on a high magnification SEM image with TDL prefiltering

(a) A high magnification SEM image. (b) Edges detected by the Sobel method with 3×3 TLD prefiltering. (c) Edges detected by the Marr-Hildreth method with 3×3 TLD prefiltering. (d) Edges detected by the Canny method with 3×3 TLD prefiltering. (The SEM image courtesy of Alexander R. LaChance)

It can be seen from edge-map images in Figure 63 that the edges are contaminated with noise caused by the material structure. The material structure edges are very visible on the Canny edge-map image Figure 63-d. The Canny method is very sensitive on edges and on images with visible material structure, such as SEM images, less sensitive edge detection methods give visually more appealing results. The edge-map images created by 9×9 median and 9×9 *TDL* prefiltering are shown in Figure 64 and Figure 65, respectively.

It can be seen from images in Figure 57, Figure 58, Figure 59, Figure 60, Figure 61, and Figure 62 that there is not significant difference between direct edge detection and prefiltering on images with a low level of noise. However, from Figure 63, Figure 64, and Figure 65 the benefits of prefiltering are visible. One can see from Figure 64 and Figure 65 that *TDL* prefiltering retains more details on the edge-map images than the same size median prefiltering.

Based on results presented in this section, topological median filters better preserve edges than conventional median filters of the same size.

4.2 *Edge Accuracy*

The edge position accuracy is very important for edge detection, especially if the results of the edge detection are used for feature size measuring or calibration. The achieved accuracy cannot be better than the accuracy of the edge detector. The Hough transform was used for the edge accuracy test. The edge positions on an image without noise are used as reference. The displacement of the edges is measured in respect to the reference. Images used in the test are shown in Figure 66. The edges were detected on the image from Figure 66-a using the Canny method. The Hough transform was used to locate the positions of already detected edges. Located positions were set as the reference positions.

The edges detected by the Canny method without prefiltering, with 5×5 median prefiltering, and with 5×5 *TDL* prefiltering are shown in Figure 67, Figure 68, and Figure 69, respectively. The Hough transform resolution was one pixel for ρ and 1 degree for θ . The Hough transform threshold was set to 50%. This means that all lines with intensity greater or equal to 50% of the strongest line are recognized. The edge locations of the images from Figure 67, Figure 68, and Figure 69 that were located by the Hough transform are shown in Table 2. The reference column ($\sigma^2 = 0$) contains all locations found by the Hough transform on the image from Figure 66-a. Each column,

“without prefiltering”, “MED prefiltering”, and “TDL prefiltering”, contains the edge locations for five values of variance – 0.005, 0.01, 0.02, 0.05, and 0.1.

Figure 66 Test images used for the edge accuracy test

(a) A bar test image. (b) Test image with added Gaussian noise ($m = 0, \sigma^2 = 0.005$). (c) Test image with added Gaussian noise ($m = 0, \sigma^2 = 0.01$). (d) Test image with added Gaussian noise ($m = 0, \sigma^2 = 0.02$). (e) Test image with added Gaussian noise ($m = 0, \sigma^2 = 0.05$). (f) Test image with added the Gaussian noise ($m = 0, \sigma^2 = 0.1$).

The locations that match the reference are shown in **boldface** font. The locations shown in grayed area are “diaphanous” edges for the Canny edge detector. These lines exist on the test image but cannot be detected by the edge detector because of its limitations. The “diaphanous” edges that are detected on noisy test images have unreliable positions that

can vary with noise. The last row shows the success rate in percents of locations that match the reference. The empty cells in the table mean that the line is not located.

Figure 67 Edges without prefiltering

(a) Reference edges. (b) Edges without prefiltering ($\sigma^2 = 0.005$). (c) Edges without prefiltering ($\sigma^2 = 0.01$). (d) Edges without prefiltering ($\sigma^2 = 0.02$). (e) Edges without prefiltering ($\sigma^2 = 0.05$). (f) Edges without prefiltering ($\sigma^2 = 0.1$).

It has to be noted that the goal of this test is not a quantity, but quality of the detected lines. The most successful method gives the highest number of the line positions that match the reference positions.

It can be seen from Table 2 that without prefiltering there are missing and miss located lines. The average success rate is around 60%. There are also two detected

“diaphanous” lines.

