

Polygon detection in images

by

Johan Magnus van Niekerk

Submitted in partial fulfilment of the requirements for the degree

Magister Scientiae

In the Department of Statistics

In the Faculty of Natural and Agricultural Sciences

University of Pretoria

Pretoria

February 2015

I, Johan Magnus van Niekerk declare that the dissertation, which I hereby submit for the degree Magister Scientiae in Applied Statistics at the University of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.

SIGNATURE:

DATE: 20 February 2015

Summary

A combination of image processing algorithms can be used to detect and extract certain objects from images. Image noise may obscure the objects in the images and must first be removed using image denoising and smoothing methods before the extraction of image objects can efficiently be attempted. Contour detection algorithms can be used to identify and extract objects from denoised images. The type of image, noise and noise intensity will determine the effectiveness of the denoising and smoothing methods. The combination of the aforementioned will also determine the efficiency of the contour detection algorithms. In this dissertation we give some background on basic mathematical morphology techniques and we see how those techniques are used in various image smoothers and contour detection techniques. We develop our own smoother using some ideas from other well-known smoothers and compare the performance of the different smoothing methods. Various contour detection algorithms are investigated and we put the combination of the smoothing methods and the contour detection algorithms to the test by comparing how efficiently they detect and extract objects from images under various conditions.

Acknowledgements

First and foremost, I would like to thank Dr. Inger Fabris-Rotelli for her patience, motivation, enthusiasm, and immense knowledge and insight while working on this dissertation. She made every mountain I faced seem possible to overcome.

I would also like to thank the Department of Statistics and the NRF Competitive Support for Unrated Researchers Grant 90315 for giving me the opportunity to attend and present at various conferences across the country and complete my research.

Last but not least, thank you to my family and friends who stood by me while working on this dissertation even when I couldn't always give them the attention they deserved.

Contents

1	Introduction	6
2	Mathematical Morphology	7
2.1	Introduction	7
2.2	Background Notions	7
2.2.1	From Continuous to Discrete Space	7
2.2.2	Image Types	7
2.3	Erosion and Dilation	18
2.3.1	Structuring Elements	18
2.3.2	Erosion	18
2.3.3	Dilation	21
2.3.4	Image Transformation Properties	21
2.3.5	Morphological Opening	25
2.3.6	Morphological Closing	26
2.4	Conclusion	28
3	Algorithms	30
3.1	Introduction	30
3.2	Noise and Smoothers	30
3.2.1	Median Filter	32
3.2.2	Robust Statistics Based Algorithm	35
3.2.3	LULU Smoothing	38
3.2.4	Connected Median Image Smoother	43
3.3	Algorithm Implementation	48
3.4	Contour Detection	56
3.4.1	A Border Following Method	56
3.4.2	Contour Detection Using Watersheds	60
3.4.3	Active Contours	66
3.5	Conclusion	67
4	Applications	69
4.1	Introduction	69
4.2	Comparison	69
4.3	Conclusion	99
5	Conclusion	100

CONTENTS

5

Appendix

106

Chapter 1

Introduction

In 1964 Georges Matheron and Jean Serra performed the first mathematical morphology image study on a mineral image sample obtained from the mine of La Mourière [38]. Later Euclidean morphology in Euclidean spaces \mathbb{R}^n lead to the creation of mathematical morphology in discrete spaces \mathbb{Z}^n which is done by sampling from \mathbb{R}^n and enables us to analyse, transform and perform calculations using image data. When we think of an image we typically think of a colour image, but any amount of data can be associated with an image over the same image definition domain. This data, may it be pixel intensity or multisensor data, can be analysed and transformed into meaningful information by using the techniques and algorithms developed in the field of mathematical morphology. The task of automatically detecting and extracting man-made objects from images is one way to extract meaningful information from images and it's a challenging one, but one that is becoming extremely important as the gathering of visual data becomes more prominent. Most man-made objects like buildings, roads and traffic signs can easily be identified by the human eye because of their distinct shape which resembles different types of polygons. This lead us to the research presented in this dissertation. We investigate different strategies that will enable us to accurately identify and extract man made objects from images. Our research will enable future researchers to use this research as reference to determine what combination of algorithms will give the best results given their unique circumstances. We provide 160 different results under various image circumstances from which object are extracted and give a thorough comparison of some algorithms that may be used to this effect. A new smoothing algorithm is also proposed and a performance comparison is done using sample images. We give an analysis of our findings and propose some possible solutions for extracting man-made objects from images.

Chapter 2 is meant as a literature review of the field of mathematical morphology and the fundamental techniques available to us. In chapter 3 we look at different image smoothing techniques and also introduce a smoothing technique of our own. Different contour detection algorithms are also investigated in this chapter. We combine the smoothing techniques and contour detection algorithms in chapter 4 and compare the end products in terms of ability to detect and extract objects from images. In Chapter 5 we present our conclusion and some ideas for future work.

Chapter 2

Mathematical Morphology

2.1 Introduction

The history of the field of mathematical morphology is given in [38] by the creators themselves. We will now provide some background on the field of mathematical morphology and the core principals that support most of the techniques and methods used in the field. Our main focus will be on greyscale images formally introduced at the end of Section 2.2.2 and the theory surrounding it. In Chapter 3 we will use this theory to explain more complex image transformations and algorithms in order to reach our goal of accurately extracting objects from images in Chapter 4. Concepts such as the transformation of images using structuring elements in order to obtain information about neighbouring pixels, image dilation and erosion, opening and closing of images and thresholding of images are discussed in this chapter. These concepts enables were used to develop and program the algorithm of our own image smoother introduced in Section 3.2.4.

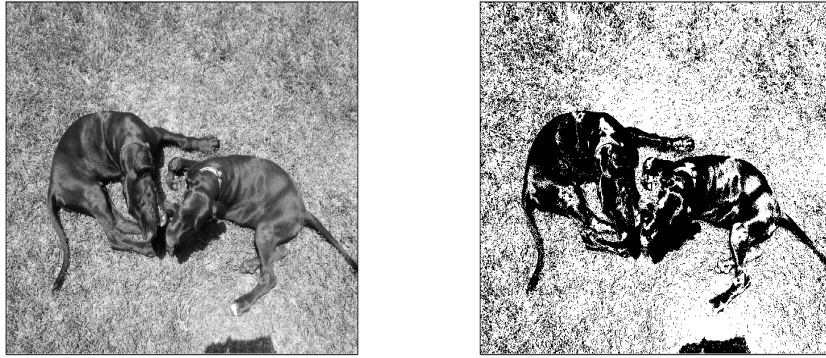
2.2 Background Notions

2.2.1 From Continuous to Discrete Space

Mathematical morphology was originally created for Euclidean spaces \mathbb{R}^n called Euclidean morphology. It was later adapted to discrete spaces \mathbb{Z}^n which is done by sampling from \mathbb{R}^n . This technique is called spatial quantisation or digitisation. A digitisation network is used to capture the sampled data in the discrete space using evenly distributed network points situated at the nodes of either a square or triangular mesh which makes up a tessellation of space. Each mesh point, also called a pixel (picture element), is used as a sampling window in order to assign a numerical value to each pixel and finally complete the digitisation process. The kind of mesh used, together with the position of the sampling grid, may cause the geometrical and topological properties of the discretised version to differ from that of the continuous version. [58]

2.2.2 Image Types

Digitisation can result in different types of discrete images. Monochannel images include grey tone and binary images and are called monochannel because each pixel assumes only one numerical value. Multichannel images have more than one numerical value associated with every pixel. [58]



(a) Original grey tone image. (b) The resultant image after binarisation.

Figure 2.1: The mapping of the grey tone image definition domain D_f onto the binary image definition domain.

Binary Image

Let D_f , a subset of \mathbb{Z}^2 , be the whole discrete definition domain of f . A binary image is the mapping of D_f onto values of either 1 or 0 [58]:

$$f : D_f \rightarrow \{0, 1\}$$

or

$$\forall \mathbf{x} \in D_f, f(\mathbf{x}) \in \{0, 1\}.$$

When presenting binary images we discriminate between the background and foreground. If the background, also called the support, is presented in black, then the foreground is white and visa versa. The morphological name for an object in a binary image is a set. An original grey tone image and the image resulting from mapping the original grey tone image onto the binary space are shown in Figure 2.1.

Grey tone image

Let D_f , a subset of \mathbb{Z}^2 , be the definition domain of f . A grey tone image is the mapping of D_f onto a bounded set of non-negative integers \mathbb{N}_0 :

$$f : D_f \rightarrow \{0, 1, \dots, t_{max}\}$$

or

$$\forall \mathbf{x} \in D_f, f(\mathbf{x}) \in \{0, 1, \dots, t_{max}\}.$$

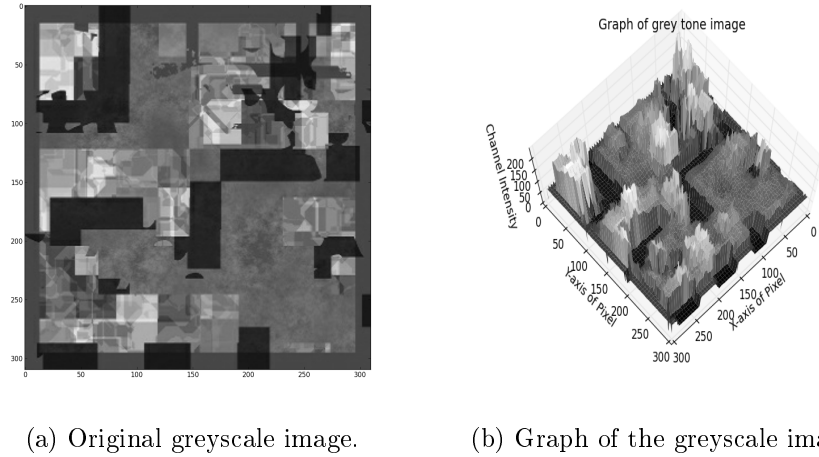


Figure 2.2: Greyscale image with the matching graph of that image

A grey tone image has a numerical value associated with each pixel of the grey tone image which assumes values defined by a finite set of non-negative integers and not limited to the set $\{0, 1\}$ like binary images. Pixels associated with numerical values of 0 are usually depicted as black and pixel with values t_{\max} are usually considered to be white. Usually for greyscale images $t_{\max} = 255$ which is an 8 bit representation.

A greyscale image f can also be considered in terms of its graph G , also called an intensity surface [58], which consists of the numerical value that each pixel is associated with at each coordinate \mathbf{x} in the definition domain of f ,

$$G(f) = \{(\mathbf{x}, t) \in D_f \times \mathbb{N}_0 | t = f(\mathbf{x})\}.$$

We can also consider the subgraph SG of an image f which consists of all the points below $G(f)$ where t is non-negative,

$$SG(f) = \{(\mathbf{x}, t) \in D_f \times \mathbb{N}_0 | 0 \leq t \leq f(\mathbf{x})\}.$$

Greyscale images in n dimensions have subgraphs in $n + 1$ dimensions. A greyscale image can be represented by its intensity surface as illustrated in Figure 2.2b. This surface is made of the greyscale values at each pixel which we can plot on a third axis to create a three-dimensional topographical image. A two dimensional subgraph can be obtained by keeping one of the horizontal axis constant as seen in Figure 2.3.

Multichannel image

An array of monochannel images defined over the same definition domain form a multichannel image which is denoted by \mathbf{f} ,

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})).$$

Instead of having one numerical value associated with each pixel \mathbf{x} , we have an array of values. The frequencies of the channel intensity levels of an image containing three channels (red, green

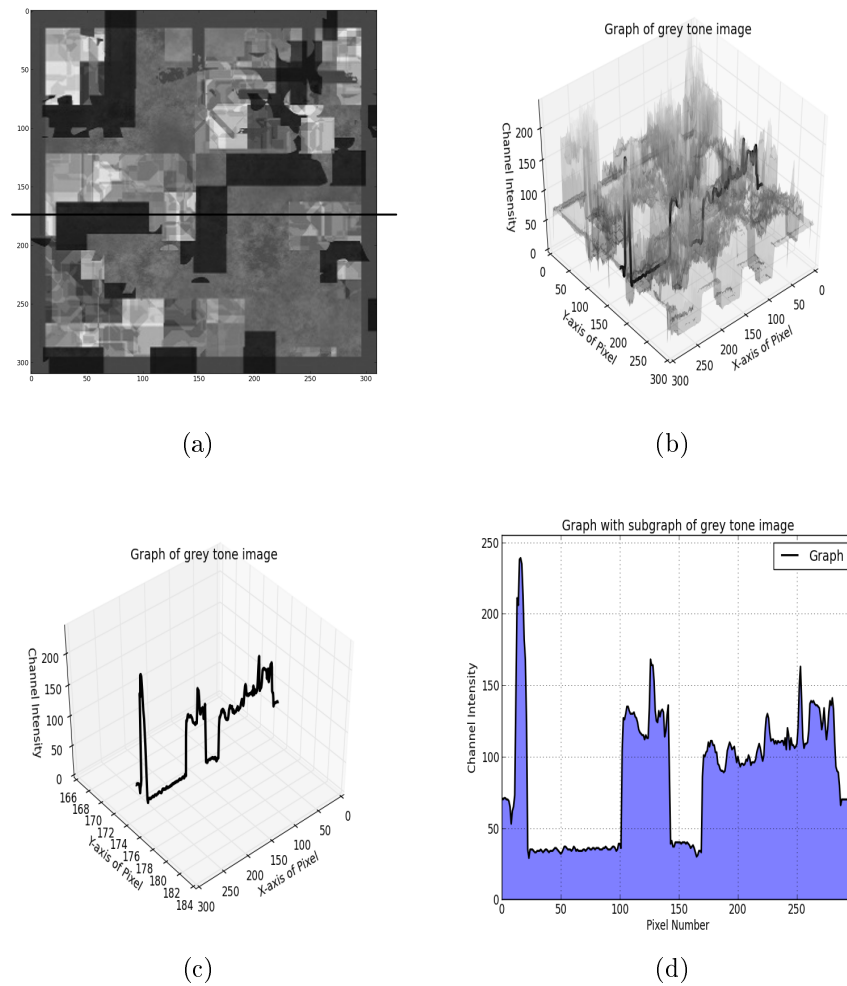


Figure 2.3: a.) Original greyscale image with cross section at row 175 represented as a line. b.) Graph of the greyscale image showing the cross section at row 175 in three dimensions. c.) The graph of the cross section taken at row 175 in three dimensions. d.) The subgraph of the greyscale image at row 175.

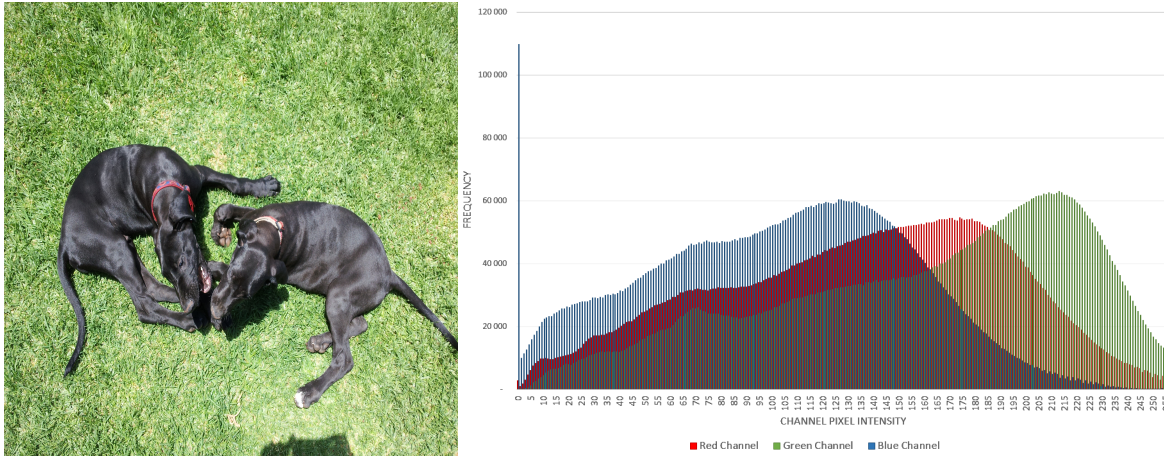


Figure 2.4: An example of a multichannel image together with the frequency levels of pixel intensity of the three separate RGB colour channels.

and blue) can be seen in Figure 2.4. Typically colour images consist of an array of three values for each pixel - this is called the red-green-blue (RGB) colour model and the frequencies of each array usually range from 0 to 255. The values in the array may contain any kind of information about that particular pixel. Examples of different kinds of information stored include values obtained using different spectral or wavelength intervals which are called multispectral images. Another example is multisensor images which are obtained by using different sensors to acquire information about the same image, for instance LIDAR (LIght Detection And Ranging). [53] A wide range of different remote sensing techniques can be found in [46]. Images defined over the same definition domain but collected at different times are called multitemporal images and are an extension of multichannel images. We will concentrate on morphology for monochannel images which can be generally extended to multichannel images. [58]

Image to image transformation

Image to image transformations take some image f defined on D_f as input and outputs some image $\Psi(f)$ also defined on D_f . The most trivial image to image transformation is the identity transformation which is denoted by id :

$$\forall f, id(f) = f.$$

Images which are not modified by a given image transformation form part of the domain of invariance of that image transformation. Iterating an image transformation, Ψ , n times is represented by $\Psi^{(n)}$:

$$\Psi^{(n)} = \Psi^{(n-1)}\Psi,$$

with $\Psi^{(0)} = id$.

We distinguish between point image transformations, which transform each pixel individually and neighborhood image transformations, which may use other pixels in the vicinity as input for a certain pixel's transformation. [58]

Point image transformation

A point image transformation of a certain pixel \mathbf{x} only depends on the numerical value(s) of that specific pixel. For the case of a multichannel image \mathbf{f} , consisting of an array of numerical values associated with pixel \mathbf{x} , the point image transformation will only use the values in the set $\{f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_p(\mathbf{x})\}$ as input for the transformation. Thus the general expression is obtained:

$$[\Psi(\mathbf{f})](\mathbf{x}) = \Psi[\mathbf{f}(\mathbf{x})].$$

The threshold operator T , also known as level slicing, is an example of a point image transformation. The transformation T assigns a value of 1 to all pixels \mathbf{x} , with associated numerical value inside a given interval $[i, j]$, in an image f and a value of 0 to those with values outside that interval:

$$[T_{[i,j]}(f)](\mathbf{x}) = \begin{cases} 1, & \text{if } i \leq f(\mathbf{x}) \leq j \\ 0, & \text{otherwise.} \end{cases}$$

This operator will therefore transform any greyscale image into a binary image. A special case of the threshold operator is the cross-section function which is the threshold operator with i and j as the level of the cross-section.

$$[CS_t(f)](\mathbf{x}) = [T_{[t,t_{\max}]}(f)](\mathbf{x}) = \begin{cases} 1, & \text{if } t \leq f(\mathbf{x}) \leq t_{\max} \\ 0, & \text{otherwise.} \end{cases}$$

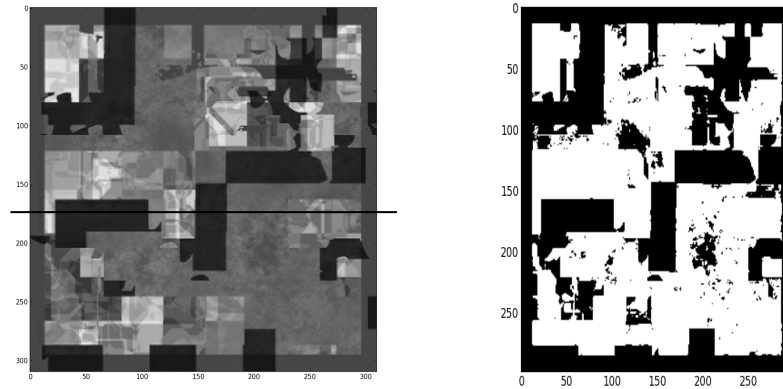
The ordering relationship between consecutive levels of cross-sections of some image f is as follows:

$$CS_{t_{\max}}(f) \subseteq CS_{t_{\max}-1}(f) \subseteq \dots \subseteq CS_1(f) \subseteq CS_0(f).$$

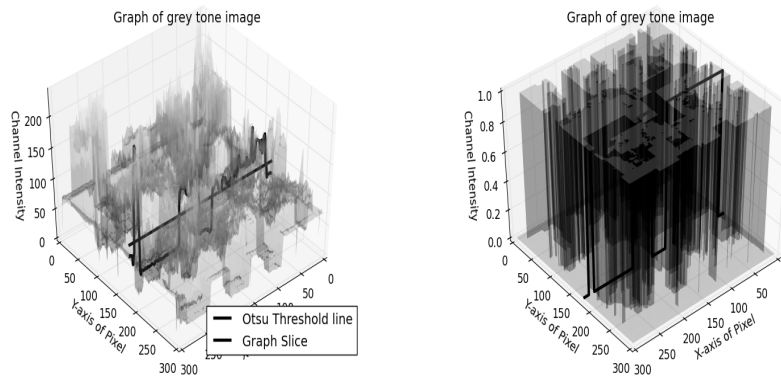
Any greyscale image can be decomposed into the sum of its cross-sections sets $CS_t(f)$, excluding $t = 0$, which is called threshold decomposition or the threshold superposition principle:

$$f = \sum_{t=1}^{t_{\max}} CS_t(f).$$

We can also say that the value which f assumes at any point \mathbf{x} is the largest value of t for which the cross-section $CS_t(f)$ includes \mathbf{x} . [58] A popular example of a thresholding technique for converting greyscale images to binary images is Otsu's method [42]. This method maximizes the separability of the resultant gray classes as a discriminant criterion. This plainly means that this method iterates through all possible threshold intensity values and select the one that minimise the inter-class variance. Thresholding on its own has been, and still is, the topic of much research. Some thresholding techniques take into account local information like illumination levels [11], making it a neighbourhood image transformation, and others use wavelet coefficients to distinguish between important image features and noise [13]. Figure 2.5 illustrates the effect of Otsu's method on the greyscale image and the graph or intensity surface of an image. The normal image is shown in a.) together with the resultant image after Otsu thresholding. The intensity surface plots are given in b.) showing the normal image and the resultant image after Otsu thresholding. In both cases consistent lines are shown to indicate what happens to a specific cross section on the image when Otsu thresholding is applied. Figure 2.6 shows what happens to a two dimensional slice of a graph when Otsu's method is applied. Figure 2.7 shows two examples of different thresholding

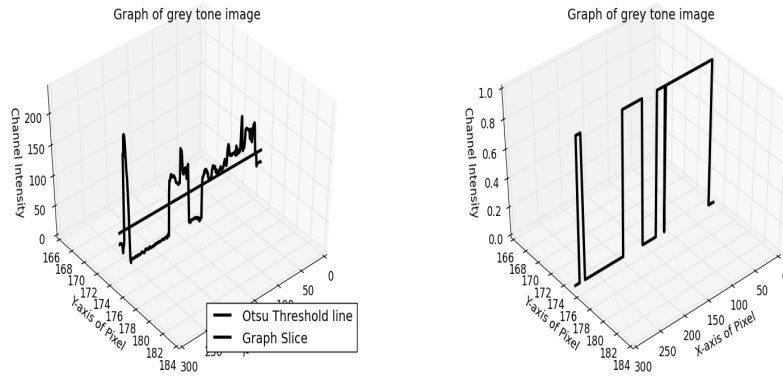


(a) A greyscale image before and after Otsu thresholding was applied.

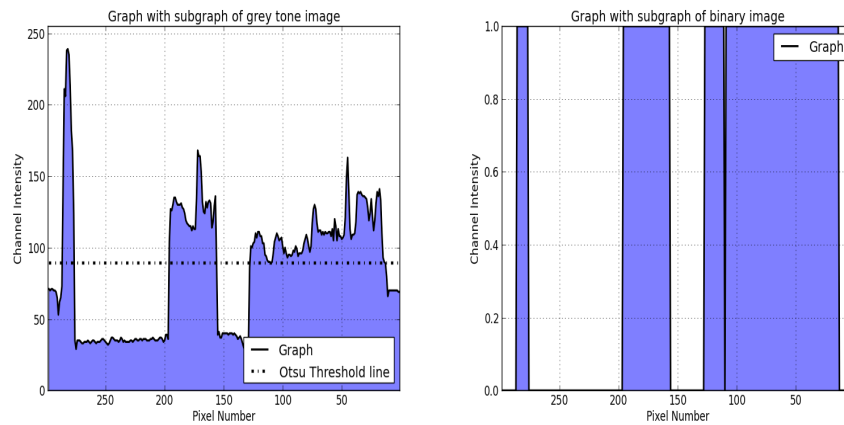


(b) The three dimensional graph of a greyscale image before and after Otsu thresholding was applied.

Figure 2.5: An illustration that shows what happens to a greyscale image when Otsu thresholding is applied.

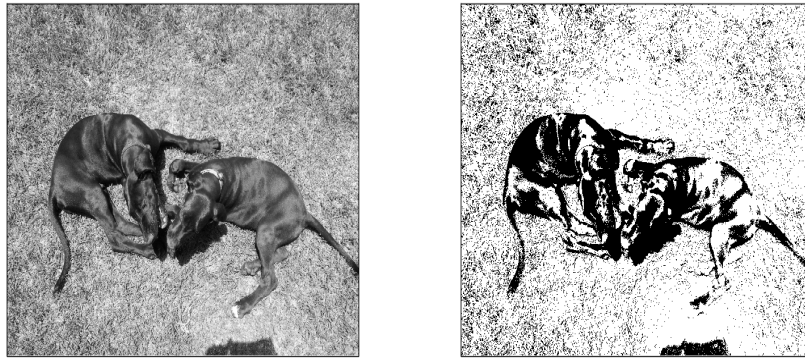


(a) A slice of the three dimensional graph of a greyscale image at row 175 before and after Otsu thresholding was applied.



(b) A slice of the subgraph of a greyscale image taken at row 175 before and after Otsu thresholding was applied.

Figure 2.6: An illustration that shows what happens to a greyscale image when Otsu thresholding is applied.


 (a) Thresholded using $t_i = 0$ and $t_j = 100$.


(b) Thresholded using Otsu thresholding.

Figure 2.7: Two examples of thresholding.

techniques. The first example was thresholded using $i = 0$ and $j = 100$ and the second example was thresholded using Otsu thresholding. The image thresholded using $j = 100$ clearly has more white pixels than the image thresholded using Otsu thresholding which leads to the conclusion that the value for j determined by Otsu thresholding is less than 100 according to the definition.

Neighbourhood image transformations

We now consider neighbourhood image transformations which, in contrast to point image transformations, use a set of nearby pixels as input. Mathematical morphology uses a transformation called spatial convolution, denoted by $*$, as basis to form more neighborhood image transformations. Spatial convolutions use images f , defined on image domain D_f , and a convolution kernel g , with image domain D_g , as input for the transformations. D_g is typically much smaller than D_f and g usually has its origin at the center of D_g . The spatial convolution transformation of f at \mathbf{x} is the weighted sum of pixels in D_f coinciding with D_g when the origin of D_g is the same as \mathbf{x} where the weights are given by $g(\mathbf{b})$ for each $\mathbf{b} \in D_g$ and the result is obtained as:

$$[f * g](\mathbf{x}) = \sum_{\mathbf{b} \in D_g} [f(\mathbf{x} - \mathbf{b})g(\mathbf{b})].$$

In a similar way we can compute the cross-correlation between images f and g :

$$[C(f, g)](\mathbf{x}) = \sum_{\mathbf{b} \in D_g} [f(\mathbf{x} + \mathbf{b})g(\mathbf{b})]$$

which is the convolution of f with a g rotated 180° and if $f = g$ then C is called the autocorrelation of the image. [58]

Ordering relations

When dealing with greyscale images ordering relations are straight forward. Each value that a greyscale image can assume at each pixel can be, unambiguously, ranked among the rest of such values. Multichannel can be seen as a combination of multiple greyscale images defined on the same image definition domain. This causes a problem when trying to find a unique order relation among different pixels. The general solution to this problem is decompose the multichannel image into greyscale images and deal with each greyscale image separately. Ordering relations are important when it comes to image transformations which is why we take pause to discuss some basic definitions here.

The definition of a *partially ordered set*, also known as a *poset*, is a set with some members, say A and B , satisfying the following conditions:

1. Reflexivity: $A \leq A$
2. Anti-symmetry: $A \leq B$ and $B \leq A$ if and only if $A = B$
3. Transitivity: if $A \leq B$ and $B \leq C$, then $A \leq C$

A set meeting an additional property, called the trichotomy property, which states that for any members A and B of such a set only one of the following must be true: either $A < B$, $A = B$ or $B < A$ is known as a *totally ordered set*. [22]

On images and image transformation

Partial ordering of images can be defined in terms of the numerical values associated with pixels in images, in terms of the cross-sections of images or in terms of the subgraphs of images.

- One can say that an image f is greater than or equal to another image g if the values of f are greater than or equal to the associated values of g for all pixels in their common definition domain.
- The same can be said if the cross-section of g is contained in the cross-section of f at all levels of t .
- Similarly the image f is greater than or equal to another image g if the subgraph of f contains the subgraph of g . This can be summarised as follows:

$$f \geq g \Leftrightarrow \forall \mathbf{x} \in D_g, f(\mathbf{x}) \geq g(\mathbf{x}) \Leftrightarrow \forall t, CS_t(f) \supseteq CS_t(g) \Leftrightarrow SG(f) \supseteq SG(g).$$

The ordering translations of transformations are similarly defined. The transformation Ψ_1 is greater than or equal to the transformation Ψ_2 if and only if $\Psi_1(f)$ is greater than or equal to $\Psi_2(f)$ for all images f ,

$$\Psi_1 \geq \Psi_2 \Leftrightarrow \forall f, \Psi_1(f) \geq \Psi_2(f).$$

We can use the threshold transformation to illustrate this ordering relation by comparing the transformation with the threshold parameter $[t_1, t_2]$ against the threshold parameter $[t_1 - n_1, t_2 + n_2]$ where $(n_1, n_2 \in \mathbb{N}_0)$. Because the second interval is larger or equal to the first interval it will always happen that either the same, or more of the values in an image f will fall inside the second interval thus making the threshold transformation, using the second interval, greater than or equal to the threshold transformation when using the first interval, that is,

$$T_{[t_1, t_2]} \leq T_{[t_1 - n_1, t_2 + n_2]}.$$

On set partitions

A set is said to be partitioned if it is divided into nonempty disjoint sets which, together, make up the original set. We say that the collection of the sets X_1, X_2, \dots, X_n , also called segments of X , define a partition Π of a set X if and only if the following hold:

1. No empty subsets: $\forall X_i \in \Pi(X), X_i \neq \emptyset$
2. All subsets are disjoint: $\forall X_i, X_j \in \Pi(X), i \neq j \Rightarrow X_i \cap X_j = \emptyset$
3. The union completely covers X : $\bigcup_i \{X_i | X_i \in \Pi(X)\} = X$

The set X may be partitioned in many different ways. Let Π_i refer to a partition of X and let $\pi_i(p)$ be an arbitrary segment found in Π_i containing a pixel p . We have that $\Pi_i \leq \Pi_j$ if and only if $\forall p \in X, \pi_i(p) \subseteq \pi_j(p)$ which translates to Π_i being a finer partition than Π_j . The finest partition of any set is the set segments each consisting of one pixel and the coarsest partition is a partition with one set which is equal to the definition domain itself [58].

Greyscale image functions

Greyscale images are seen as functions rather than sets as is the case for binary images. The union \cup and intersection \cap operators used with sets are thus no longer applicable. Instead we use the point-wise maximum \vee and point-wise minimum \wedge operators. For any image f and g these operators are defined as follows:

$$(f \vee g)(\mathbf{x}) = \max[f(\mathbf{x}), g(\mathbf{x})],$$

$$(f \wedge g)(\mathbf{x}) = \min[f(\mathbf{x}), g(\mathbf{x})].$$

This means that the point-wise maximum operator outputs the maximum value of the two images f and g at a certain point \mathbf{x} in the shared image definition domain D . Similarly the point-wise minimum operator outputs the minimum value of the two images f and g at a certain point \mathbf{x} in the shared image definition domain D [58].

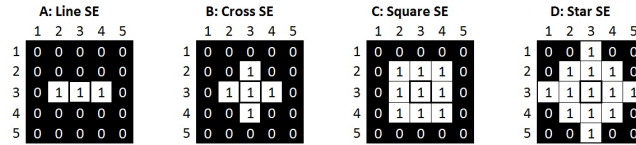


Figure 2.8: Examples of different binary structuring elements with origin at the center of each.

2.3 Erosion and Dilation

2.3.1 Structuring Elements

A structuring element (SE) is a small image with predetermined pixel intensity used to transform, operate or extract information from an image. In order to position the structuring element at a certain pixel we require it to have an origin, say \mathbf{x} . The origin doesn't necessarily have to be at the center of the structured element. Structuring elements can also be combined to form new structuring elements by taking the union of two or more structuring elements for example. When using a structuring element to examine an image, one should consider the geometry of the image in question when choosing which structuring element to use. While we shall only use binary structuring elements to analyse binary images, we can use either binary structuring elements or greyscale structuring elements to analyse greyscale images as well. Popular structuring elements are shown in Figure 2.8 where the neighbourhood of the structuring elements are given by the pixels with intensity level 1. [58]

2.3.2 Erosion

Binary erosion makes use of a technique first introduced by Minkowski and is called Minkowski subtraction [20]. This technique combines two sets using vector subtraction of set elements [57]. The erosion ε_B of set X by some structuring element B , with origin \mathbf{x} , determines the subset of points $\varepsilon_B(X)$ in X where B can be contained inside of X and is denoted by

$$\varepsilon_B(X) = \{\mathbf{x} | B_{\mathbf{x}} \subseteq X\} = \bigcap_{\mathbf{b} \in B} X_{-\mathbf{b}}$$

where the latter equality is the intersection of the set translations being defined by the structured element B [58]. This definition also applies to greyscale images. The erosion ε_B of a greyscale image f by a structuring element B is defined as

$$\varepsilon_B(f) = \bigwedge_{\mathbf{b} \in B} f_{-\mathbf{b}}$$

where \bigwedge is the point-wise minimum operator. In terms of a greyscale image this means that the erosion of f , over the area of the structured element B , is the minimum value in that area [58]. The process of eroding an image by placing the SE at each pixel is shown in Figure 2.9 and an example of an eroded greyscale image is shown in Figure 2.10. Figure 2.11 shows the original two dimensional graph of an image and the result after erosion with different size radius structuring elements. It is clear that the image pixel intensities are decreased as the size of the SE increases.

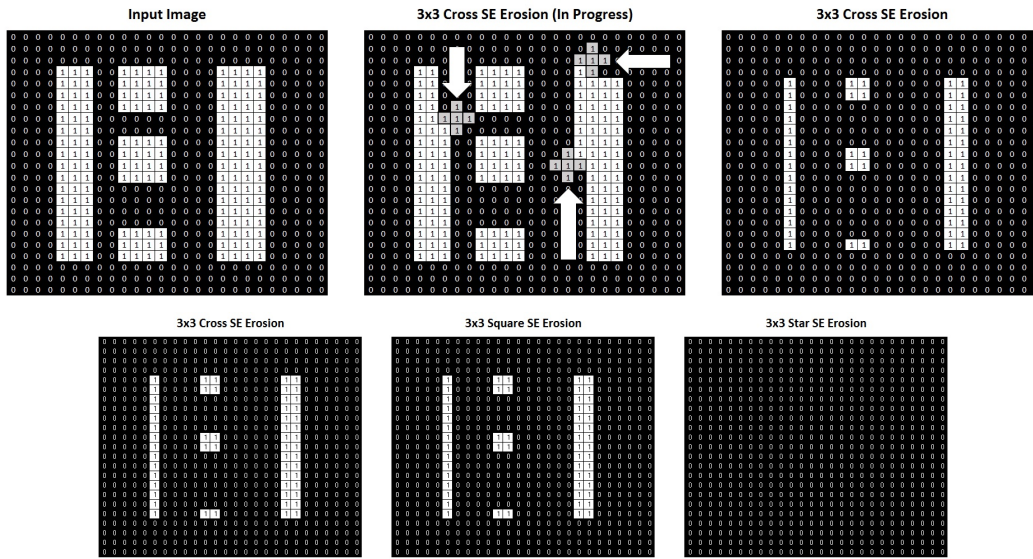
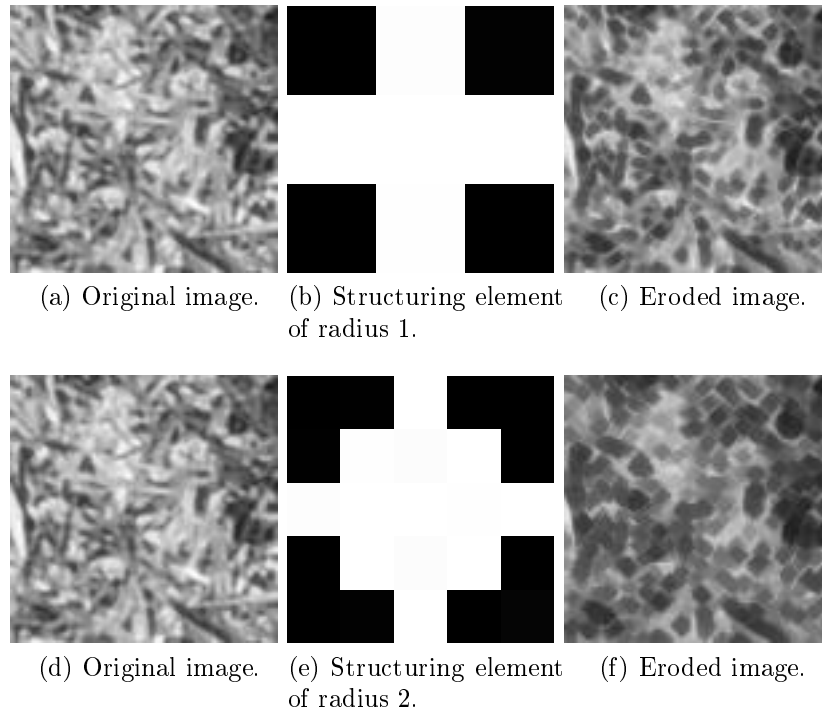


Figure 2.9: The final product of erosion of the input image using structured elements B, C and D as seen in Figure 2.8.



(a) Original image. (b) Structuring element of radius 1. (c) Eroded image.
(d) Original image. (e) Structuring element of radius 2. (f) Eroded image.

Figure 2.10: An example of an eroded greyscale image with two different size radius structuring elements.

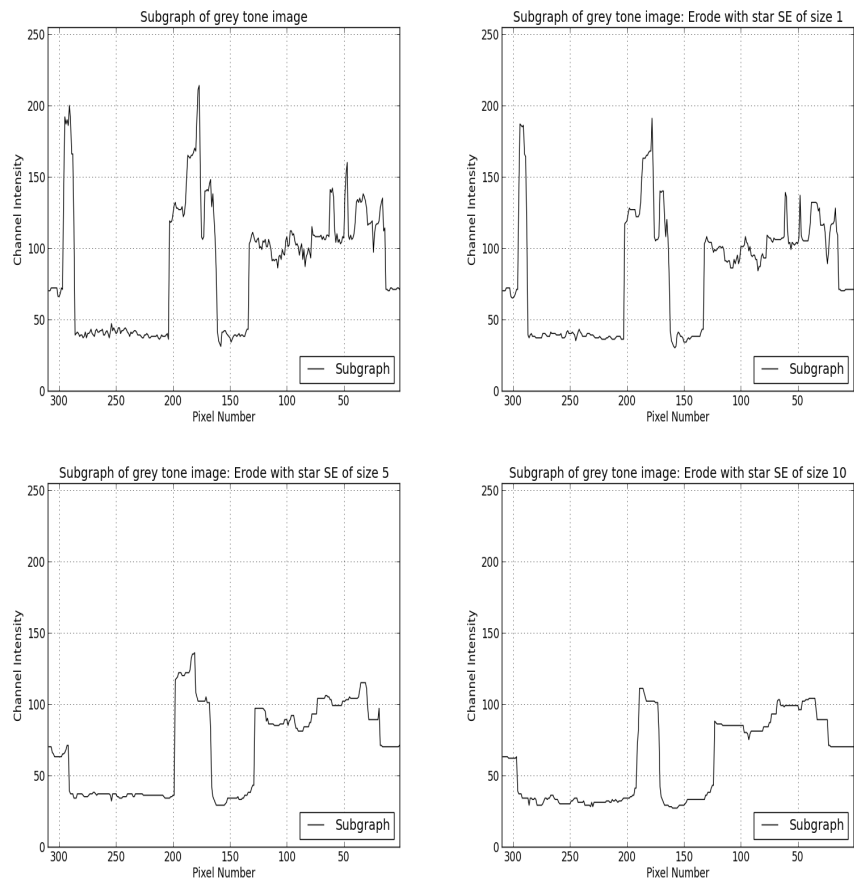


Figure 2.11: Example of the subgraph of a greyscale image eroded by a star-shaped structuring element of radius 1, 5 and 10 respectively.

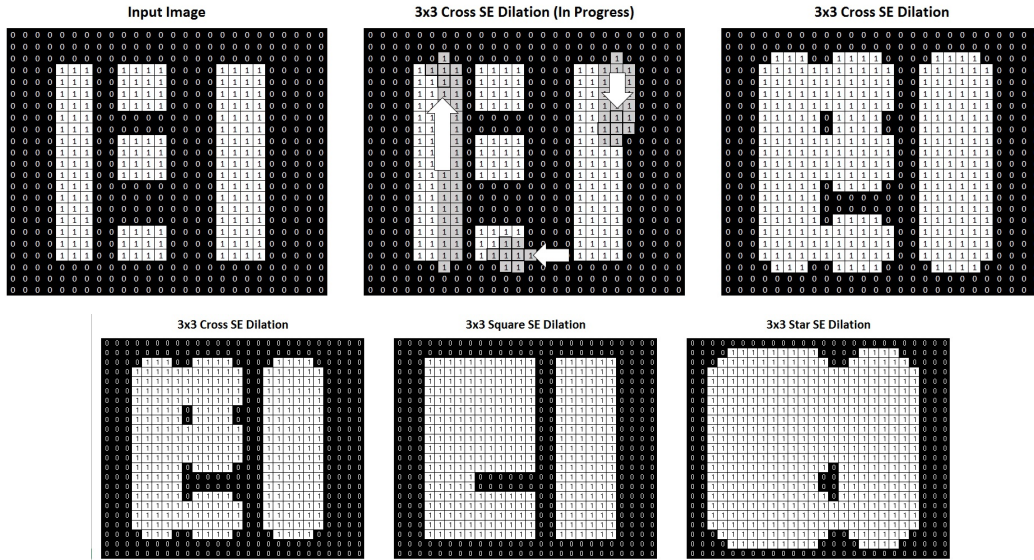


Figure 2.12: The final product of dilations of the input image using structuring elements B, C and D.

2.3.3 Dilation

The dilation δ_B of set X by some structuring element B , with origin \mathbf{x} , determines the set of points $\delta_B(X)$ for which B will make contact with X and is given by

$$\delta_B(X) = \{\mathbf{x} | B_{\mathbf{x}} \cap X \neq \emptyset\} = \bigcup_{\mathbf{b} \in B} X_{-\mathbf{b}}$$

where the latter equality is the union of the set of translations being defined by the structured element B [58]. This definition also applies to greyscale images. The dilation δ_B of a greyscale image f by a structuring element B is defined as:

$$\delta_B(f) = \bigvee_{\mathbf{b} \in B} f_{-\mathbf{b}}$$

where \bigvee is the point-wise maximum operator. In terms of a greyscale image this means that the dilation of f , over the area of the structured element B , is the maximum value in that area [58]. The process of dilating an image by placing the SE at each pixel is shown in Figure 2.12 and an example of a dilated greyscale image is shown in Figure 2.13. Figure 2.14 shows the original two dimensional graph of an image and the result after dilation with different size radius structuring elements. It is clear that the image frequencies are increase as the size of the SE increases.