Figure 68 Edges with median prefiltering

(a) Reference edges. (b) Edges with 5×5 median prefiltering ($\sigma^2 = 0.005$). (c) Edges with 5×5 median prefiltering ($\sigma^2 = 0.01$). (d) Edges with 5×5 median prefiltering ($\sigma^2 = 0.02$). (e) Edges with 5×5 median prefiltering ($\sigma^2 = 0.05$). (f) Edges with 5×5 median prefiltering ($\sigma^2 = 0.1$).

The median prefiltering shows high missing rate especially at high variance level

($\sigma^2 = 0.1$). On the other variance levels there are only miss located lines. The average success rate is similar to the method without prefiltering and there are also “diaphanous”

lines. The TDL prefiltering shows only one missing line at high variance level ($\sigma^2 = 0.1$)

and a few miss located lines. The average success rate is around 90% and there is no “diaphanous” line detected by this method.

Figure 69 Edges with TDL prefiltering

(a) Reference edges. (b) Edges with 5×5 TDL prefiltering ($\sigma^2 = 0.005$). (c) Edges with 5×5 TDL prefiltering ($\sigma^2 = 0.01$). (d) Edges with 5×5 TDL prefiltering ($\sigma^2 = 0.02$). (e) Edges with 5×5 TDL prefiltering ($\sigma^2 = 0.05$). (f) Edges with 5×5 TDL prefiltering ($\sigma^2 = 0.1$).

These results are confirmation of the results from section 4.1 where similar results were obtained on the quality of line. The results show that use of the topological median filter improves edge detection quality. Topological median filters preserve not only edges, but also the edge positions much better than conventional median filters of the same size.

Table 2 Edges located by the Hough transform

		without prefiltering					MED prefiltering					TDL prefiltering				
σ^2	0	.005	.01	.02	.05	.1	.005	.01	.02	.05	.1	.005	.01	.02	.05	.1
E d g e l o c a t i o n s		5	5	5	5	5	7	7	7	7	7					
		16	16	16	16	16	14	14	14	14	14					
	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18
	22	22	22	22	21	21	21	22	21	21		21	22	22	22	
	24	25		25	24	24	24	24	24	24		24	24	24	24	24
	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28
	31	31	31	32	32	32	31	32	31	32		32	32	32	32	32
	36	36	36	36	36	36	36	36	36	36	36	36	36	36	36	36
	41		40	41	40	41	41	40	41	40		41	41	41	41	41
	45	45	46	45	46	46	45	46	46	46	45	45	45	45	45	45
	51	51	51	51	51	50	51	51	51	51	51	51	51	51	51	51
	56	56	56	57	57	56	56	56	56	57	56	56	56	56	56	56
	63	63	62	62	63	63	63	62	62	63		63	63	63	63	63
	70	71	70		71	70	71	70	70	71		70	70	70	70	70
	78	78	78	78	77	78	77	77	77	77	77	78	78	78	78	78
	87	88	87	87	88	87	87	87	87	87	87	87	87	87	87	87
	98	98	97	98	97	97	97	97	97	98		98	98	98	98	98
	109	110	110	110	110	110	110	110	110	110	110	109	109	109	109	109
Success [%]		68.8	62.5	62.5	37.5	62.5	68.8	56.3	62.5	50.0	43.8	87.5	93.8	93.8	93.8	87.5

4.3 *Processing Speed*

Processing speed is always important, because the processing speed determines the use and usefulness of the filter. The first implementation of the *TL* algorithm in MATLAB took almost 4 hours to process an 256×256 image on AMD Athlon™ XP2500+ processor running at 1.8 GHz with 512M of RAM. After improving the algorithm, by vectorizing data and using MATLAB profiler, the processing speed is dramatically improved. The profiling data for this MATLAB m-file implementation of the *TL* algorithm for an 512×512 image and 9×9 window is shown in Table 3. The first column in Table 3 shows the name of the called function, the second column shows the function file type, the third column shows the number of the function calls during profiling, and the last column shows the total time the function executed during profiling. It can be seen from Table 3 that the most of the time the algorithm spent calculating the DOC (Degree of Connectedness). The problem with MATLAB is that the type of function parameters is checked each time the function is called. One solution for this problem is to reduce the number of function calls. However the number of function calls is proportional to the image and window size. This approach has limitation in that all MATLAB built-in functions have to accept vectorized arguments. This is not the case with most of MATLAB image processing built-in functions. Another solution is to

eliminate the parameter checks completely by making C code that is integrated into the MATLAB m-file. This is called a mex-file. The C code can be compiled by typing *mex filename.c* on the MATLAB command line (for code *tmf.c*, as well as, *tl.m* and *td.m* please see Appendix).