2.3.4 Image Transformation Properties

Image to image transformation properties can be used to identify what transformation will be appropriate to obtain a desired result. We will discuss some basic image transformation properties and later show some fundamental morphological transformations where they are applicable.

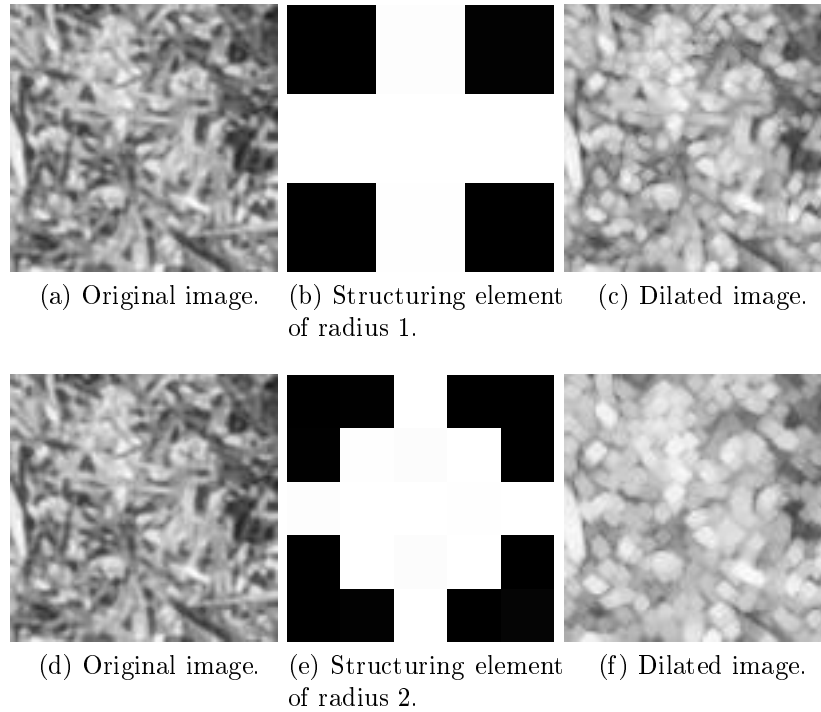


Figure 2.13: An example of a dilated greyscale image with two different size radius structuring elements.

Robustness Criterion

We define two types of robustness criteria. The first is invariance to translation which means that a transformation Ψ commutes with image translations. The formal definition is as follows:

$$\Psi \text{ is invariant to translation} \iff \forall f, \forall \mathbf{b} \in \mathbb{Z}^2, \Psi(f_{\mathbf{b}}) = [\Psi(f)]_{\mathbf{b}}.$$

Operators that transform one pixel at a time are all invariant to translation since the transformation of that pixel is exactly the same no matter what value its neighbours assume. The second criterion is invariance to rotations which means that a transformation Ψ commutes with rotations Θ . This is defined as follows:

$$\Psi \text{ is invariant to rotations} \iff \Psi\Theta = \Theta\Psi,$$

where Θ represents a general, image rotation transformation. This is a very useful property when it comes to detecting objects in images since one doesn't have to be concerned about the angle at which an object lies in the image.

Local Knowledge

When considering a neighbourhood image transformation on a certain definition domain D we can imagine that some problems may occur around the edges of such a definition domain considering that the necessary neighbouring points may not exist. This leads to the definition of the local knowledge property: Let D' be some subset of the definition domain D , ie. D' is in D , then the image transformation Ψ has the property of local knowledge if the transformation of any point

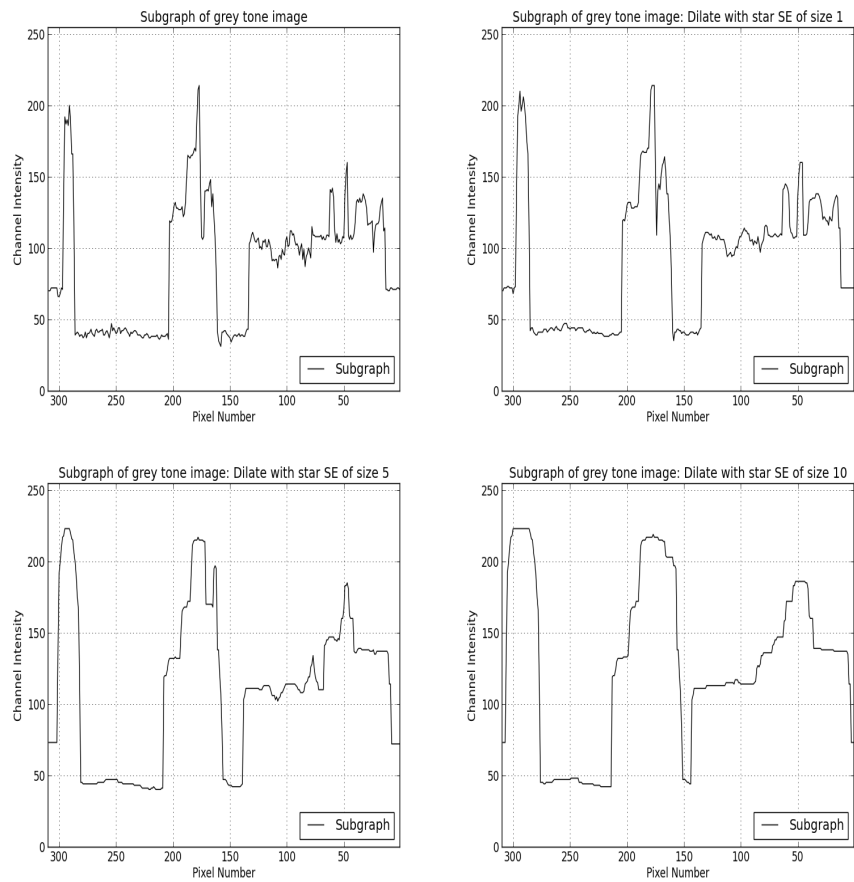


Figure 2.14: Example of the subgraph of a greyscale image dilated by a star-shaped structuring elements of radius 1, 5 and 10 respectively.

in D , given knowledge of all of D , is exactly the same as when the same point is contained and transformed in any D' with only knowledge of D' . We can also write

$$\Psi(f|D)|D' = \Psi(f)|D'.$$

Any point-wise image transformation trivially also has the local knowledge property.

Idempotence

Image transformations have the idempotence property if, for all images f , the same transformation can be applied again to an image f and the transformed image will be the same as the image f transformed once which give us the following definition:

$$\Psi \text{ is idempotent} \iff \Psi\Psi = \Psi.$$

Extensivity/Anti-Extensivity

An image transformation Ψ is said to be have the extensivity property if for all images f the transformation of f is greater than or equal to the image used as input. This can also be written as:

$$\Psi \text{ is extensive} \iff id \leq \Psi.$$

The opposite is true for image transformations with the anti-extensivity property. A image transformation Ψ which causes the transformed image to be less than or equal to the original image for all images f is said to have the anti-extensivity property and can be written as:

$$\Psi \text{ is anti-extensive} \iff id \geq \Psi.$$

Increasingness

The image transformation Ψ is said to have the property of increasingness if, for all images f and g , it preserves the ordering relation between the images, hence,

$$\Psi \text{ is increasing} \iff \forall f, g, f \leq g \implies \Psi(f) \leq \Psi(g).$$

A transformation that adds a constant value for each pixel value of an image would be an example of a image transformation that has the increasingness property.

Duality

If applying an image transformation Ψ to an image results in the same output when applying the image transformation Φ to the complement of said image and taking the complement of the result, then such image transformations are said to be dual with respect to complementation. The definition is given as follows:

$$\Psi \text{ and } \Phi \text{ are dual with respect to complementation } \mathbb{C} \iff \Psi = \mathbb{C}\Phi\mathbb{C}$$

where the complement image \mathbb{C} is of an image consisting intensity levels calculated as the difference between the maximum possible image intensity and the actual image intensity. The following results then hold:

$$\Phi \text{ idempotent} \implies \Psi \text{ idempotent}$$

$$\Phi \text{ extensive} \iff \Psi \text{ anti-extensive}$$

$$\Phi \text{ increasing} \implies \Psi \text{ increasing}$$

Composition or Separability

This property implies that if a structuring element can be decomposed into smaller structuring elements, of which the union is the same as the original structuring element, then eroding/dilating with such a structuring element would be the same as eroding/dilating with its smaller counterparts. The following equations clearly express the composition of erosions $\varepsilon_{B_2}, \varepsilon_{B_1}$ and dilations $\delta_{B_2}, \delta_{B_1}$:

$$\delta_{B_2} \delta_{B_1} = \delta_{(\delta_{\bar{B}_2} B_1)},$$

$$\varepsilon_{B_2} \varepsilon_{B_1} = \varepsilon_{(\delta_{\bar{B}_2} B_1)}.$$

This property allows for massive computational savings. Using this property there would only be $2(n-1)$ comparisons but n^2-1 without when eroding an image [58].

Distributivity

The following holds for dilation δ and erosion ε in terms of distributivity:

$$\delta \left(\bigvee_i f_i \right) = \bigvee_i \delta(f_i)$$

and

$$\varepsilon \left(\bigwedge_i f_i \right) = \bigwedge_i \varepsilon(f_i).$$

The above means that the dilation operator δ (erosion operator ε) is indifferent to the order in which it is applied together with the point-wise maximum operator \bigvee (minimum operator \bigwedge) [58].

2.3.5 Morphological Opening

Since the erosion operator removes a lot of foreground information together with some of the noise, as seen in 2.11, we dilate the eroded image in order to recover some of the foreground information lost. The opening of the set X by some structuring element B is the erosion $\varepsilon_B(X)$, followed by the dilation $\delta_B(X)$ and is given by

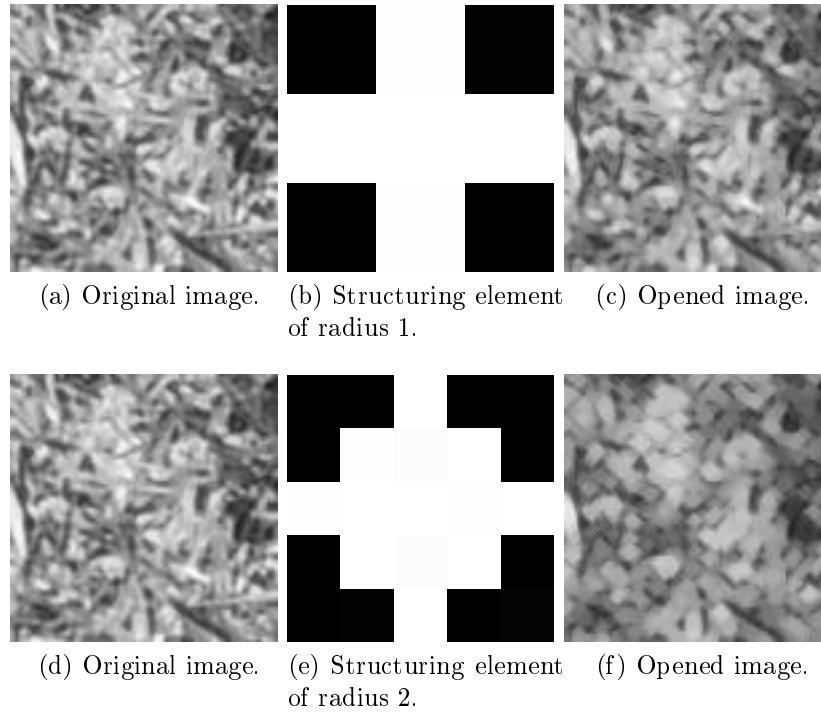


Figure 2.15: Before and after the morphological opening of a greyscale image using different size radius structuring elements.

$$\gamma_B(X) = \bigcup_{\mathbf{x}} \{B_{\mathbf{x}} | B_{\mathbf{x}} \subseteq X\}$$

which is union of all translations of B such that B is included in X . This definition is extendable to a greyscale image f and is defined as the erosion of f by B followed by the dilation by B [58]:

$$\gamma_B(f)(x) = \delta_B[\varepsilon_B(f)] = \bigvee_{y \in \check{B}_{\mathbf{x}}} \bigwedge_{z \in B_{\mathbf{y}}} f(z).$$

The effect of a morphological opening can be seen on images in Figure 2.15 and how the two dimensional graph is effected in Figure 2.16.

2.3.6 Morphological Closing

Dilating an image exaggerates the foreground of the image, as seen in Figure 2.14. By eroding the dilated image, small holes in the foreground may be filled while keeping the general structure of the foreground. The closing of the set X by some structuring element B is the dilation $\delta_B(\mathbf{X})$, followed by the erosion $\varepsilon_{\check{B}}(\mathbf{X})$ with the reflected structuring element \check{B} and is given by

$$\phi_B(X) = \left[\bigcap_{\mathbf{x}} \{B_{\mathbf{x}} | B_{\mathbf{x}} \subseteq X^c\} \right]^c$$

which is the intersection of the translations of the complement of the structuring element B such that the complement of the structuring element B contains \mathbf{X} . This definition is again extendable

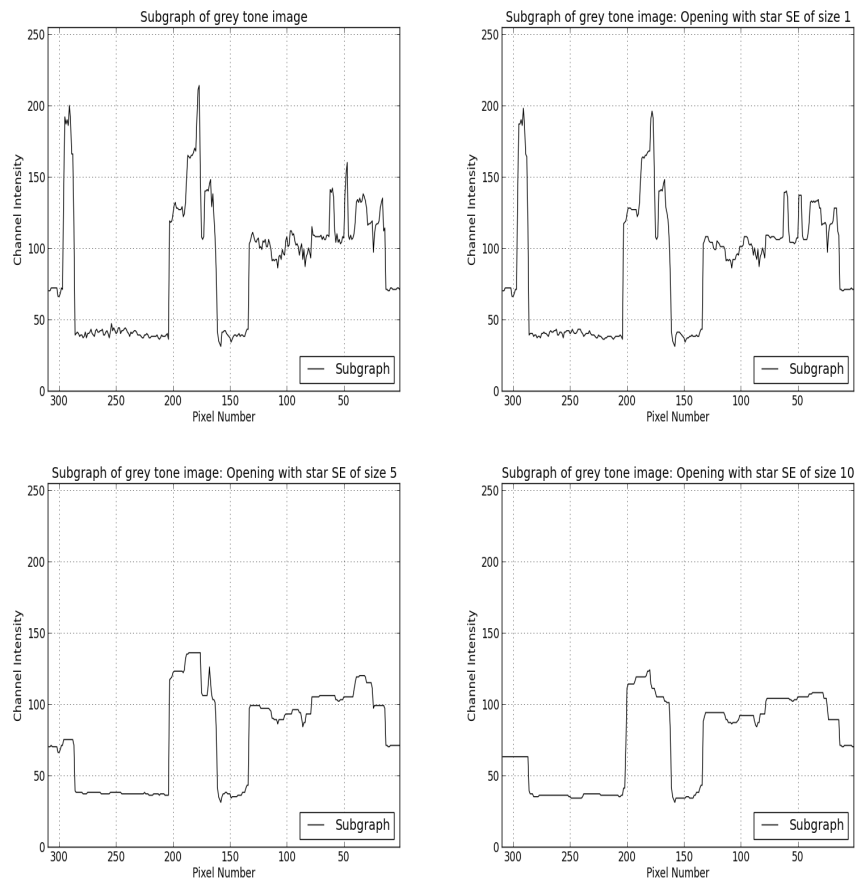


Figure 2.16: An example of the morphological opening of the subgraph of a greyscale image by a star-shaped structuring element of radius 1, 5 and 10 respectively.

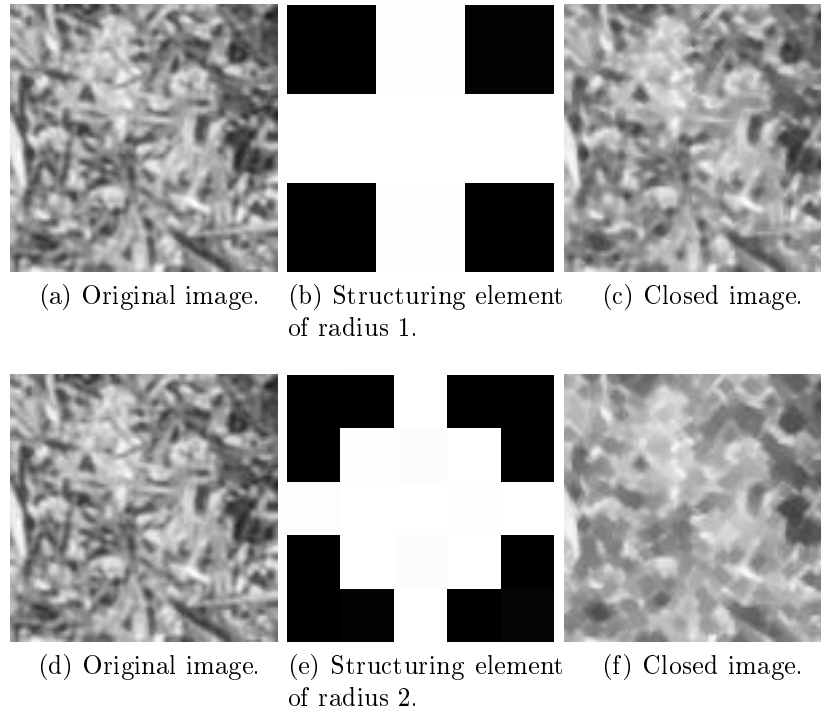


Figure 2.17: Before and after the morphological closing of a greyscale image using different size radius structuring elements.

to a greyscale image f eroded by a structuring element B and is defined as the dilation of f by B followed by the erosion by B [58]:

$$\phi_B(f) = \varepsilon_B[\delta_B(f)] = \bigwedge_{y \in \tilde{B}_x} \bigvee_{z \in B_y} f(z).$$

The effect of a morphological closing can be seen on images in Figure 2.17 and how the two dimensional graph is effected in Figure 2.18.

2.4 Conclusion

In this chapter we have discussed how different images are digitised into a format that enable us to analyse and manipulate them. We have shown how image transformations work and gave some properties surrounding them and we discussed the basic operators that form the building blocks of mathematical morphology algorithms. This theory will now enable us to continue to Chapter 3 where we will discuss methods and algorithms used in the field of image analysis.

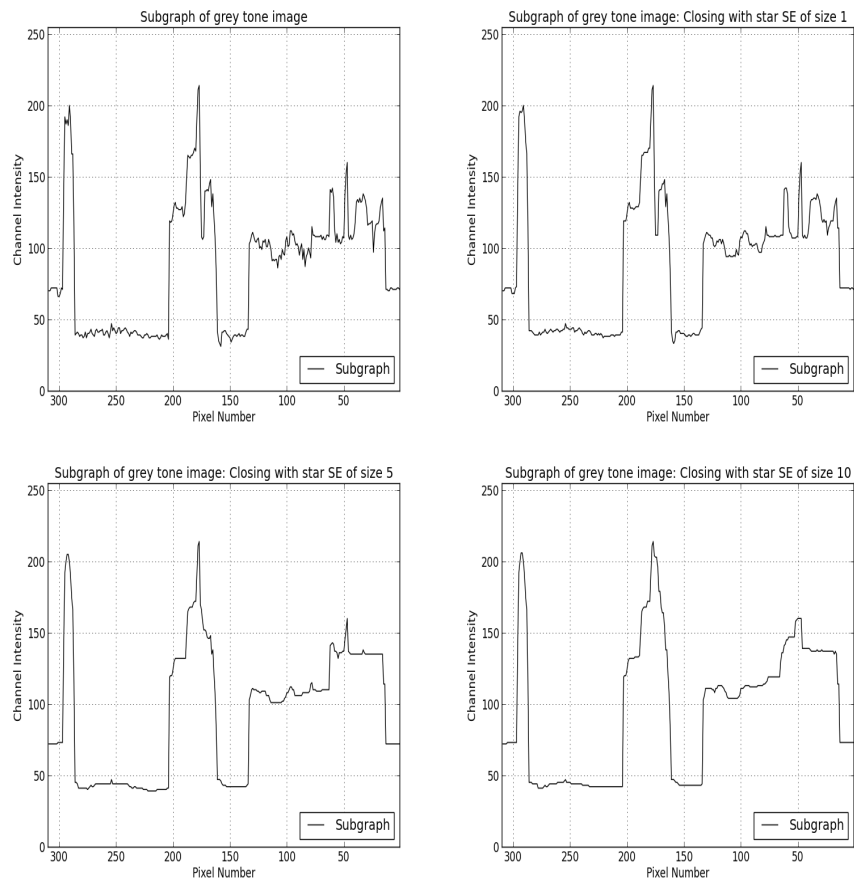


Figure 2.18: An example of the morphological closing of the subgraph of a greyscale image by a star-shaped structuring element of size 1, 5 and 10 respectively.

Chapter 3

Algorithms

3.1 Introduction

We now look at various smoothing techniques as well as several contour detection algorithms. We focus on these two different types of methods as they will form the basis of the combinations of algorithms that we'll use in Chapter 4 in order to identify and extract objects from images. In Section 3.2 the median, together with the adaptive median, image smoothing technique is explored as it is widely used in practice [60, 32, 1, 27, 64] and can be seen as the golden standard of image smoothing techniques [64, 1]. Next we investigate a noise specific smoothing method called the robust statistic based algorithm [62] which we expect to perform well under certain noise conditions. We also look at an image smoother called the *LULU* smoother which takes a different approach in identifying image pixels that need to be smoothed [4]. Finally we introduce a new image smoothing method called the connected median filter constructed using different elements of the aforementioned image smoothers. We compare the different image smoothing techniques in terms of the amount of noise removed from the images using the peak signal to noise ratio [26]. In Section 3.4 we look at three different algorithms that can be used to extract objects from images. The first contour detection algorithm is a traditional border following approach [52, 43, 59, 21] and is followed by the well known and very popular watershed segmentation algorithm [7, 6]. We conclude this chapter by looking at a machine learning approach to the problem of extracting contours from images, namely the active contour detection algorithm [33, 41, 37, 5, 39] which is a more robust algorithm.

3.2 Noise and Smoothers

The removal of image noise, incurred with image acquisition, is often the first step in image analysis [28]. The most common types of noise are Gaussian random noise (speckle noise) and salt & pepper noise [12, 19] which is why we investigate techniques to remove these types of noise. Gaussian random noise can occur as a result of the image acquisition system and can be represented by statistical noise having a normal distribution. Salt & pepper noise is a type of noise that is presented as pixels at maximum or minimum intensity level within the image and may occur as a result of information loss due to malfunctioning pixels in camera sensors, faulty memory locations in hardware and other reasons that would result in the loss of data [10]. Throughout this section we add various intensity levels of these two different kinds of noise to images. Figures 3.1 and 3.2

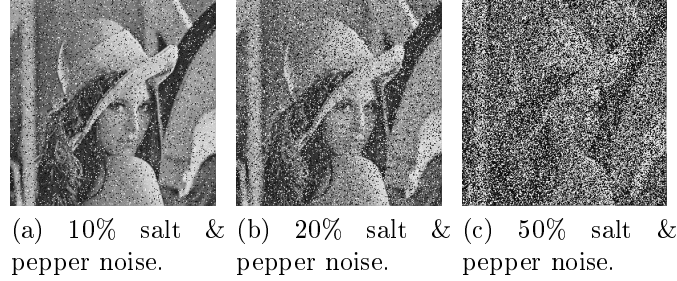


Figure 3.1: ‘Lena’ image with different levels of salt & pepper noise added.

show various noise levels added to the ‘Lena’ image. For Gaussian random noise (speckle noise) we show the corruption level as a level of increased total variation in the image pixel intensities. If $f : \mathbb{Z}^2 \rightarrow \{0, \dots, 255\} \subset \mathbb{Z}$ is a greyscale image with intensity level given by f_{ij} at pixel location i, j then the total variation (TV) [54] of f is given by

$$TV(f) = \sum_i \sum_j [|f_{ij} - f_{i,j+1}| + |f_{ij} - f_{i-1,j}|].$$

Let f_{OI} be the original image and f_{NI} the noisy image then we can write the increase in total variation as follows

$$\text{Noise}_{\text{Speckle}}\% = 100 \left[\frac{TV(f_{NI})}{TV(f_{OI})} - 1 \right].$$

This can be interpreted as the % increase in total variation caused by the added speckle noise. For salt & pepper noise we corrupt a certain percentage of the total image pixels in the image with pixels at maximum intensity levels with no preference between 0 or 255. The noise ratio can be written as

$$\text{Noise}_{S\&P}\% = 100 \left[\frac{\text{card}\{y : y \in D_{f_{NI}}, f_{NI}(y) \in \{0, 255\}\}}{\text{card}\{x : x \in D_{f_{OI}}\}} - 1 \right].$$

We will now look at different kinds of smoothers and compare them with each other in terms of how closely the smoothed image resembles the original image, under different noise conditions, by using a peak signal to noise ratio. The peak signal to noise ratio (PSNR) is given by

$$\text{PSNR} = 10 \log_{10} \left[\frac{255^2}{\frac{1}{MN} \sum_i \sum_j (o_{ij} - s_{ij})^2} \right]$$

where o_{ij} is the pixel intensity of the original pure signal image $f : M \times N$ and s_{ij} is the pixel intensity of the smoothed image at pixel (i, j) . Increasing (decreasing) values of PSNR indicate that the number of pixels containing signal is increasing (decreasing) relative to the number of pixels containing noise. The range of the PSNR is usually between 30 and 50 for compressed 8bit images. [26]

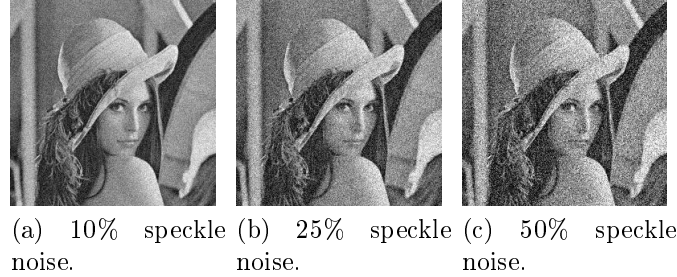


Figure 3.2: ‘Lena’ image with different levels of speckle noise added.

3.2.1 Median Filter

The median filter is one of many non-linear smoothing methods and was first introduced by Tukey in the late 70’s to smooth economic time series [60, 32]. Since then, many other non-linear smoothing filters have been created based on the original work done by Tukey [27, 64]. This filter is based on order statistics and is preferred to many other non-linear filters because of its simplicity and its ability to reduce local pixel intensity variability . [1, 32].

If $\{x_{(1)}, x_{(2)}, \dots, x_{(n)}\}$ is a set of ordered numbers and n is odd, then the median of the set is the number in the middle of the ordered set also denoted by

$$M_n = \text{median} \{x_1, x_2, \dots, x_n\} = \begin{cases} x_{(\frac{n+1}{2})} & \text{for } n \text{ odd.} \\ \frac{x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}}{2} & \text{for } n \text{ even.} \end{cases} .$$

The median, together with a two dimensional structuring element (SE), can be used to construct a filter to remove noise from images. This is done by replacing the pixel intensity of the pixel at the center of a SE by the median of the original intensities of the pixels in the SE [32]. When additive i.i.d Laplacian noise is present, the median is also the maximum likelihood estimate of the signal level, which means that the median value M_n will minimise the sum of absolute differences τ as follows according to [66]

$$\tau(M_n) = \sum_{i=1}^n |x_i - M_n|.$$

Smoothing Algorithm 1

If we use a 3×3 SE to smooth an image, with the median filter, the process will be as follows:

1. First we create a copy of the image and call it the Transformed Image (TI) and we also keep an original copy of the image (OI).
2. We then place the SE on OI centered at, say, pixel (i, j) .
3. The intensity levels of the pixels covered by the SE are then placed in a set $\{x_1, x_2, \dots, x_9\}$ and ordered from smallest to largest to form the set $\{x_{(1)}, x_{(2)}, \dots, x_{(9)}\}$.
4. We then replace the intensity level f_{ij} in TI with the selected element $x_{(\frac{9+1}{2})} = x_{(5)}$ from OI.

5. By moving the SE across the image and repeating this process, from step 1, for every pixel f_{ij} in OI will result in an image transformed by the median filter [24].

This process is illustrated in Figure 3.3.

Example 3.2.1.

The median filter is a useful non-linear smoother, but does not discriminate between signal and noise in images.

The adaptive median filter (AMF) was created in the 1980s to overcome some of the problems faced by median filter [16]. The main improvement is the AMF's ability to change the size of the window according to the each median of every window being considered in order to differentiate between pixels containing noise and pixels containing signal [1].

The algorithm works as follows:

Smoothing Algorithm 2

Let f denote a noisy image with pixels intensity values $\{f_{ij}\}$. Let

$$S_{ij} = \{f_{i-u,j-v} : -L \leq u \leq L, -L \leq v \leq L\}$$

be a window defined on f and centered at (i, j) . The size of the window can be calculated as $w = (2L + 1)^2$. Let w represents the current SE size and w_{\max} the maximum allowed SE size. For each pixel (i, j) :

1. Set $L = 1$ such that $w = (2(1) + 1)^2 = 9$ for the initial window size (usually $L = 1$).
2. Compute the minimum $(S_{\min}^{(ij)})$, median $(S_{\text{med}}^{(ij)})$ and the maximum $(S_{\max}^{(ij)})$ inside the window.
3. If $S_{\min}^{(ij)} < S_{\text{med}}^{(ij)} < S_{\max}^{(ij)}$,
then proceed with step 5,
else go to step 4.
4. If $w \leq w_{\max}$,
then set $L = L + 1$ and go back to step 2,
else set $f_{ij} = S_{\text{med}}^{(ij)}$.
5. If $S_{\min}^{(ij)} < f_{ij} < S_{\max}^{(ij)}$,
then we assume that f_{ij} is a pixel intensity value containing signal and leave it as is,
else we replace f_{ij} with $S_{\text{med}}^{(ij)}$.

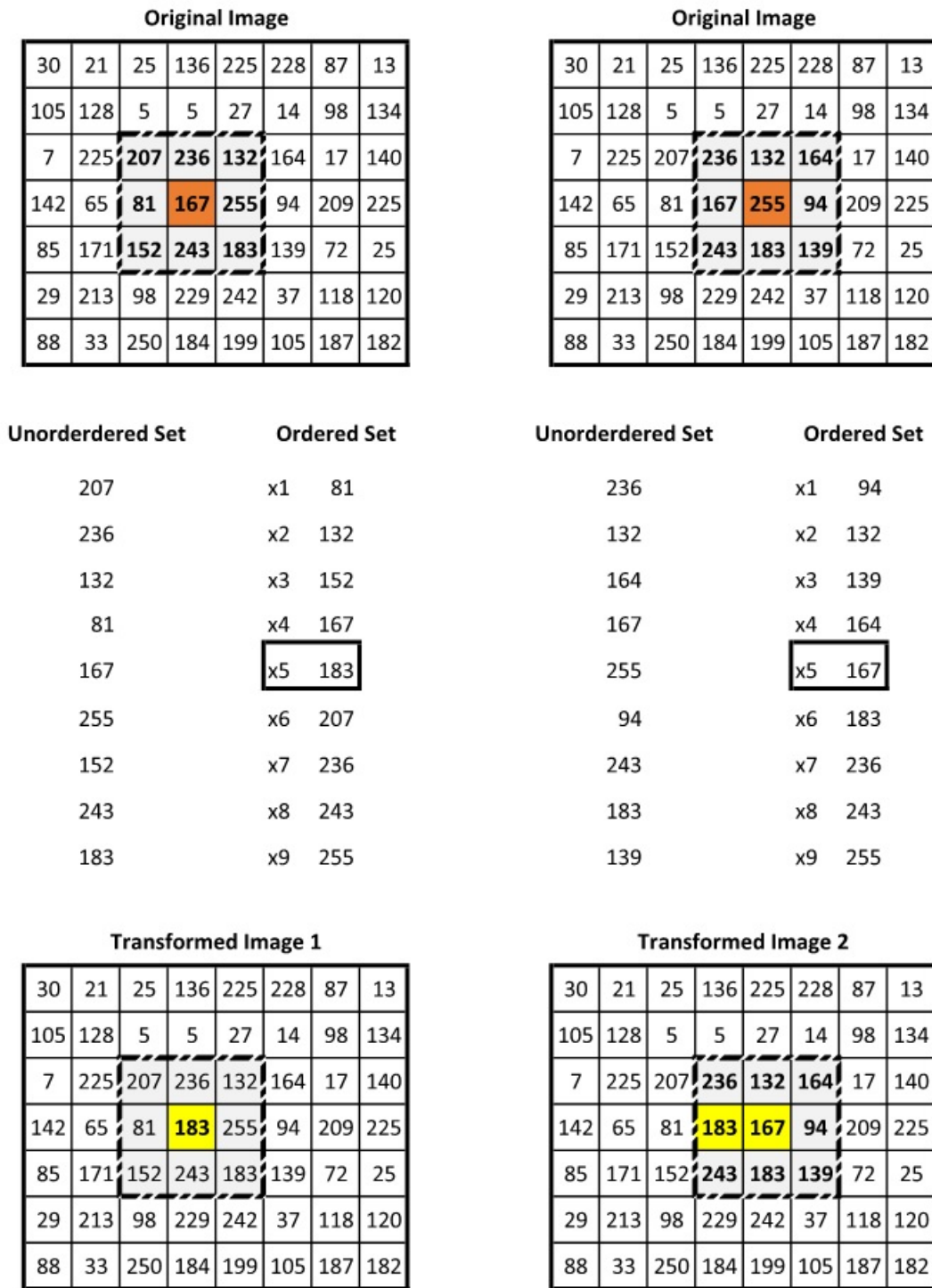


Figure 3.3: An illustration of the process followed by the median filter.

Original Image

39	1	205	77	46	106	81	117
185	126	75	40	60	41	52	37
30	135	193	201	244	35	101	251
249	154	190	107	111	191	204	188
205	145	210	237	202	238	178	18
85	17	119	91	218	150	88	158
233	165	39	154	81	82	110	161

Figure 3.4: The original image with SE centered at f_{ij} .

Example 3.2.2.

The original image is shown in Figure 3.4. We will illustrate how the algorithm works for a single pixel at position (i, j) .

1. Set $L = 1$ to give a SE of size 3×3 and we center that SE at pixel (i, j) .
2. We calculate values $S_{\min}^{(ij)}$, $S_{\text{med}}^{(ij)}$ and $S_{\max}^{(ij)}$:

$$S_{\min}^{(ij)} = 107$$

$$S_{\text{med}}^{(ij)} = 201$$

$$S_{\max}^{(ij)} = 244$$
3. Since $S_{\min}^{(ij)} < S_{\text{med}}^{(ij)} < S_{\max}^{(ij)}$ we proceed with step 5.

For step 5 $f_{ij} = 107$ but f_{ij} is also equal to $S_{\min}^{(ij)}$ so our new estimate for f_{ij} is $S_{\text{med}}^{(ij)} = 201$.

3.2.2 Robust Statistics Based Algorithm

The robust statistics based algorithm proposed in [62] uses redescending estimators in order to cancel the influence of outliers. An estimator is said to be redescending if it's non-decreasing near origin and non-increasing further away from the origin. The redescending property is in part thanks to the Lorentzian influence function used to distributed weights to intensity values [62]. The Lorentzian estimator is defined by

$$\rho_{\text{LOR}}(x) = \log \left(1 + \frac{x^2}{2\sigma^2} \right)$$

and the its influence function by

$$\psi_{\text{LOR}}(x) = \rho'_{\text{LOR}}(x) = \log \left(1 + \frac{2x}{2\sigma^2 + x^2} \right).$$

Smoothing Algorithm 3

Let f denote a noisy image with pixels intensity values $\{f_{ij}\}$. Let

$$S_{ij} = \{f_{i-u,j-v} : -L \leq u \leq L, -L \leq v \leq L\}$$

be a window defined on f and centered at (i, j) using u and v to determine the pixels in the the window relative to L . The size of the window can be calculated as $w = (2L + 1)^2$. For example $L = 1$ gives a 3×3 SE and w will be $w = (2 \cdot 1 + 1)^2 = 9$.

For each pixel (i, j) :

1. Set the minimum window size $L = 1$.
2. Compute the minimum $(S_{\min}^{(ij)})$, maximum $(S_{\max}^{(ij)})$ and the median value $(S_{\text{med}}^{(ij)})$ inside the window.
3. If $S_{\min}^{(ij)} < f_{ij} < S_{\max}^{(ij)}$, then it is assumed that the pixel is uncorrupted and left unchanged, else go to step 4.
4. Select the pixels in the window such that $S_{\min}^{(ij)} < f_{i-u,j-v} < S_{\max}^{(ij)}$:
If the number of such pixels is less than 1, then $L = L + 1$ and return to step 2, else go to step 5.
5. The difference of each pixel intensity value inside the window is calculated as $x_{i-u,j-v} = f_{i-u,j-v} - S_{\text{med}}^{(ij)}$ and substituted into the robust influence function,

$$r(x_{i-u,j-v}) = \frac{2x_{i-u,j-v}}{2t^2 + x_{i-u,j-v}^2}$$

where $t = \frac{\tau_s}{\sqrt{2}}$ is the outlier rejection point, $\tau_s = \zeta\sigma_\nu$ the maximum expected outlier, σ_ν is the local estimate of the image standard deviation and ζ a smoothing factor. For median smoothing, $\zeta = 0.3$ works well according to [62].

6. The value of the pixel is then estimated using the ratio of S_1 and S_2 where

$$S_1 = \sum_{S_{i,j}} \frac{f_{i-u,j-v} \cdot r(x_{i-u,j-v})}{x_{i-u,j-v}},$$

$$S_2 = \sum_{S_{i,j}} \frac{r(x_{i-u,j-v})}{x_{i-u,j-v}}.$$

Step 5 is based on the influence function of the Lorentzian estimator which results in the influence of outliers converging to zero as they stray further away from the point of interest. There will always be an $x_{i-u,j-v}$ that is equal to zero because one of the $f_{i-u,j-v}$ pixels will be the median. In the referenced article this was not specifically mentioned, we assume that the sums in step 6 exclude the terms where $x_{i-u,j-v} = 0$.

Original Image

237	184	158	103	6	238	151	191
188	77	179	124	59	59	136	215
128	79	179	124	52	252	85	199
78	205	229	22	109	99	192	46
183	36	72	182	175	44	70	72
53	127	143	24	164	74	113	19
180	28	1	178	172	107	175	169

Figure 3.5: The original image considered with a 3×3 SE.

Example 3.2.3.

Consider Figure 3.5 where a SE of size 3×3 is centered at pixel intensity value $f_{ij} = 22$. We now follow the steps of the algorithm proposed in [62].

1. $L = 1$ which gave us the 3×3 SE used to evaluate the image.
2. The following values were calculated::

$$S_{\min}^{(ij)} = 22$$

$$S_{\text{med}}^{(ij)} = 124$$

$$S_{\max}^{(ij)} = 229$$
3. Since $f_{ij} = 22$ is equal to $S_{\min}^{(ij)}$, the algorithm tells us to go to step 4.
4. The number of pixels with values between the $S_{\min}^{(ij)} = 22$ and $S_{\max}^{(ij)} = 229$ is 6, which means that we can go to step 5.
5. We calculate the distance vector x as

$$x = \{55, 0, -72, 105, -102, -15, -52, 58, 51\}$$

and the vector r as

$$r = \{0.03, 0.00, -0.03, 0.02, -0.02, -0.05, -0.03, 0.03, 0.03\}$$

and using $\zeta = 0.3$ we get $\tau_s = 20.797$.

6. This gives $S_1 = 0.7548$ and $S_2 = 0.0062$ and the ratio of the two is then calculated as $\frac{S_1}{S_2} = 123$ which is the new estimate for the pixel value of f_{ij} .

This process followed for every pixel (i, j) in the image and the result will then be the smoothed image.

3.2.3 LULU Smoothing

The concept of *LULU* smoothers was first introduced by Rohwer [47] in the late 1980s and Rohwer and Toerien [50] in the early 1990s. Rohwer then furthered the development of *LULU* theory in collaboration with Wild [51, 47] and worked on the implementation and application with Laurie [49, 36]. The *LULU* smoothers are a class of non-linear smoothers which are based on extreme order statistics. In recent years there has been a lot of theoretical work done around *LULU* smoothers. The statistical properties and distributions of *LULU* smoothers were derived and multiple comparisons were done with other smoothers [15, 45, 51, 14]. Until recently *LULU* smoothers were only considered for sequences in one dimension after which the theory was extended to the multidimensional space - preserving their essential statistical properties [4]. A smoother is idempotent if $P(Px) = Px$ and co-idempotent if $(I - P)((I - P)x) = Px$. It has been shown that *LULU* smoothers are both idempotent and co-idempotent [4], essential properties since the smoothers are nonlinear so these are not natural properties. This means that *LULU* smoothers are also consistent in the sense that they regard the output signal of the smoother as pure signal which will result in the same output if the smoother was applied again [48].

LULU Theory

We will give a brief discussion on the underlying *LULU* theory and later compare the *LULU* smoother to other smoothers in applications. First we discuss the smoothers in one dimension.

The minimum and maximum operators, \bigwedge and \bigvee , are defined as :

$$\bigvee x = \max \{x_i, x_{i+1}\}$$

and

$$\bigwedge x = \min \{x_{i-1}, x_i\}.$$

It is clear that the minimum operator is a backwards operator and the maximum operator is a forward operator. Upward impulses will be suppressed by the minimum operator while downward impulses will be suppressed by the maximum operator. The *LULU* operators are defined as follows:

$$L_n(x) = \bigvee^n \bigwedge^n x$$

and

$$U_n(x) = \bigwedge^n \bigvee^n x$$

where $(\bigvee^n x)_i = \max \{x_i, x_{i+1}, \dots, x_{i+n}\}$ and $(\bigwedge^n x)_i = \min \{x_{i-n}, x_{i-n+1}, \dots, x_i\}$.

Figures 3.6 to 3.9 show the result of the $\bigwedge^n, \bigvee^n, U_n, L_n, L_n U_n$ and $U_n L_n$ operators on the one dimensional sequence of numbers $x = \{10, 3, 8, 9, 4, 9, 7\}$. Note that zeros are appended to the left and the right of the sequences for the edge operators.

We now introduce the multidimensional *LULU* smoothers. Some work has been done to extend

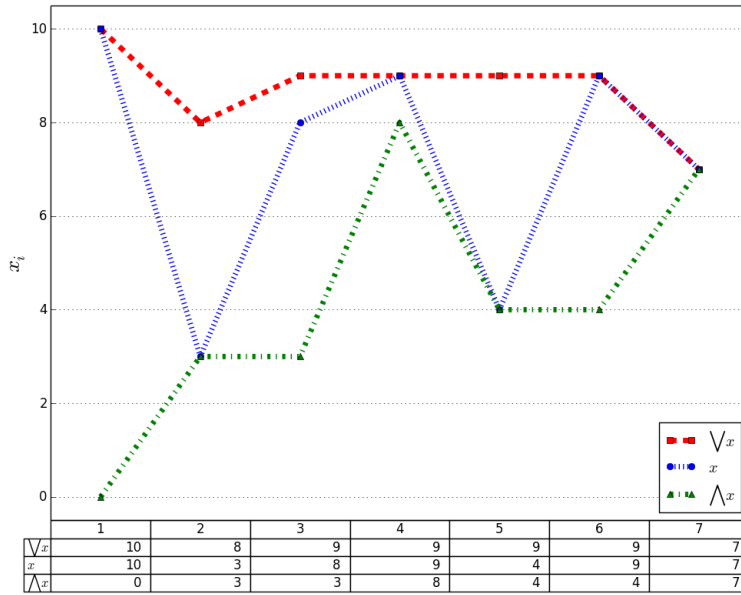


Figure 3.6: The result of the \vee and \wedge operators.