Table 3 Processing speed of MATLAB implementation of *TL*

Filename	File Type	Calls	Total
tl_1	M-function	1	880.125 s
doc_2_mod_1	M-function	262144	856.108 s
bwlabel	M-function	262144	440.677 s
...s/images/private/checkinput	M-function	786433	279.084 s
...ges/images/private/bwlabel1	MEX-function	262144	269.630 s
...checkinput/check_attributes	M-subfunction	786433	146.408 s
imthreshold	M-function	262144	124.463 s
...te/checkinput/check_classes	M-subfunction	786433	110.183 s
strread	M-function	1048576	88.133 s
unique	M-function	262144	34.550 s
dataread	MEX-function	1048576	16.634 s
nargchk	M-function	1048576	14.144 s
...ivate/checkinput/init_table	M-subfunction	786433	13.038 s
med	M-function	262144	13.007 s
...ges/images/private/bwlabel2	MEX-function	262144	11.728 s
dmperm	M-function	262144	10.655 s
...ivate/checkinput/is_numeric	M-subfunction	262144	9.176 s
.../images/private/checknargin	M-function	524289	8.517 s
.../checkinput/check_nonsparse	M-subfunction	524289	7.914 s
...private/checkinput/check_2d	M-subfunction	524288	7.779 s
...ate/checkinput/check_scalar	M-subfunction	262144	5.427 s
...ivate/checkinput/check_real	M-subfunction	262144	3.248 s
im2double	M-function	1	0.031 s
profile	M-function	1	0 s
pad_image	M-function	1	0 s

This approach allows for some optimizations such as dilation with only one structuring element. The profiling results obtained for the mex-file are shown in Table 4. It has to be mentioned that the same 512x512 image and 9x9 window was used. The processing speed improvement is dramatic.

Table 4 Processing speed of MATLAB MEX implementation of *TL*

Filename	File Type	Calls	Total
tl	M-function	1	39.625 s
tmf	MEX-function	1	39.579 s
im2double	M-function	1	0.031 s
...te/checkinput/check_classes	M-subfunction	1	0.016 s
...s/images/private/checkinput	M-function	1	0.016 s
pad_image	M-function	1	0.015 s
profile	M-function	1	0 s
...ivate/checkinput/init_table	M-subfunction	1	0 s
...checkinput/check_attributes	M-subfunction	1	0 s
.../images/private/checknargin	M-function	1	0 s

This C code was run inside MATLAB. Probably a slight improvement could be expected if the C code runs without MATLAB overhead, but it was not tested.

5. Conclusions

Topological median filters were implemented in MATLAB as m-files and mex-files. They were applied on test images with and without added noise. It was shown that topological median filters preserve edges in images with high level of noise. Edge detection on the same noisy images was more successful with topological median prefiltering, even with the Canny method that was optimized for noisy step edges. It was also shown that the topological median filters do not displace the lines detected by the Hough transform. Topological median filters are an effective tool for prefiltering noisy images prior to edge detection. It was also noticeable that the Canny method reveals the surface structure on semiconductor images. In order to avoid the surface structure edges the Sobel operator is used, but it lacks the edge sensitivity of the Canny method. A method with a Sobel front end and Canny's nonmaximum suppression and hysteresis thresholding could be implemented and evaluated.

The drawback of using the topological median filter is that the algorithm has a relatively long processing time. The calculation of the DOC is the most time consuming part of the algorithm. C code implementation of the filter is acceptable for low volume off-line processing. Some other noniterative algorithms for finding connected

components have to be evaluated. Another possible path for algorithm improvement is parallel processing of more than one image at the same time. These algorithm improvements are ideas for further research.