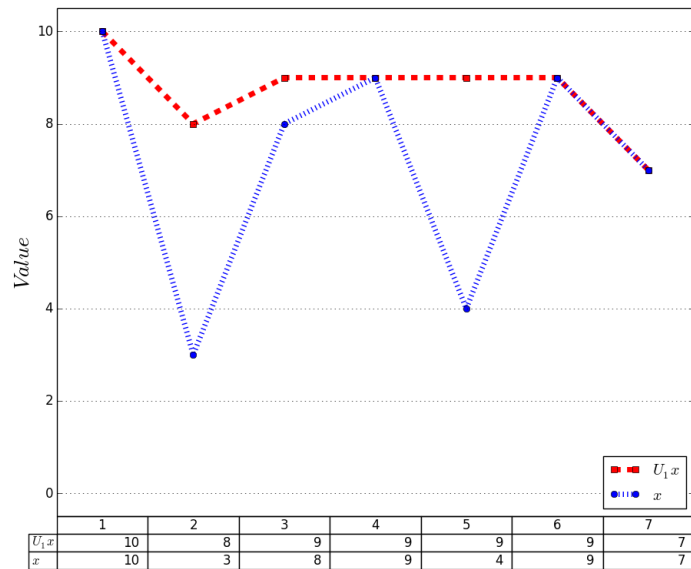


Figure 3.7: The result of the U_1 operator.

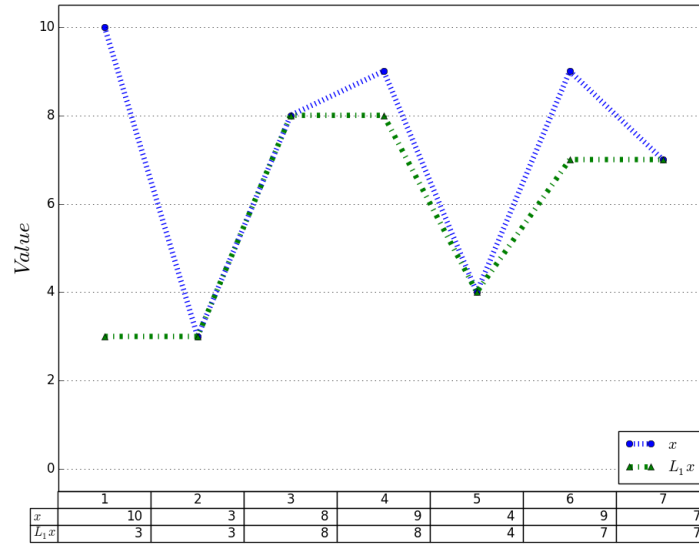


Figure 3.8: The result of the L_1 operator.

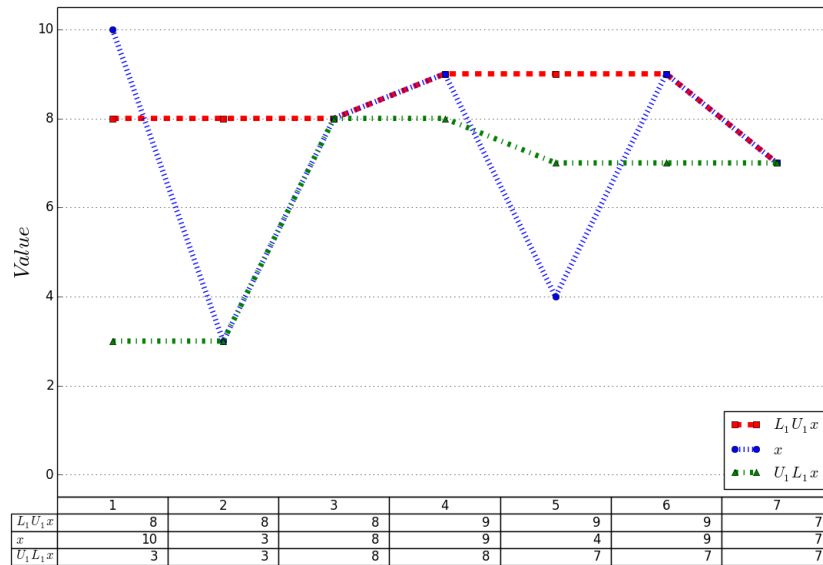


Figure 3.9: The results of the L_1U_1 and U_1L_1 operators.

LULU smoothers to multiple dimensions [17, 4]. The idea is to use the morphological concept of set connection [56] for the extension to multidimensions as \mathbb{Z}^d , $d \geq 2$, has no natural ordering.

Let $\mathcal{A}(\mathbb{Z}^d)$ denote the vector lattice for all functions defined on \mathbb{Z}^d . For $x \in \mathbb{Z}^d$, let $\mathcal{N}_n(x)$ be the set of all connected sets of size $n + 1$ that contain the point x where the connection used is define by Serra [56]. The multidimensional *LULU* operators are

$$L_n(f)(x) = \max_{V \in \mathcal{N}_n(x)} \min_{y \in V} f(y), \quad x \in \mathbb{Z}^d,$$

$$U_n(f)(x) = \min_{V \in \mathcal{N}_n(x)} \max_{y \in V} f(y), \quad x \in \mathbb{Z}^d$$

where $f \in \mathcal{A}(\mathbb{Z}^d)$ and $n \in \mathbb{N}$ [17, 4].

The local minimum and maximum set are defined as follows: Let V be a connected subset of \mathbb{Z}^d , then a V is a local maximum set of $f \in \mathcal{A}(\mathbb{Z}^d)$ if $\sup_{y \in \text{adj}(V)} f(y) < \inf_{x \in \text{adj}(V)} f(x)$. V is a local minimum set if $\inf_{y \in \text{adj}(V)} f(y) < \sup_{x \in \text{adj}(V)} f(x)$.

The L_n and U_n operators act on local maximum and minimum sets respectively. That is, L_n removes local maximums and U_n removes local minimums in f of size n or less. Operators L_n and U_n will never create new local minimum- or maximum sets where there were none before, but is it possible that they may extend minimum- or maximum sets by removing other minimum- or maximum sets. $L_n(f) = f$ and $U_n(f) = f$ if and only if there are no minimum- or maximum sets of size n or less in f [4], in addition *LULU* smoothers have been shown to be comparable and in some cases better than median smoothers [45, 30].

Example 3.2.4. The original image is shown in Figure 3.10a. Local max sets of size 1 shown in Figure 3.10a are marked and will be the target of the L_0 smoother. The results of the L_0 smoother and the U_0 smoother is given in Figures 3.10b and 3.10d respectively. Figure 3.10c also shows local max sets of size 2 on which L_1 will be applied as shown in 3.10d. After applying U_1 , we can see in Figure 3.10e that there are no local max sets of size 3 but rather of size 4 which means that neither L_2 or U_2 need be applied. We continue to apply L_3 and obtain the results as can be seen in Figure 3.10f which also shows that the smallest sized local max set left is a set is of size 6. After applying L_5 , as in Figure 3.10g, we are left with a constant valued image and further smoothing will have any affect. In practice we will seldom apply *LULU* smoothers until we reach a constant image, this is only done to obtain the DPT [4]. *LULU* smoothers are usually applied until a satisfactory convergence of smoothness is reached - which can be measured using the total variation of the pixel values. That is, if δ is some convergence criterion parameter then $L_n U_n(f_{NI})$ will be applied to the noisy image f_{NI} until the following condition is true

$$TV(L_n U_n(f_{NI})) - TV(L_{n+1} U_{n+1}(f_{NI})) < \delta.$$

We will denote the *LULU* smoother with a convergence criterion δ by $LULU_{\text{conv.}}$.

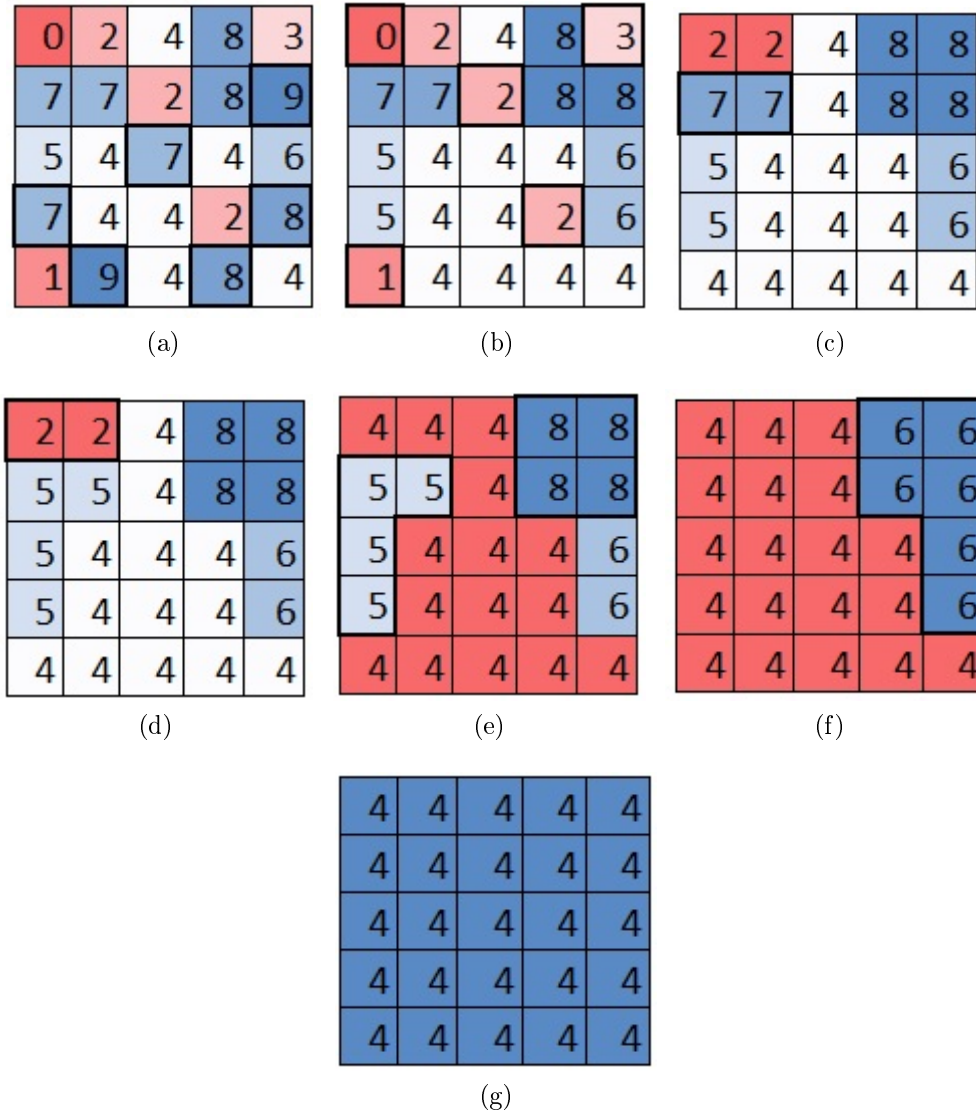


Figure 3.10: a.) The original image f with marked local maximum connected components of size 1 w.r.t. 4-connectivity. b.) After applying $L_0(f)$ with marked local minimum connected components of size 1 w.r.t. 4-connectivity. c.) After applying $U_0L_0(f)$ with marked local maximum connected components of size 2 w.r.t. 4-connectivity. d.) After applying $L_1U_0L_0(f)$ with marked local maximum connected components of size 2 w.r.t. 4-connectivity. e.) After applying $U_1L_1U_0L_0(f)$ with marked local maximum connected components of size 4 w.r.t. 4-connectivity. f.) After applying $L_3U_1L_1U_0L_0(f)$ with marked local maximum connected components of size 6 w.r.t. 4-connectivity. g.) After applying $L_5L_3U_1L_1U_0L_0(f)$.



Figure 3.11: An example of a $V \in J^{(2)}$ connected set.



Figure 3.12: An example of $\mathcal{N}_1(i, j)$ and $\mathcal{N}_1(i, j + 1)$ assuming 4-connectivity.

3.2.4 Connected Median Image Smoother

The same connected components (constant sets) used by the *LULU* operators can be used to adjust the adaptive median filter to use connected components instead of the normal box SE [17]. The algorithm will change as follows:

Smoothing Algorithm 4 (This is a new algorithm proposed by the author)

Define the set

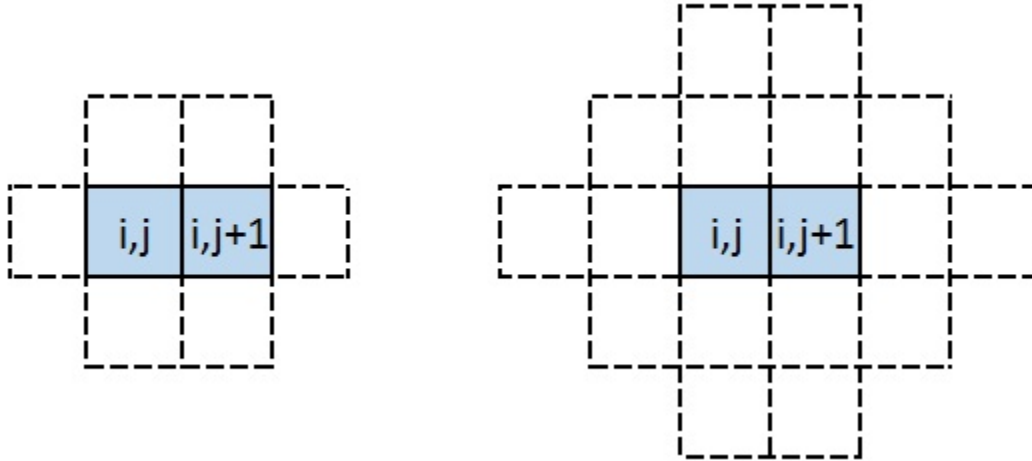
$$J^{(p)} = \{V : V \in \mathcal{C}, f_x = f_y \forall x, y \in V, \text{card}(V) = p\}$$

of connected level sets of size p of the original image and the sets $B_V^{(n)} = \bigcup_{(i,j) \in V \in J^{(p)}} \bigcup_{K \in \mathcal{N}_n(i,j)} K$, $n =$

$1, \dots, n_{\max}$ where n_{\max} is selected by the user ($n_{\max} = 3$ works well). In Figure 3.11 we show an example of what one of the V sets in $J^{(2)}$ may look like and Figure 3.12 shows how \mathcal{N}_1 looks over the pixels in V according to the definition in Section 3.2.3. When we take the union of all the pixels covered by the \mathcal{N}_n 's over all the pixels of a V in $J^{(p)}$ we will get the set $B_V^{(n)}$ which is shown in Figure 3.13.

We start with $p = 1$

1. If $J^{(p)} \neq \emptyset$, continue to step 2.
 Else if $p < p_{\max}$ then $p = p + 1$ ($5 \leq p_{\max} \leq 10$ works well) and repeat step 1.
 Else exit.
2. For each set $V \in J^{(p)}$:
 - (a) Set $n = 1$.
 - (b) Obtain $B_V^{(n)}$
 - (c) Calculate $\min(f_{B_V^{(n)}}) = \text{minimum} \{ f_{ij} : (i, j) \in B_V^{(n)} \}$,
 $\text{med}(f_{B_V^{(n)}}) = \text{median} \{ f_{ij} : (i, j) \in B_V^{(n)} \}$ and
 $\max(f_{B_V^{(n)}}) = \text{maximum} \{ f_{ij} : (i, j) \in B_V^{(n)} \}$


 Figure 3.13: An example of $B_V^{(n)}$ for $n = 1$ and $n = 2$.

- (d) If $\min(f_{B_V^{(n)}}) < \text{med}(f_{B_V^{(n)}}) < \max(f_{B_V^{(n)}})$,
 proceed to step 2(f).
 Else proceed to step 2(e).
 - (e) If $n \leq n_{\max}$, set $n = n + 1$, repeat from step 2(b).
 Else set each $f_{ij} = \text{med}(f_{B_V^{(n)}}) \forall (i, j) \in V$.
 - (f) If $\min(f_{B_V^{(n)}}) < f_{ij} < \max(f_{B_V^{(n)}})$, we assume f_{ij} is a connected component containing signal and leave the pixel value as is, else replace each $f_{ij} \in V$ with the new value $\text{med}(f_{B_V^{(n)}})$.
3. If $p < p_{\max}$ then set $p = p + 1$, repeat from step 1.
 Else exit.

Instead of running the algorithm until p_{\max} is reached, one could also specify a total variation convergence criterion and exit the algorithm if either p_{\max} is reached or the difference in total variation of the image after two consecutive iterations is less than such convergence criterion specified in step 1 and step 3. That is exit if

$$TV(f_{NI_i}) - TV(f_{NI_{i+1}}) < \delta$$

where δ is the convergence criterion for the total variation of two consecutive iterations and f_{NI_i} is the smoothed image after iteration i . Example 3.2.5 gives an overview of how the algorithm works in practice. The algorithm was implemented in Python. A 5×6 image is used and the algorithm is applied with parameters $p = 10$ and $n = 5$. First the original image is given and then the connected sets are shown where each pixel of a connected set is the same number starting at 0. Next a list is given of all the unique numbers, identifying each connected set, which contains a certain number of pixels. Each set of pixels, containing p number of pixels, in the aforementioned list will then be evaluated as in step 2 of the algorithm above starting with $n = 1$. In the example, in order to obtain $B_V^{(n)}$, we first obtain a binary mask identifying each V contained in $J^{(p)}$. We perform a binary dilation on this mask to obtain another binary mask which can be multiplied

element wise with f_{NI} in order to obtain $B_V^{(n)}$ in step 2(b). Step 2(c) is then performed where all the necessary statistics of $B_V^{(n)}$ is calculated and it is decided if n should increase, f_{ij} should stay the same or f_{ij} should change. Next p is increased and the process continues until p reaches its maximum value.

Example 3.2.5.

Original image:

```
[[ 9  9  8  2  6  6]
 [ 9  9  2 10  8  6]
 [10 10 10 10 10  3]
 [ 6  3  2 10  2 10]
 [ 6  6  4  9 10 10]]
```

For $p = 1$ Connected sets matrix:

```
[[ 0  0  1  2  3  3]
 [ 0  0  4  5  6  3]
 [ 5  5  5  5  5  7]
 [ 8  9 10  5 11 12]
 [ 8  8 13 14 12 12]]
```

Connected sets of size 1 :

```
[1, 2, 4, 6, 7, 9, 10, 11, 13, 14]
```

In the output above we see the pixel intensities of the original image and in the second matrix we show which of the pixels form connected level sets. Our algorithm start with $p = 1$ for which the last part shows which of the level sets are of size 1. Now that we know which level sets we need to evaluate, we can set $n = 1$ and continue the algorithm.

for $n = 1$

Dilated binary mask applied to image:

```
[[0 9 8 2 0 0]
 [0 0 2 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]
```

List of pixel values covered by the mask:

```
[9 8 2 2]
```

f_{ij} : 8

K_{min} : 2

K_{med} : 5

K_{max} : 9

f_{ij} did not change.

We start evaluating the first V in $J^{(1)}$ marked as 1 in connected sets matrix obtained in the output above. $B_V^{(1)}$ is then obtained as described above, we calculate the necessary statistics and according

to our algorithm the intensity level of this V should not change as it probably contains signal. We continue:

```
for n = 1
```

Dilated binary mask applied to image:

```
[[ 0 0 8 0 0 0]
 [ 0 9 2 10 0 0]
 [ 0 0 10 0 0 0]
 [ 0 0 0 0 0 0]
 [ 0 0 0 0 0 0]]
```

List of pixel values covered by the mask:

```
[ 8 9 2 10 10]
```

```
f_ij: 2
```

```
K_min: 2
```

```
K_med: 9
```

```
K_max: 10
```

```
f_ij changed!
```

```
New f_ij = 9
```

We provide more example output for $p = 1$ and $n = 1$, but this time the algorithm found that the intensity level of V should change as it is probably noise.

```
for n = 1
```

Dilated binary mask applied to image:

```
[[ 0 0 0 0 0 0]
 [ 0 0 0 0 0 0]
 [ 0 0 0 0 10 0]
 [ 0 0 0 10 2 10]
 [ 0 0 0 0 10 0]]
```

List of pixel values covered by the mask:

```
[10 10 2 10 10]
```

```
f_ij: 2
```

```
K_min: 2
```

```
K_med: 10
```

```
K_max: 10
```

```
for n = 2
```

Dilated binary mask applied to image:

```
[[ 0 0 0 0 0 0]
 [ 0 0 0 0 8 0]
 [ 0 0 0 10 10 3]
 [ 0 0 2 10 2 10]
 [ 0 0 0 9 10 10]]
```

List of pixel values covered by the mask:

```
[ 8 10 10 3 2 10 2 10 9 10 10]
```

f_ij: 2

K_min: 2

K_med: 10

K_max: 10

for n = 3

Dilated binary mask applied to image:

```
[[ 0 0 0 0 6 0]
 [ 0 0 2 10 8 6]
 [ 0 0 10 10 10 3]
 [ 0 3 2 10 2 10]
 [ 0 0 4 9 10 10]]
```

List of pixel values covered by the mask:

```
[ 6 2 10 8 6 10 10 10 3 3 2 10 2 10 4 9 10 10]
```

f_ij: 2

K_min: 2

K_med: 8

K_max: 10

f_ij changed!

New f_ij = 8

In the iterations above n had to be increased as there weren't enough new information in the immediate vicinity to determine if the pixels in V contained noise or signal. The process continued until n reached 3 which eventually lead to a change in the pixel intensity level of the pixels in V .

For p = 3 Connected sets matrix:

```
[[ 0 0 1 2 3 3]
 [ 0 0 0 4 5 3]
 [ 4 4 4 4 4 6]
 [ 7 8 9 4 10 11]
 [ 7 7 9 12 11 11]]
```

Connected sets of size 3 :

```
[3, 7, 11]
```



```
for n = 1
```

```
Dilated binary mask applied to image:
```

```
[[0 0 0 7 6 6]
 [0 0 0 0 8 6]
 [0 0 0 0 0 8]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]
```

```
List of pixel values covered by the mask:
```

```
[7 6 6 8 6 8]
f_ij: 6
K_min: 6
K_med: 6
K_max: 8
f_ij changed!
New f_ij = 6
```

The last piece of the output shows an iteration where $p = 3$ and how the new intensity level for the pixels in V came to be. The idea for this algorithm was to combine some elements from the *LULU* smoother with a decision based algorithm in order to obtain the robustness of the *LULU* smoother and the detail preservation characteristics of a decision based algorithm. In section 3.3 we will look at how this new image smoothing methods compares to other established methods.

3.3 Algorithm Implementation

We now compare the different smoothing techniques in terms of how well they eliminate image noise, as defined in Section 3.2, and preserve image signal. Table 3.1 shows the peak signal to noise ratio results of an image containing four different polygons and an image of buildings after adding different densities of salt & pepper noise and speckle noise and then applying either the connected median-, median-, adaptive median-, L_1U_1 -, $LULU_{conv}$. and robust statistic estimation filter. It is clear that the connected median filter compares well with the other filters and in some cases even surpass them. It is clear that the PSNR, as defined in Section 3.2, decreases for all smoothers as the different noise levels increase. We also investigated the smoothing effect of the *LULU* smoother if just L_1U_1 was applied and from the table it can be seen that it outperformed one of the other filters on the polygon image where speckle noise was present. This was done to illustrate the amount of noise that get removed from the initial steps in the $LULU_{conv}$. smoother. Figures 3.14, 3.15 and 3.16 shows the resultant image after applying the various filters to the polygon image after 10%, 20% and 50% salt & pepper noise was added respectively. Figures 3.17, 3.18 and 3.19 shows resultant image after applying the various filters to the buildings image after 10%, 20% and 50% salt & pepper noise was added respectively. Figures 3.20, 3.21 and 3.22 shows resultant image after applying the various filters to the polygon image after 10%, 25% and 50% speckle noise was added respectively. Figures 3.23, 3.24 and 3.25 shows resultant image after applying the various filters to the Lena image after 10%, 25% and 50% speckle noise was added respectively. It was seen that the median smoother performed best in the building image

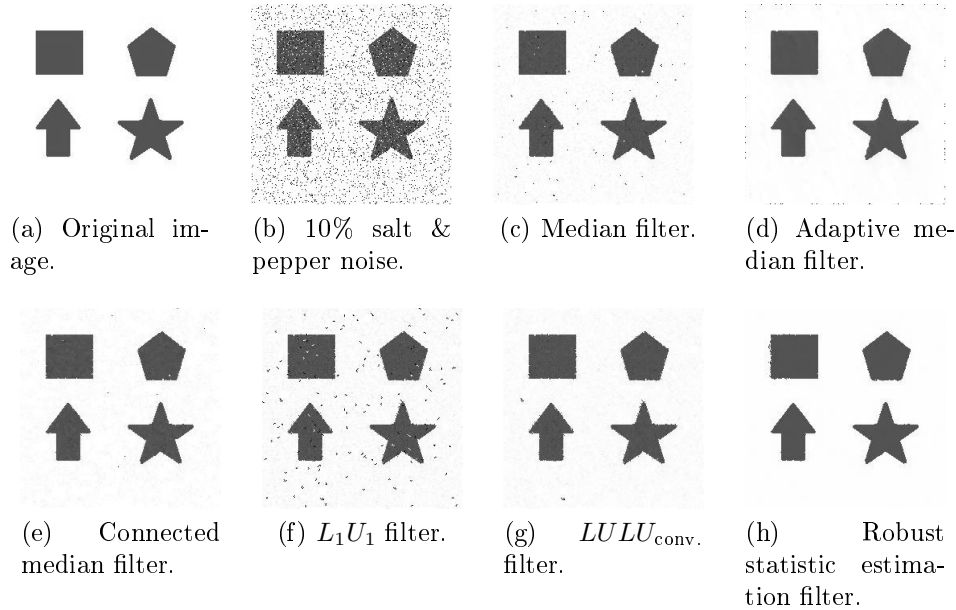


Figure 3.14: Example 10% salt & pepper noise being smoothed by smoothers.

with the adaptive median smoother and the robust statistic estimation filter not far behind. The adaptive median smoother performed best on the polygon image followed by the connected median filter which performed very well on images where speckle noise was present. The robust statistic estimation algorithm performed very well when salt & pepper noise was present, but it was to be expected since extreme noise is removed in the algorithm itself - speckle noise does not present itself under the same extreme pixel intensity values and so the results are considerably worse when noise intensity levels follow a more gradual distribution. The difference in performance of the median filter on the two different images shows that the content of the image plays a big part the ability of an image smoother to remove image noise. The connected median filter performed better in the polygon toy image and the $LULU_{conv.}$ image smoother provided relative consistent performance on both images.

Peak Signal Noise Ratio (Image = Buildings)	Smoothed PSNR												Smoothed PSNR Ranked											
	S&P						Speckle						S&P						Speckle					
	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	25%	50%	Average Rank		
Filter/Noise	38.17	37.32	16.94	38.65	36.55	34.17	2	2	3	2	2	3	1	1	1	1	1	1	1	1	1	1	1.67	
Median	32.39	30.34	18.50	38.45	35.41	32.86	5	4	2	5	4	2	2	2	2	2	2	2	2	2	2	2	2.83	
Adaptive Median	34.23	28.88	11.57	36.71	33.25	29.89	4	5	5	4	5	5	3	3	4	3	3	3	3	3	3	4	4.00	
Connected Median	35.25	33.17	14.36	35.34	32.72	30.04	3	3	3	3	4	4	5	4	3	4	3	3	4	4	3	3	3.67	
LULU	28.24	19.68	8.68	32.82	27.10	23.39	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6.00	
L1U1	44.19	43.59	39.43	35.74	30.66	26.18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2.83	
Robust Statistic Estimation																								

Peak Signal Noise Ratio (Image = Polygons)	Smoothed PSNR												Smoothed PSNR Ranked											
	S&P						Speckle						S&P						Speckle					
	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	25%	50%	Average Rank		
Filter/Noise	26.69	19.70	6.81	28.22	24.58	21.38	5	5	5	5	5	5	4	3	4	4	4	4	4	3	3	4	4.33	
Median	28.47	25.25	13.01	30.16	27.42	24.90	3	3	3	3	3	3	2	1	1	1	1	1	1	1	1	1	1.83	
Adaptive Median	30.64	25.03	8.55	29.18	25.87	22.93	2	4	4	2	4	4	2	2	2	2	2	2	2	2	2	2	2.67	
Connected Median	28.21	25.91	8.81	28.39	24.50	22.32	4	2	3	4	3	3	3	4	3	3	4	3	3	4	3	3	3.17	
LULU	21.63	14.50	5.40	25.51	21.66	18.62	6	6	6	6	6	6	6	5	5	5	5	5	5	5	5	5	5.50	
L1U1	30.80	30.26	24.52	24.48	20.19	16.96	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3.50	
Robust Statistic Estimation																								

Table 3.1: Peak signal to noise ratio (PSNR) of images after smoothing.

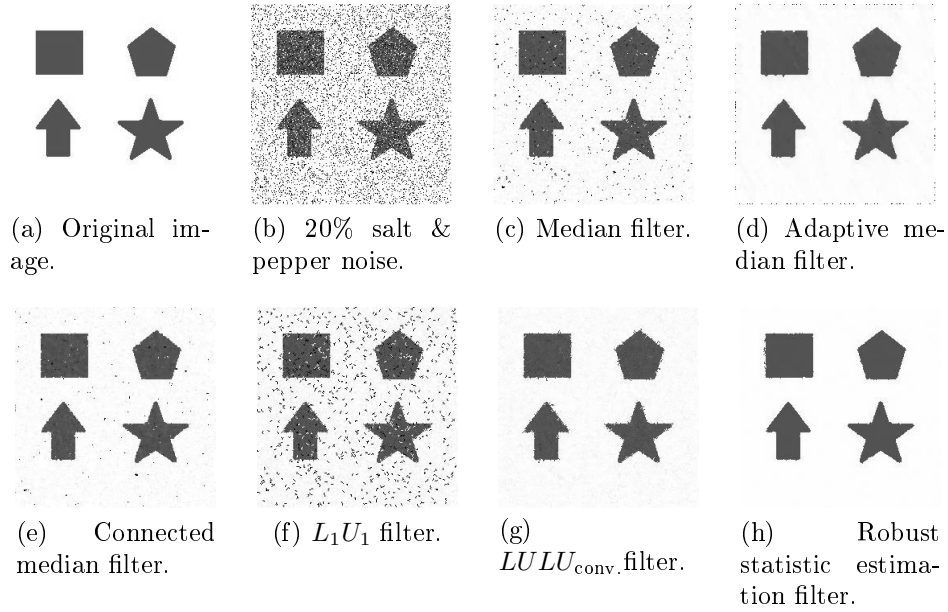


Figure 3.15: Example 20% salt & pepper noise being smoothed by smoothers.

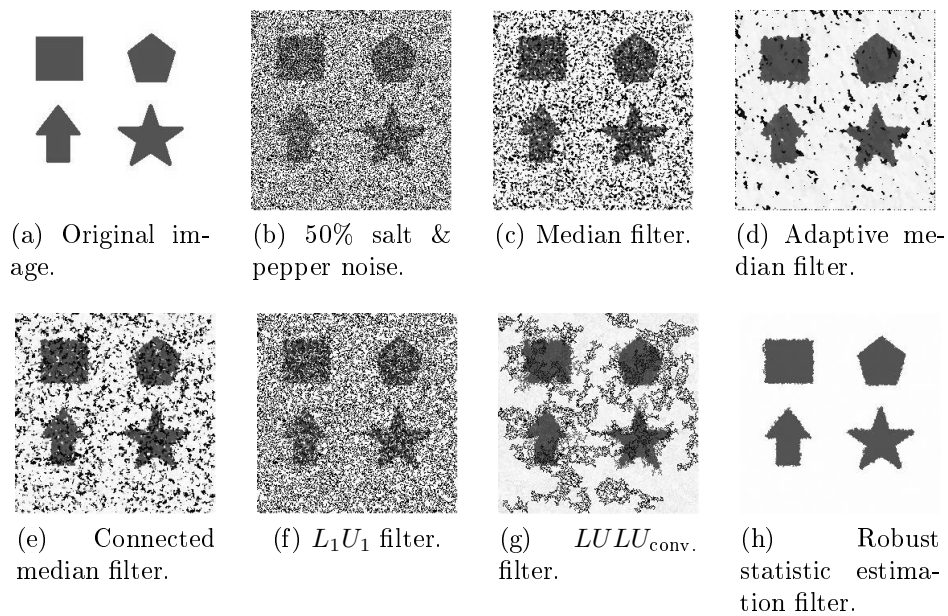


Figure 3.16: Example 50% salt & pepper noise being smoothed by smoothers.

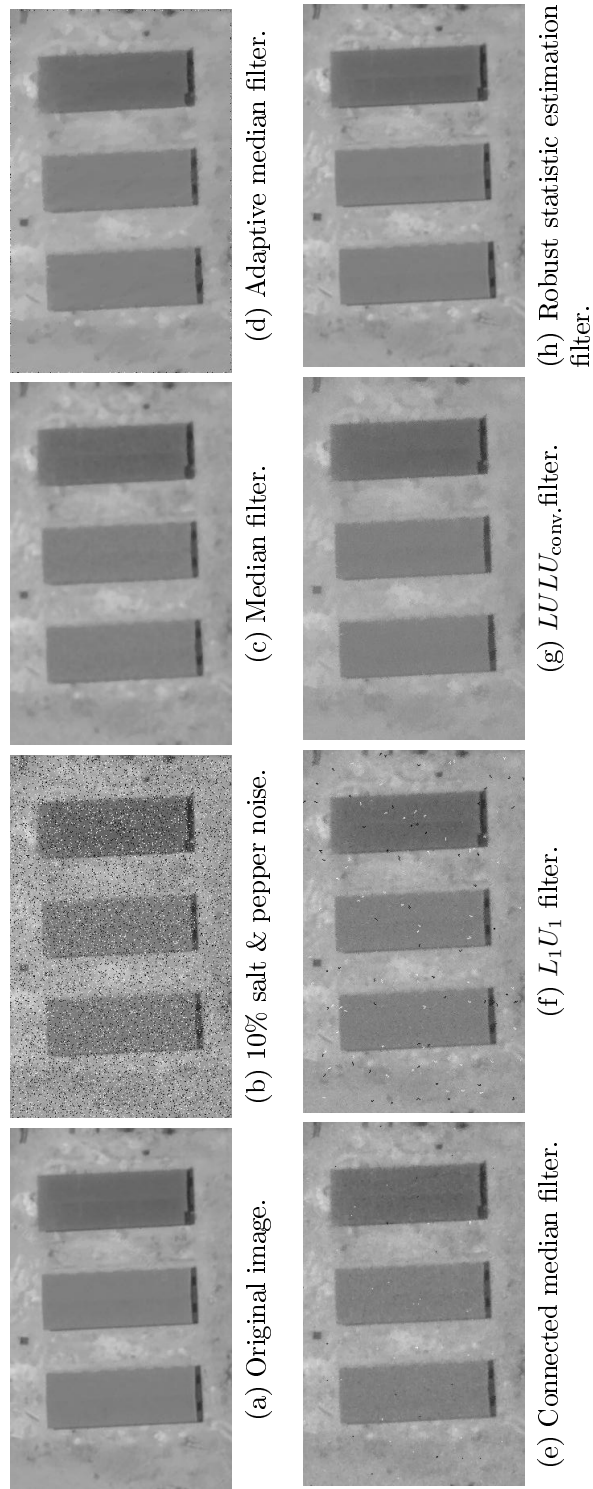


Figure 3.17: Example 10% salt & pepper noise being smoothed by smoothers.

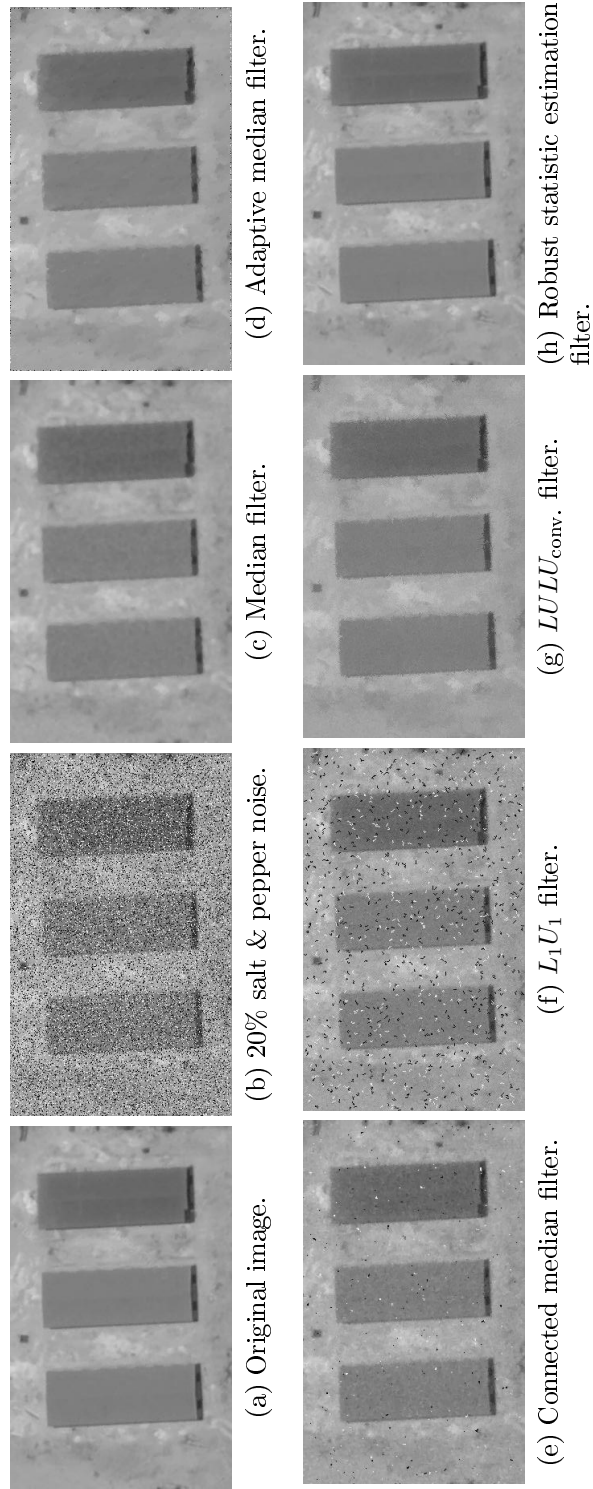


Figure 3.18: Example 20% salt & pepper noise being smoothed by smoothers.

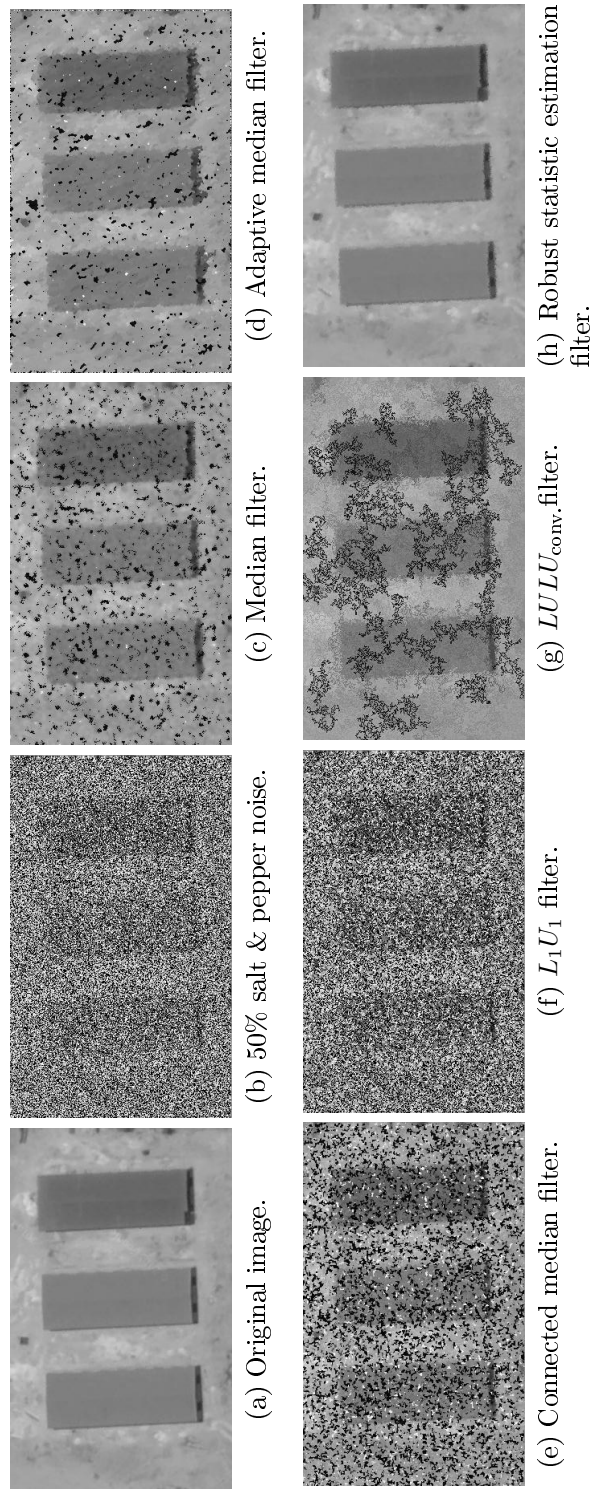


Figure 3.19: Example 50% salt & pepper noise being smoothed by smoothers.

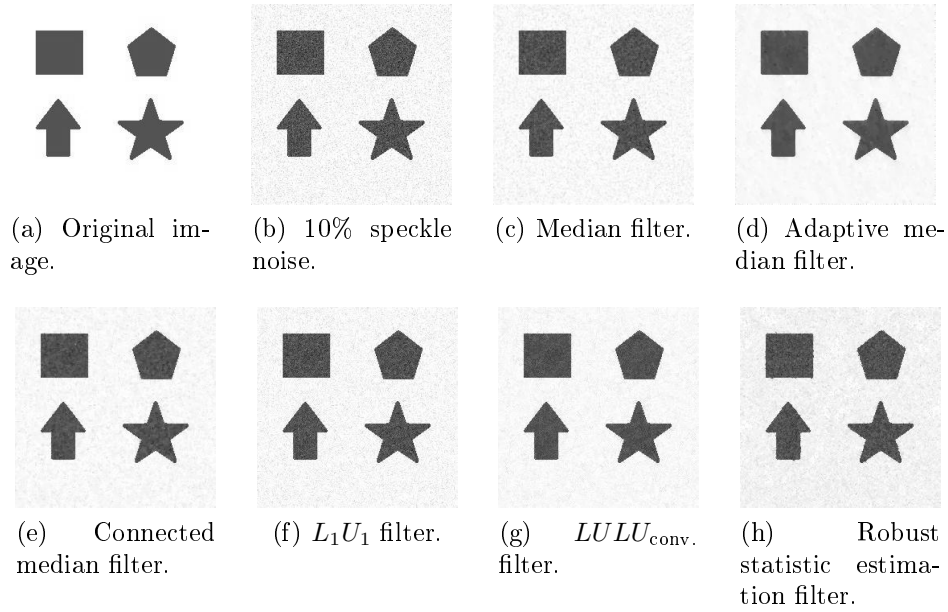


Figure 3.20: Example 10% speckle noise being smoothed by smoothers.