References

- [1] Rafael C. Gonzales and Richard E. Woods, *Digital Image Processing, Second Edition*, Upper Saddle River, NJ: Prentice Hall, 2002.
- [2] E. R. Davies, *Machine Vision: Theory, Algorithms, Practicalities*, London UK: Academic, 1990.
- [3] Yaron C. Hecker and Ruud M. Bolle, "On Geometric Hashing and the Generalized Hough Transform", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 9, pp.1328-1338, September 1994.
- [4] Al Bovik, *Handbook of Image and Video Processing*, San Diego, CA: Academic Press 2000.
- [5] J.R. Parker, *Algorithms for Image Processing and Computer Vision*, John Wiley & Sons, Inc., 1997.
- [6] Hamid R. Tizhoosh, "Fast Fuzzy Edge Detection", *Fuzzy Information Processing Society, 2002. Proceedings NAFIPS, 2002 Annual Meeting of the North American*, pp. 239-242, 27-29 June 2002.
- [7] Zahid Hussain, *Digital Image Processing – Practical Applications of Parallel Processing Techniques*, Ellis Horwood Limited, 1991.
- [8] Hakan Güray Şenel, Richard Alan Peters,II, and Benoit Dawant, "Topological Median Filters", *IEEE Transactions on Image Processing*, vol. 11, no. 2, pp. 89-104, February 2002.
- [9] John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications, Third Edition*, Upper Saddle River, NJ: Prentice Hall, 1996.

- [10] E. R. Dougherty, *An Introduction to Morphological Image Processing*, D. C. O'Shea, Ed. Bellingham, WA: SPIE, 1992.
- [11] John Goutsias and Henk J. A. M. Heijmans, *Mathematical Morphology*, Amsterdam NL: IOS Press, 2000
- [12] John Canny, "A computational Approach to Edge Detection", *IEEE Transactions on Patern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, November 1986

Appendix

```
/*
=====
tmf.c
Implementation of topological median filter.
Takes a grayscale image and odd size filter window.
Returns topological median "light" of the input image
This is a MEX-file for MATLAB
Vladimir Martincevic 2004
=====
*/

#include "mex.h"
#include "matrix.h"

/*****
Implementation of quick sort algorithm. Takes a pointer to an array and
the array size and sort the array.
*****/
/
void q_sort(double* array, int size)
{
    if (size > 1)
    {
        int i;
        double temp;    /* temporary storage for swapping */
        int p = 0;      /* pivot */
        int fd = 1;      /* forward counter */
        int rv = size - 1 ; /* reverse counter */

        for (i = 1; i < size; i++) /* sort array */
        {
            if (array[fd] < array[p])
            {
                temp = array[p];
                array[p++] = array[fd];
            }
        }
    }
}
```

```

        array[fd++] = temp;
    }
    else
    {
        temp = array[rv];
        array[rv--] = array[fd];
        array[fd] = temp;
    }
}
q_sort(array, p);    /* repeat for the first part of the array */
q_sort(array + p + 1, size - (p + 1)); /* repeat for the second
part of the array */
}

/*****
Implementation of median filter. Takes data window of the size w_size
and returns the median value
*****/
/
double median_filter(double* window, int w_size)
{
    int i;
    double median;
    double* temp_image = (double*) mxCalloc(w_size, sizeof(double));

    for (i = 0; i < w_size; i++)
        temp_image[i] = window[i];

    q_sort (temp_image, w_size);
    median = temp_image[(w_size / 2)];
    mxFree(temp_image);
    return median;
}

/*****
Implementation of get_window. Takes pointer to window and image image,
number of rows and columns of the image, start indexes of the window,
and size of the window.
*****/
/
void get_window (double* window, double* image, int rows, int cols,
                int r_index, int c_index, int h_size, int v_size)
{
    int i, j;

    for (i = 0; i < h_size; i++)
    {
        for (j = 0; j < v_size; j++)
            window[j + i * v_size] = image[(r_index + j) + (c_index + i) *
rows];
    }
}