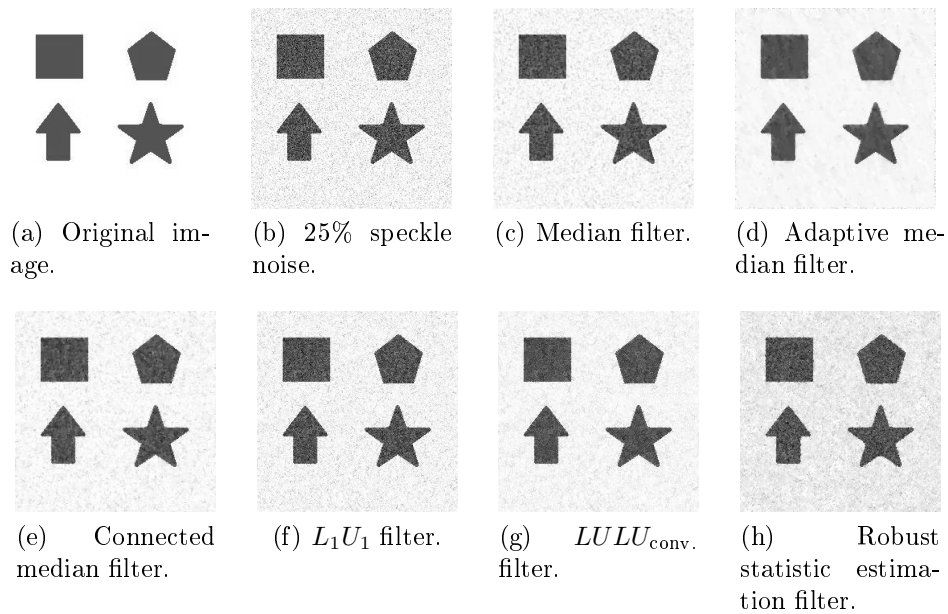


Figure 3.21: Example 25% speckle noise being smoothed by smoothers.

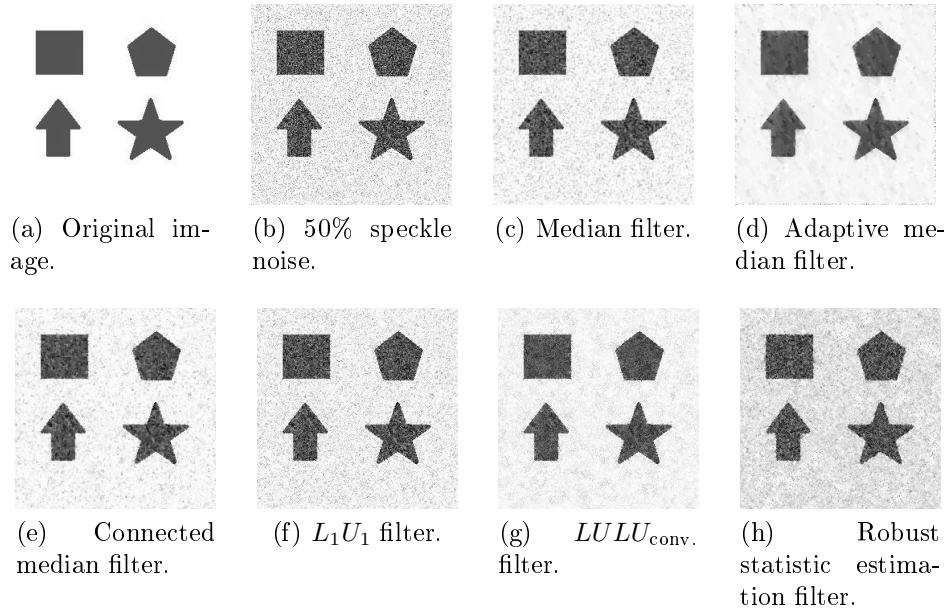


Figure 3.22: Example 50% speckle noise being smoothed by smoothers.

3.4 Contour Detection

3.4.1 A Border Following Method

Border following has been widely used in the image processing field and many variations have been proposed [52, 43, 59, 21]. The process of border following can be simply put as scanning an image for object borders and providing the output as a set of contour chains describing the outline of those objects. There have been many applications of border following including object detection [23, 69], gesture recognition [34], motion tracking [18], texture application [68] and topological analysis [35].

A path in image f that joins two pixels $a, b \in f$ is a continuous mapping $g : [0, 1] \rightarrow f$ such that $g(0) = a$ and $g(1) = b$ and the path length is the number of number of pixels on the path. In general border following techniques are based on a raster scan of an image f using a 4- or 8-connected structured element S centered at some pixel (i, j) . A counterclockwise search of S , starting at the top middle pixel or the pixel counterclockwise from the previous border point found, is then performed to get the next border point. S is then centered at the new border point and the process continues until S is again centered at the first border point found and guaranteed convergence is reached with an 8-connected SE [67].

Here follows the proposed border following algorithm for an image that has been thresholded:

1. Start with a raster scan of the binary image until the first 1 is found and mark it as the first border pixel.
2. Using a 4-connectivity SE centered at the last border point found, start searching for the next 1 starting from the pixel counterclockwise from the previous border point found or the top pixel if this is the first border point and do this in a counterclockwise manner.

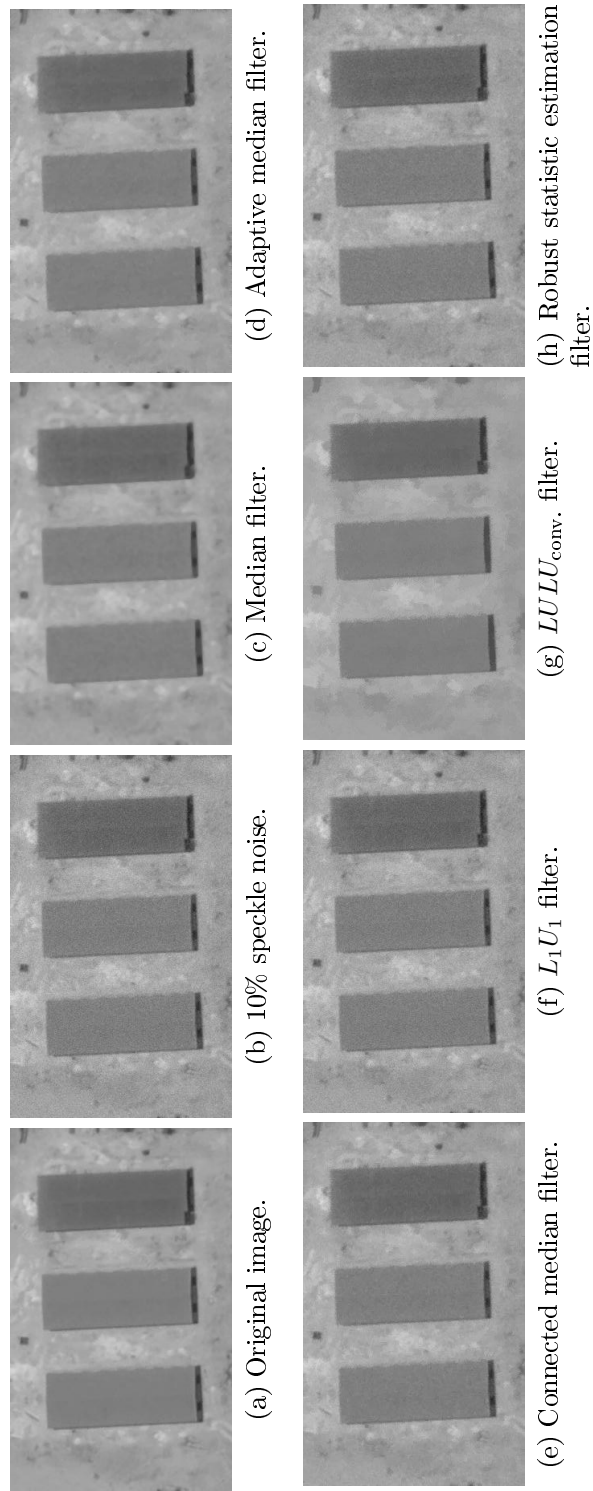


Figure 3.23: Example 10% speckle noise being smoothed by smoothers.

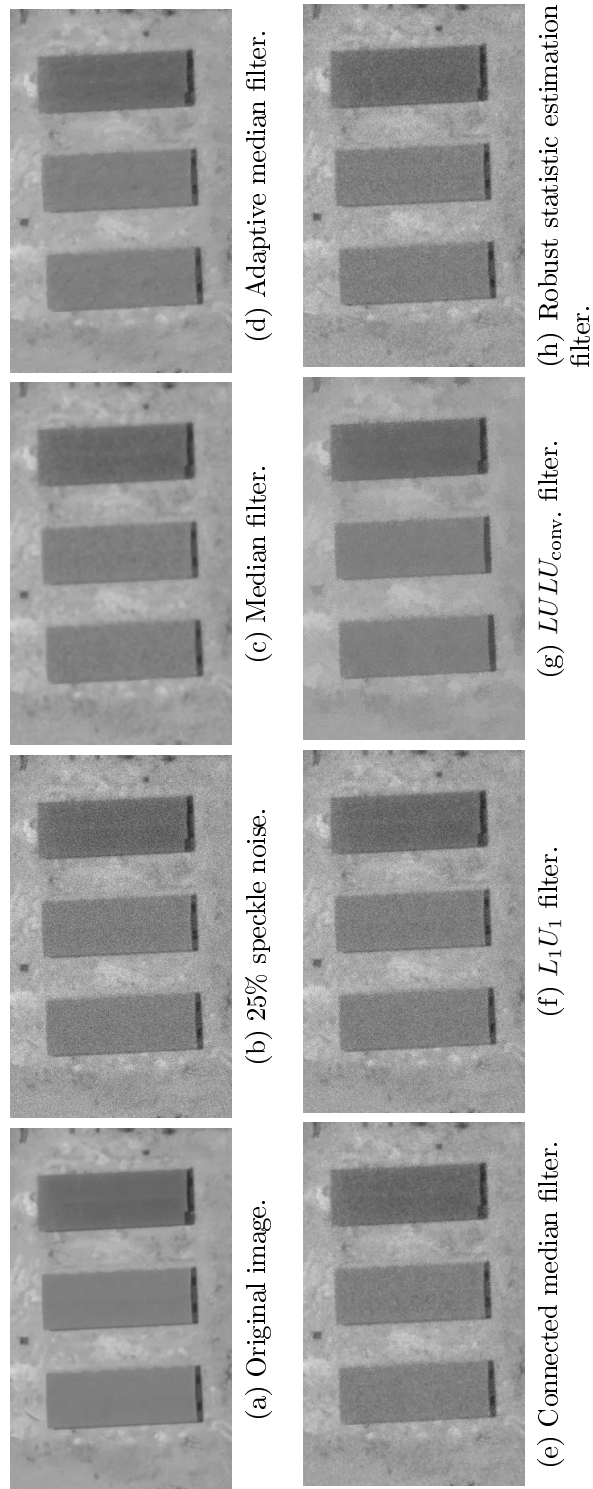


Figure 3.24: Example 25% speckle noise being smoothed by smoothers.

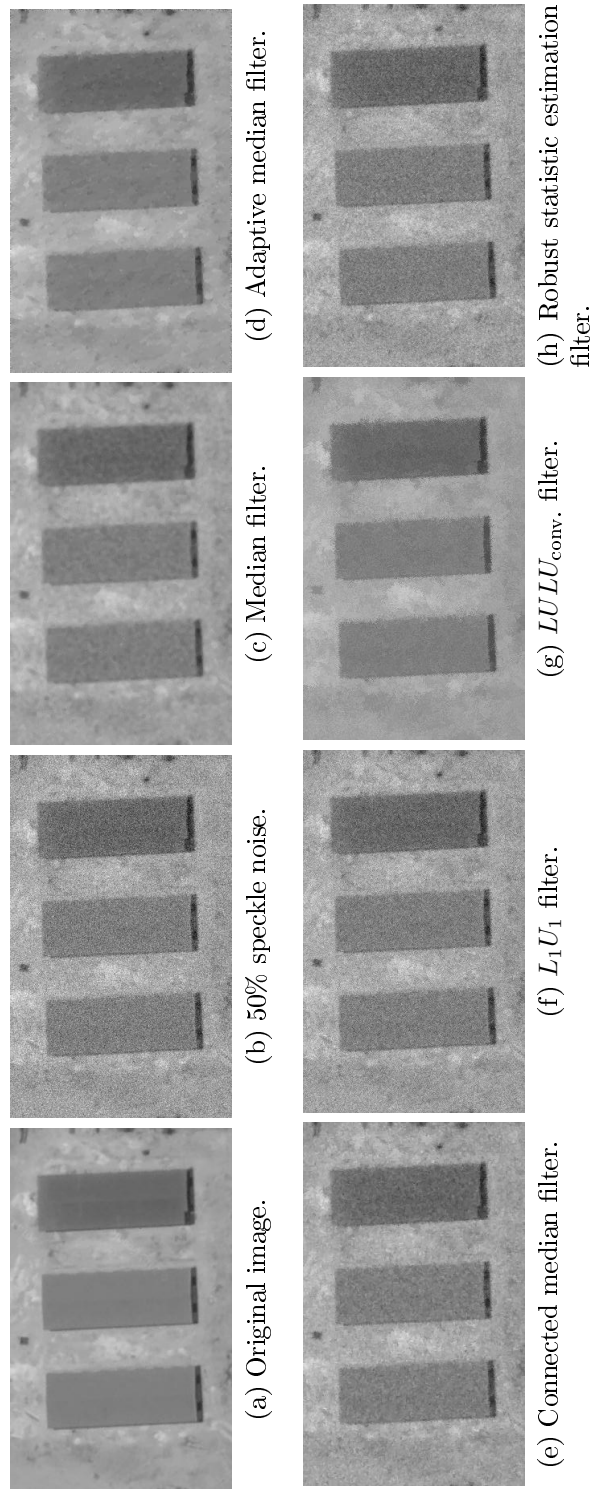


Figure 3.25: Example 50% speckle noise being smoothed by smoothers.

If a new border pixel was found,
then move to the new border pixel and repeat step 2,
else if
the border pixel found is the first border pixel,
then the border is complete and the algorithm will terminate,
else if no border points were found repeat step 2. using 8-connectivity.

Example 3.4.1. Figure 3.26a shows the original binary image and the first pixel found, using a raster scan, to be on a border i.e. of value 1 (foreground). Using a 4-connected and then 8-connected SE (if there are no border pixels found using 4-connectivity) we then search for the next border pixel starting at the top middle pixel of the SE in a counterclockwise direction. The next border pixel was found below the first border pixel and is shown in Figure 3.26b. We start the next search from the previous border pixel found in a counterclockwise direction and find the next border pixel in Figure 3.26c. Continuing in this way we find all the border pixel and show the complete set of border pixels in Figure 3.26d. The order in which the border pixels were found is given in Figure 3.27.

Chain codes can be used to summaries and compress the values of a border [29]. We assign a value (direction) to every connected pixel according to the SE in Figure 3.28 for every border point found. We can store the direction of the next border pixel in a list called chain codes and be able to reconstruct the complete border using that list and the location of the first border pixel found. The SE mapped with the direction is given in Figure 3.28 and the chain codes in Figure 3.29.

The border following method that we use is one that was proposed by Suzuki and Abe in 1985 [59] for binary images. This method uses a unique naming convention in the algorithm to distinguish the order of the border pixels found. For greyscale images the technique can be used once the image is thresholded.

3.4.2 Contour Detection Using Watersheds

In the late 1970s Beucher and Lantuejoul wrote a paper on the use of watersheds in contour detection [7, 6]. This paper describes the nonparametric method for contour extraction in greyscale images using watersheds and formed the basis for a great amount of research that followed [9, 40, 44, 55].

The method for finding contours using watersheds can be explained as follows:

Watershed Algorithm Explanation

For a greyscale image, find the regional minimums of the intensity plot of that image. Place a water source at each of those regional minimums and fill the topographic relief, using those water sources, while erecting barriers or dam walls so that the water from the different sources never touch. The resulting segmentation is then given by the erected barriers. This process is illustrated in Figure 3.30.



Figure 3.26: a.) The original image with the first border point found using a raster scan. b.) From the first border point the next border point is found. c.) The next border point is found the counterclockwise search. d.) The final border is given after all the border points are found.

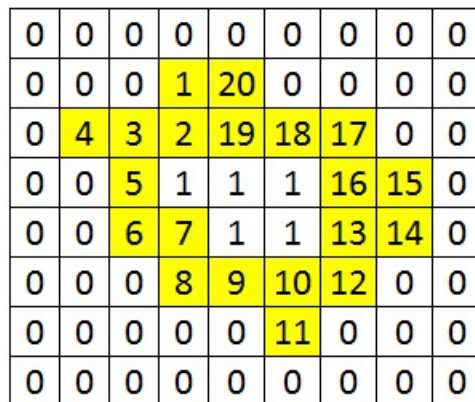


Figure 3.27: The indexes of the chain codes indicated on each border pixel that was found.

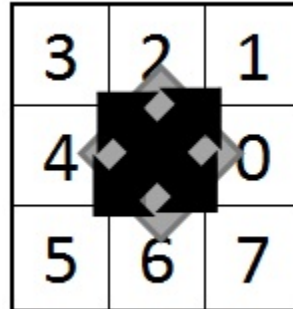


Figure 3.28: The structuring element indicating the direction of the next border point used to decipher border chain codes.

Index	Direction	Index	Direction
1	6	11	1
2	4	12	2
3	4	13	0
4	7	14	2
5	6	15	4
6	0	16	2
7	6	17	4
8	0	18	4
9	0	19	2
10	6	20	4

Figure 3.29: The list of chain codes of the border found.

We will now give a brief theoretical overview of the watershed transform. Let $x, y \in G$ where G is a set of pixels. The geodesic distance $d_G(x, y)$ between x and y is the minimum path length among all paths within G from x to y . Let $H = \{H_1, \dots, H_k\}$ be a subset of G partitioned into k connected components then

$$d_G(x, H) = \min_{y \in H} \{d(x, y)\}.$$

We can now write the geodesic influence zone of the set H_i in G as

$$IZ_G(H_i, H) = \{t \in G \mid d_G(t, H_i) < d_G(t, H \setminus H_i)\}$$

and

$$IZ_G(H) = \bigcup_{i=1}^k IZ_G(H_i, H).$$

The pixels in G that do not belong to any geodesic influence zone form the *skeleton by influence zones* (SKIZ) of H in G or $SKIZ_G(H)$ and is given by

$$SKIZ_G(H) = G / IZ_G(H)$$

where $/$ stands for set difference.

Let $T_h(f)$ be the set of pixels in $f : \mathbb{Z}^2 \rightarrow \mathbb{Z}$ where $f(t)$, the greyscale image intensity of f at pixel t , is less than or equal to h and is defined as

$$T_h(f) = \{t \in \mathbb{Z}^2 : f(t) \leq h\}.$$

$T_h(f)$ is also known as the *threshold set* of f at level h .

We define a recursion over h that ranges from h_{\min} up to h_{\max} which will increase the size of the connected sets of pixels in $T_h(f)$ as h increases. If \min_h is the union of all regional minima at intensity level h then the first two iterations of the watershed transform algorithm are given as follows

1. $X_{h_{\min}} = T_{h_{\min}}(f)$
2. $X_{h_{\min}+1} = \min_{h_{\min}+1} \bigcup IZ_{T_{h_{\min}+1}(f)}(X_{h_{\min}})$

The final set of catchment basins obtained using this watershed algorithm is then given by $X_{h_{\max}}$ and can be obtained in general by the following algorithm [8]:

1. $X_{h_{\min}} = T_{h_{\min}}(f)$
- $\forall h \in [h_{\min}, h_{\max} - 1], X_{h+1} = \min_{h+1} \bigcup IZ_{T_{h+1}(f)}$

An effective algorithm that calculates the watershed contours was developed by Vincent and Soille [63] and is based on the immersion process analogy. This algorithm was proven to be faster and more accurate than other watershed algorithms. The improvement in speed was mainly due to an efficient way that was developed to only use information from pixels in the immediate vicinity to evaluate pixels and not use all the information in the image to process each iteration.

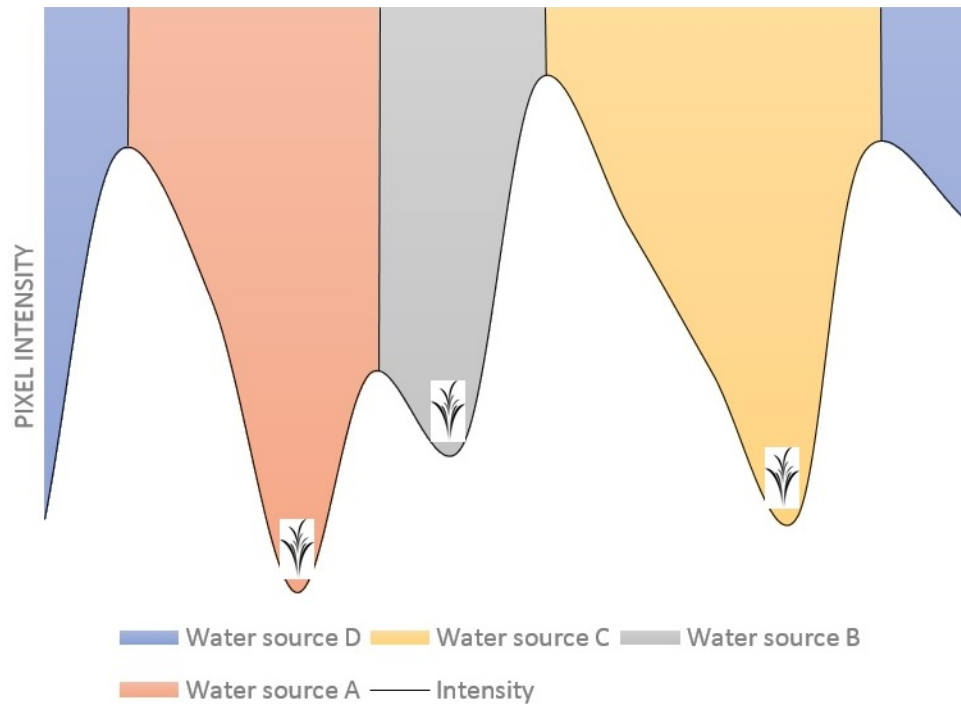


Figure 3.30: An illustration of how the watershed algorithm works.