```

```

/*****
Implementation of find_thresholds. Takes pointer to the list of
thresholds
and length to the list, image, image size and threshold value. th_list
contains the values for thresholding.
*****/
/
void find_thresholds(double* th_list, int* list_size, double* image,
                    int size, double threshold)
{
    int i, j;
    *list_size = 0;

    for (i = 0; i < size; i++)
    {
        if (image[i] <= threshold)
        {
            if (*list_size == 0)
                th_list[(*list_size)++] = image[i];
            else
            {
                for (j = 0; j < *list_size; j++)
                {
                    if (th_list[j] == image[i])
                        break;
                }

                if (j == *list_size)
                    th_list[(*list_size)++] = image[i];
            }
        }
    }
}

/*****
Implementation of im_threshold. Takes pointer to the thresholded image
and image, and the image size and threshold level. thresholded_image
contains binary image.
*****/
/
void im_threshold(int* thresholded_image, double* image,
                 int image_size, double threshold)
{
    int i;

    for(i = 0; i < image_size; i++)
    {
        if (image[i] >= threshold)
            thresholded_image[i] = 1;
        else
            thresholded_image[i] = 0;
    }
}

```



```

/*****
Implementation of dilate. Takes pointer to the dilation result
and image, and rows and columns of the image. d_result contains dilated
image by 8-connected structuring element.
*****/
/
void dilate (int* d_result, int* image, int rows, int cols)
{
    int i, j;

    for (i = 0; i < cols; i++)
    {
        for (j = 0; j < rows; j++)
        {
            if (image[j + i * rows] == 1)
            {
                d_result[j + i * rows] = 1;
                if (i != 0 && j != 0 && i != cols - 1 && j != rows - 1)
                {
                    d_result[(j-1) + (i-1) * rows] = 1;
                    d_result[j + (i-1) * rows] = 1;
                    d_result[(j+1) + (i-1) * rows] = 1;
                    d_result[(j-1) + i * rows] = 1;
                    d_result[(j+1) + i * rows] = 1;
                    d_result[(j-1) + (i+1) * rows] = 1;
                    d_result[j + (i+1) * rows] = 1;
                    d_result[(j+1) + (i+1) * rows] = 1;
                }
            }
            else
            {
                if (i == 0)
                {
                    d_result[j + (i+1) * rows] = 1;

                    if (j != 0)
                    {
                        d_result[(j-1) + i * rows] = 1;
                        d_result[(j-1) + (i+1) * rows] = 1;
                    }

                    if (j != rows - 1)
                    {
                        d_result[(j+1) + i * rows] = 1;
                        d_result[(j+1) + (i+1) * rows] = 1;
                    }
                }
            }
            else if (j == 0)
            {
                d_result[(j+1) + i * rows] = 1;
                if (i != 0)

```

```

        {
            d_result[j + (i-1) * rows] = 1;
            d_result[(j+1) + (i-1) * rows] = 1;
        }

        if (i != cols - 1)
        {
            d_result[j + (i+1) * rows] = 1;
            d_result[(j+1) + (i+1) * rows] = 1;
        }
    }

    else if (i == cols - 1)
    {
        d_result[j + (i-1) * rows] = 1;

        if (j != 0)
        {
            d_result[(j-1) + (i-1) * rows] = 1;
            d_result[(j-1) + i * rows] = 1;
        }

        if (j != rows - 1)
        {
            d_result[(j+1) + (i-1) * rows] = 1;
            d_result[(j+1) + i * rows] = 1;
        }
    }

    else /* j == rows - 1 */
    {
        d_result[(j-1) + i * rows] = 1;

        if (i != 0)
        {
            d_result[(j-1) + (i-1) * rows] = 1;
            d_result[j + (i-1) * rows] = 1;
        }

        if (i != cols - 1)
        {
            d_result[(j-1) + (i+1) * rows] = 1;
            d_result[j + (i+1) * rows] = 1;
        }
    }
}
}
}
}
}
}
}

/*****

```

```

Implementation of mask. Takes pointer to the image, and mask, and
images
size value. image contains the result of masking.
*****
/
void mask (int* image, int* mask, int size)
{
    int i;

    for (i = 0; i < size; i++)
        image[i] = image[i] & mask[i];
}

/*****
Implementation of equal_images. Takes pointer to the image1 and image2,
and images size value. Returns 1 if images are equal, 0 otherwise.
*****
/
int equal_images (int* image1, int* image2, int size)
{
    int i;

    for (i = 0; i < size; i++)
    {
        if (image1[i] != image2[i])
            return 0;
    }
    return 1;
}

/*****
Implementation of copy_images. Takes pointer to the image1 and image2,
and images size value. Returns 1 if images are equal, 0 otherwise.
*****
/
void copy_images (int* image1, int* image2, int size)
{
    int i;

    for (i = 0; i < size; i++)
        image1[i] = image2[i];
}

/*****
Implementation of conn_pixels. Takes pointer to the connected pixels
and image, and rows and columns of the image. conn contains binary image
of pixels 8-connected to the center pixel.
*****
/
void conn_pixels (int* conn, int* image, int rows, int cols)
{
    int size = rows * cols;
    int* temp = (int*) malloc(size, sizeof(int));

```

```

while (1)
{
    dilate (temp, conn, rows, cols);
    mask (temp, image, size);
    if (equal_images(conn, temp, size))
        break;

    copy_images (conn, temp, size);
}

mxFree(temp);
}

/*****
Implementation of doc (degree of connectedness). Takes pointer to ce,
and window, and number of window rows and columns.
*****/
/
void doc (double* ce, double* window, int rows, int cols)
{
    int size = rows * cols;
    int i, j, list_size;
    double p = window[size / 2];
    double* th_list = (double*) mxMalloc(size, sizeof(double));
    int* bw_image = (int*) mxMalloc(size, sizeof(int));
    ce[(rows * cols)/2] = p;

    find_thresholds(th_list, &list_size, window, size, p);

    for (i = 0; i < list_size; i++)
    {
        int* conn = (int*) mxMalloc(size, sizeof(int));
        conn[size / 2] = 1;
        im_threshold(bw_image, window, size, th_list[i]);
        conn_pixels (conn, bw_image, rows, cols);

        for (j = 0; j < size; j++)
        {
            if (conn[j] == 1 && ce[j] <= th_list[i])
                ce[j] = th_list[i];
        }
        mxFree(conn);
    }
    mxFree(th_list);
    mxFree(bw_image);
}

/*****
MEX gateway function
*****/
/
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])

```

```

{
    double *image, *w, *result, *arg2;
    int i, j, mrows, ncols, w_rows, w_cols;

    if (nrhs != 2)
    {
        mexErrMsgTxt("Incorrect number of arguments.");
    }

    else if (nlhs > 1)
    {
        mexErrMsgTxt("Too many output arguments.");
    }

    mrows = mxGetM(prhs[0]);
    ncols = mxGetN(prhs[0]);
    image = mxGetPr(prhs[0]);

    if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]))
    {
        mexErrMsgTxt("Input must be grayscale image in the range
[0,1].");
    }

    if (!mxIsDouble(prhs[1]) || mxIsComplex(prhs[1]) ||
        !(mxGetM(prhs[1]) == 1 && mxGetN(prhs[1]) == 2))
    {
        mexErrMsgTxt("Secon argument is invalid.");
    }

    arg2 = mxGetPr(prhs[1]);
    w_rows = arg2[0];
    w_cols = arg2[1];

    if (!(w_rows % 2 && w_cols % 2))
    {
        mexErrMsgTxt("the window size must be odd.");
    }

    plhs[0] = mxCreateDoubleMatrix(mrows - w_rows + 1, ncols - w_cols +
1, mxREAL);

    result = mxGetPr(plhs[0]);
    w = (double*) mxCalloc(w_rows * w_cols, sizeof(double));

    for (i = 0; i < mxGetN(plhs[0]); i++)
    {
        for (j = 0; j < mxGetM(plhs[0]); j++)
        {
            double* ce = (double*) mxCalloc(w_rows *
w_cols, sizeof(double));
            get_window (w, image, mrows, ncols, j, i, w_rows, w_cols);
            doc (ce, w, w_rows, w_cols);
        }
    }
}

```

```

        result[j + i * mxGetN(plhs[0])] = median_filter(ce, w_rows *
w_cols);
        mxFree(ce);
    }
}
mxFree(w);
}

```

```

function J = TL(I, W)
% TL operator (topological median filter "light")
% I - image to be filtered
% W - 1x2 matrix that specified filter window size.
% W must contain positive odd numbers

IM = im2double(pad_image(I,W));
J = tmf(IM,W);

```

```

function J = td(I, W)
% TD operator (topological median filter "dark")
% I - image to be filtered
% W - 1x2 matrix that specified filter window size.
% W must contain positive odd numbers
IM = imcomplement(im2double(pad_image(I,W)));
[m,n] = size(I);
J = ones(m,n);
J = J - tmf(IM,W);

```