Example 3.4.2.

In order for us to apply the watershed algorithm, we first need to identify the markers or water sources together with a suitable topography of the image. We do a kind of edge detection to detect potential borders of objects. This can be accomplished by obtaining the gradient image of the original image and thresholding it. We then identify level areas in the thresholded image and use them, together with another gradient image, to run the watershed algorithm. This process is shown in Figure 3.31.

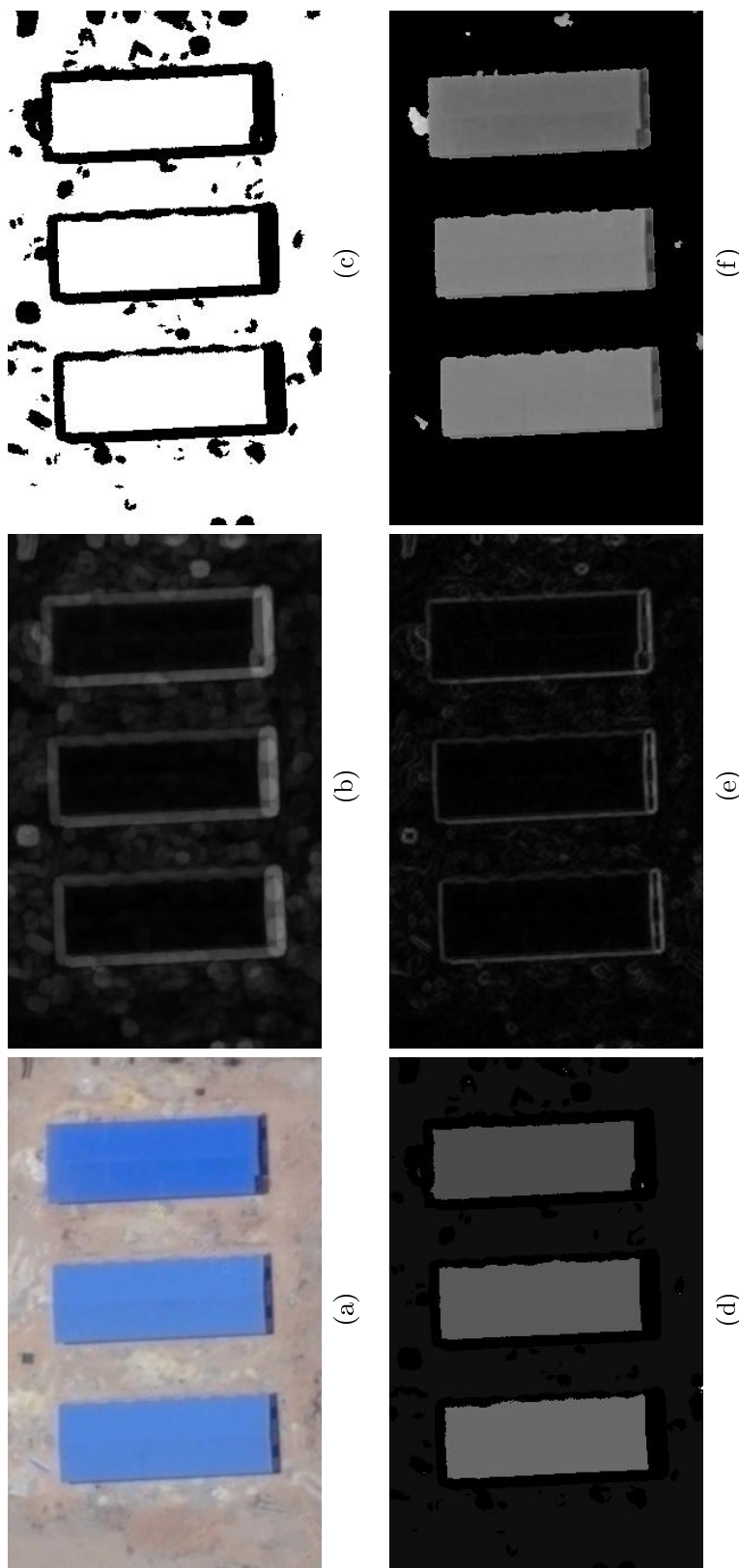


Figure 3-31: a.) Original image. b.) Gradient image. c.) Thresholded gradient image. d.) The markers or “water sources”. e.) The gradient image to be used with the markers to apply the watershed algorithm. f.) The result of the watershed mask applied to the original image.

3.4.3 Active Contours

Active contours or snakes have been widely researched and have many applications such as image segmentation, edge detection, line detection, subjective contour detection and motion tracking [33, 41, 37, 5, 39]. There are, fundamentally, two different kinds of active contours: parametric active contours [35] and the geometric active contours [65]. The difference and comparison of the two methods can be found in [65]. The idea behind parametric active contours, developed by Kass et al. in the late 1980s [33], is to start with an initial set of points, called a snake $\mathbf{v}(s) = \{(x(s), y(s)), s \in D_f\}$ where D_f is the image definition domain, and then change the coordinates of the points in order to minimise an energy function using an iterative algorithm. The energy functions can be used to control the snake's movement toward features of interest while penalising certain form features that the snake may take on. The result will be a contour as close to the object of interest as the energy functions will allow.

The first energy function we use is based on the gradient function

$$\mathbf{v}_s = \frac{d\mathbf{v}(s)}{ds}$$

and is given by

$$E_{\text{elastic}} = \frac{1}{2} \int_{s \in D_f} (\alpha(s) |\mathbf{v}_s|^2) ds.$$

This energy function causes tension in the snake and discourages stretching of the snake. The weight $\alpha(s)$ can be adjusted to change the elasticity allowed in the snake.

The next energy function is based on the second derivative function

$$\mathbf{v}_{ss} = \frac{d^2\mathbf{v}(s)}{ds^2}$$

and is given by

$$E_{\text{bending}} = \frac{1}{2} \int_{s \in D_f} (\beta(s) |\mathbf{v}_{ss}|^2) ds.$$

This energy function is used to control the amount of bending in the snake. One can almost think of the tension that is present in a bent ruler that wants to restore the ruler to its original shape. The weight $\beta(s)$ can be used to control the amount of bending allowed in the snake. If $\beta(s) = 0$, for some s , then the snake can be second-order discontinuous at s which allows for the snake to create a corner at that point.

The two energy functions mentioned control some of the shape properties of the snake itself and are called *internal* energy functions. In order to let the snake be attracted to certain image features, we use *image* energy functions. These energy functions can be based on any feature that is deemed important for the snake, such as edges, lines, contours, intensity levels etc. An example of such an energy function is gradient function of an image based on the gradient function

$$\nabla I(x, y) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$$

and the energy function used is

$$E_{\text{edge}} = -|\nabla I(x, y)|^2.$$

We now have enough energy functions to construct a snake which will be attracted toward edges in images while controlling the elasticity and bending of the snake. The complete energy function is given by:

$$E_{\text{snake}} = \frac{1}{2} \int_{s \in D_f} [\alpha(s)|\mathbf{v}_s|^2 + \beta(s)|\mathbf{v}_{ss}|^2 - |\nabla I(x, y)|^2] ds$$

and needs to be minimised which can be done by using variational calculus and applying Euler-Lagrange differential equation to obtain the force balance equation [33]. When the energy function of the snake has been iteratively minimised, the snake should surround the relevant features in the image in question.

Figure 3.32 shows the result of snakes applied to four polygons and the resulting contours after 120 iterations using an adaptive active contour method [3].

3.5 Conclusion

In this chapter we introduced two types of image noise, namely salt & pepper noise and speckle noise, and applied these different noise to two different images. As test subjects, we used a toy polygon image and a satellite image of buildings. We discussed five different image smoothing techniques, of which one was a new proposed image smoother, and compared them in terms of how well they remove two different types of noise at different intensity levels from image. In order to extract objects from the smoothed images, we used three different contour detection algorithms which were discussed in detail. In the next chapter we will see how the contour detection algorithms compared when applied to the smoothed images using different smoothers when different noise are present at different intensity levels.

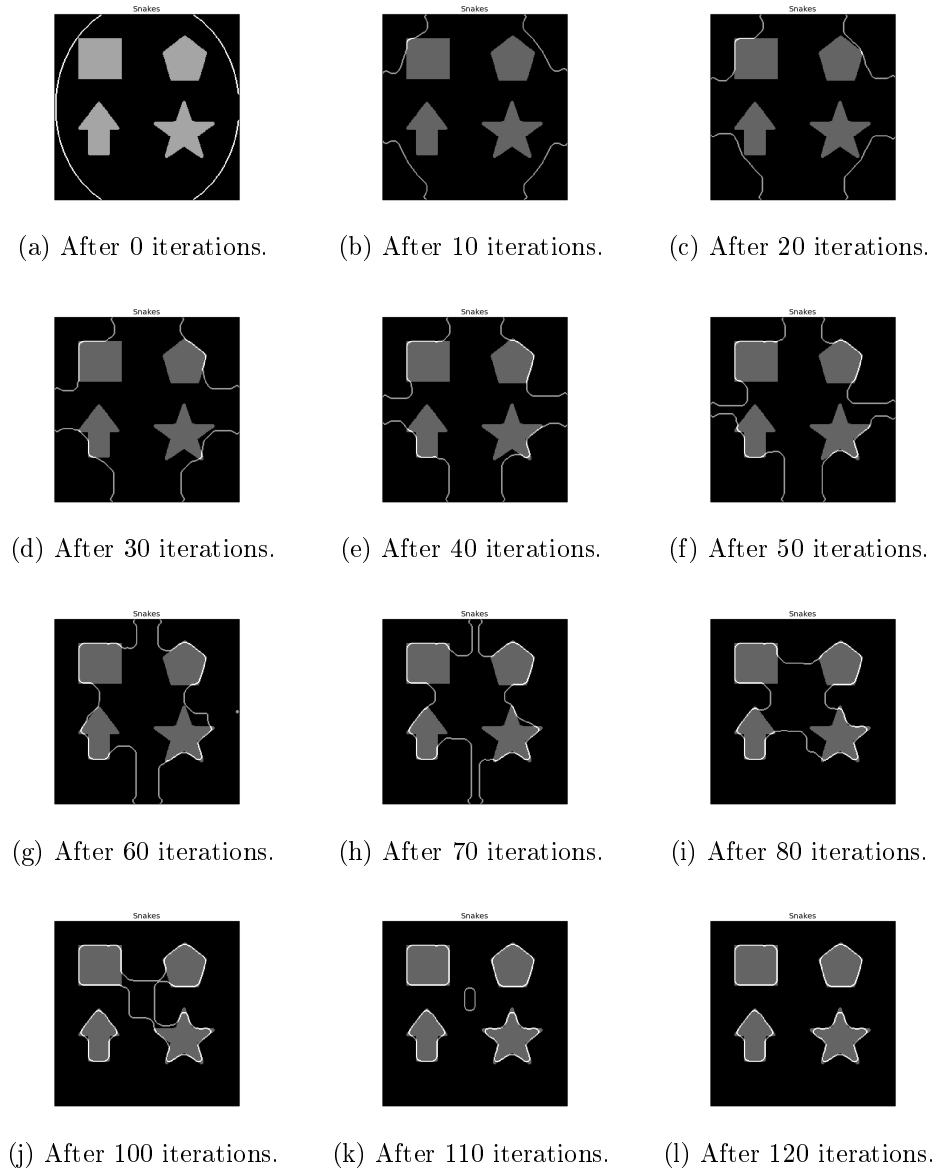


Figure 3.32: An example of active contours applied to arbitrary polygons.

Chapter 4

Applications

4.1 Introduction

After introducing some basic mathematical morphology principles in Chapter 2 and discussing various image smoothing techniques and contour detection algorithms in Chapter 3, we use Chapter 4 to combine these chapters into useable algorithms that are viable in practice and give some advice on which combinations to use under certain circumstances. We first discuss how we compare the results of the different algorithms and then show the output of the physical results. A thorough discussion is then given comparing the results of the different algorithms.

4.2 Comparison

We now combine the different smoothing algorithms with the contour detection algorithms in order to detect objects in images. We apply these algorithms to two images, one of which was generated with random polygons and the other is a real life satellite image with buildings¹. All coding was done in Python². We coded the robust statistic estimation algorithm and the connected median smoother ourselves as well as all preprocessing of the images before the contour detection algorithms were applied. All the code can be found in the Appendix of this dissertation and on GitHub³. In order to simulate noise that may be found in practice, we added various levels of salt & pepper noise as well as various levels of speckle noise to the images. The smoothing algorithms discussed in Chapter 3 are then applied as well as the contour detection algorithms and we measure the accuracy of the polygons detected for the combinations of algorithms. We exclude the L_1U_1 smoother we used in Chapter 3 as we saw that the $LULU_{conv}$ outperformed it in all cases. We used *True Positive (TP)*, *True Negative (TN)*, *False Positive (FP)* and *False Negative (FN)* classification ($FPFN^*$) to measure the accuracy of the different algorithms [2]. This measure is given by

$$FPFN^* = \frac{TP + TN}{TP + TN + FP + FN}$$

¹Image of buildings obtained using Google Maps (<https://www.google.co.za/maps/place/Cairo,+Cairo+Governorate,+Egypt/@30.1458956,31.4528905,574m/data=!3m1!1e3!4m2!3m1!1s0x14583fa60b21beeb:0x79dfb296e8423bba>)

²Python Software Foundation. Python Language Reference, version 2.7. Available at <http://www.python.org>

³<https://github.com/Magnusvann/WST895>

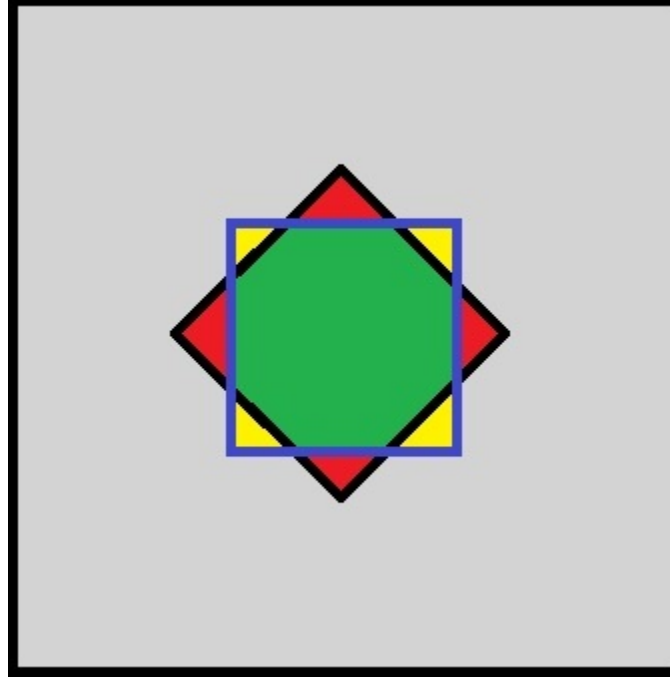


Figure 4.1: An illustration of the sets used in the false positive, false negative ratio.

Figures 4.8 to 4.49 show some results that illustrate the different steps executed in order to extract the objects from the images. Starting from the original image a.), noise was added and is shown in b.), the image was smoothed as seen in c.) in order to try to remove the noise and then a contour detection algorithm was used to identify the objects in the images shown in d.). The extracted objects were then compared with a ground truth image given in e.) and the image difference is shown f.). In Tables 4.1 to 4.3 we provide the final results for the extraction of the objects in the images using different combinations of smoothers and contour detections techniques. The *FPFN* measure in the tables is of the form

$$FPFN = \frac{TP}{TP + FP + FN}$$

since we expect TN to be very accurate and we value TP a lot more than TN. In Figure 4.1 we show a figure containing a black square and a blue square. The black square illustrates a polygon that need to be identified and extracted. The blue square is what an algorithm detected. The green area represents a TP area, the yellow area is the FP area. The red area was not correctly classified and is the FN area. The grey area is the TN area. Our images are of the nature that we expect a large percentage of TN and thus we want to confine accuracy measure to the three sets TP, FP and FN to try to exclude the possible bias that including TN may bring to our results. From the tables it is clear that prior knowledge of the kind of image you're dealing with together with the kind of noise you expect to be present in an image can help considerably in improving the accuracy at which an automated system will be able to extract objects from images. We see that most of the contour detection algorithms seize to give acceptable results at noise levels of 50% except for some combinations where the adaptive median filter and the robust statistic estimation filter were used. Most of the combinations performed relatively well with a noise level of 20% and lower.

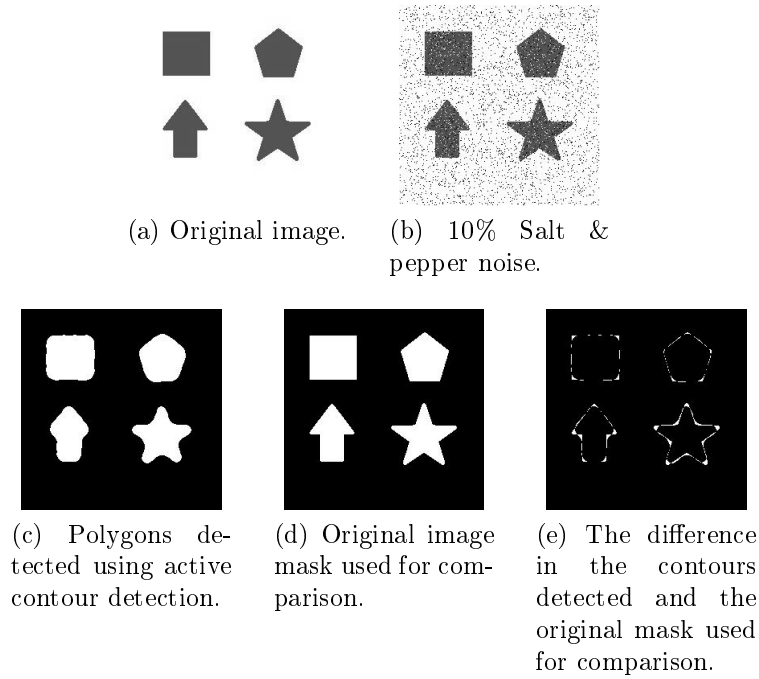


Figure 4.2: 10% Salt & pepper noise, no smoother and active contour detection algorithm. $FPFN = 92.45\%$

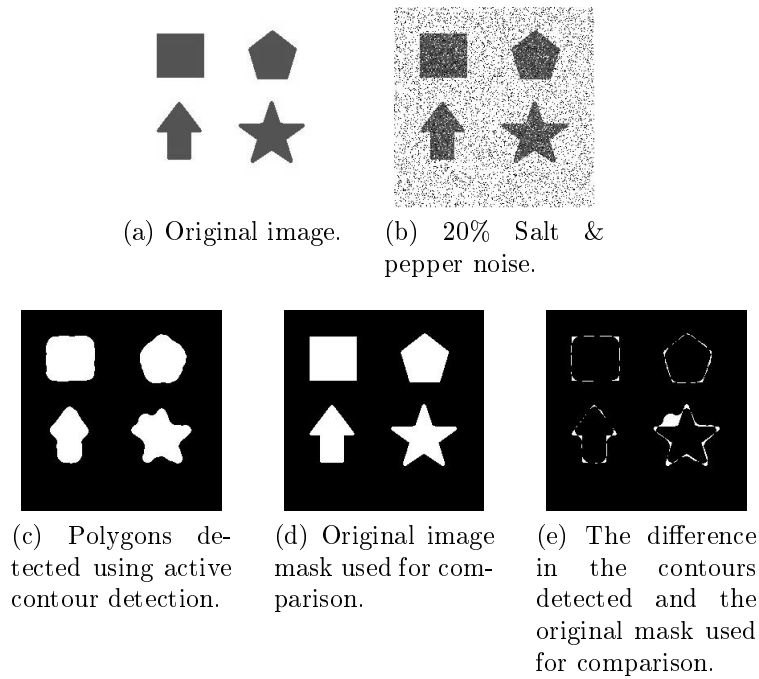


Figure 4.3: 20% Salt & pepper noise, no smoother and active contour detection algorithm. $FPFN = 89.53\%$

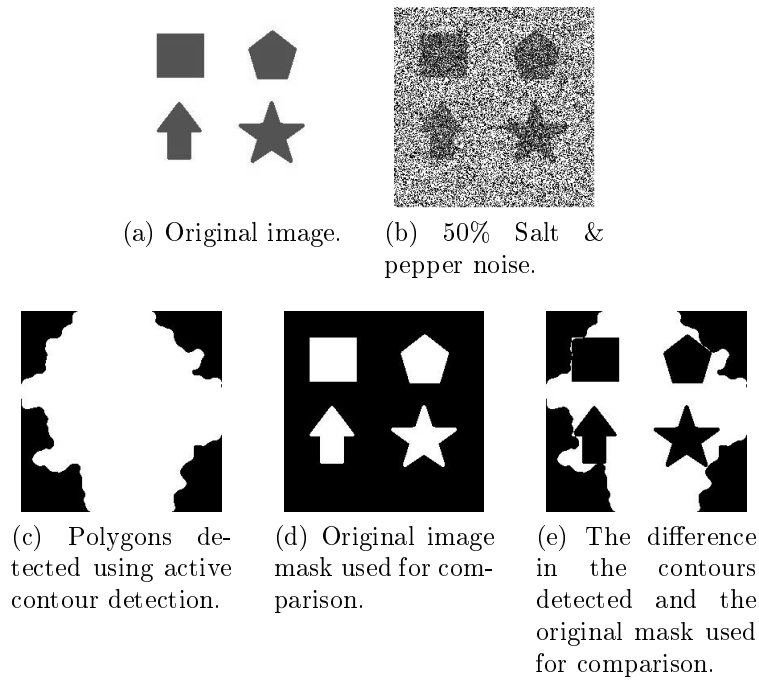


Figure 4.4: 50% Salt & pepper noise, no smoother and active contour detection algorithm. $FPFN = 24.01\%$

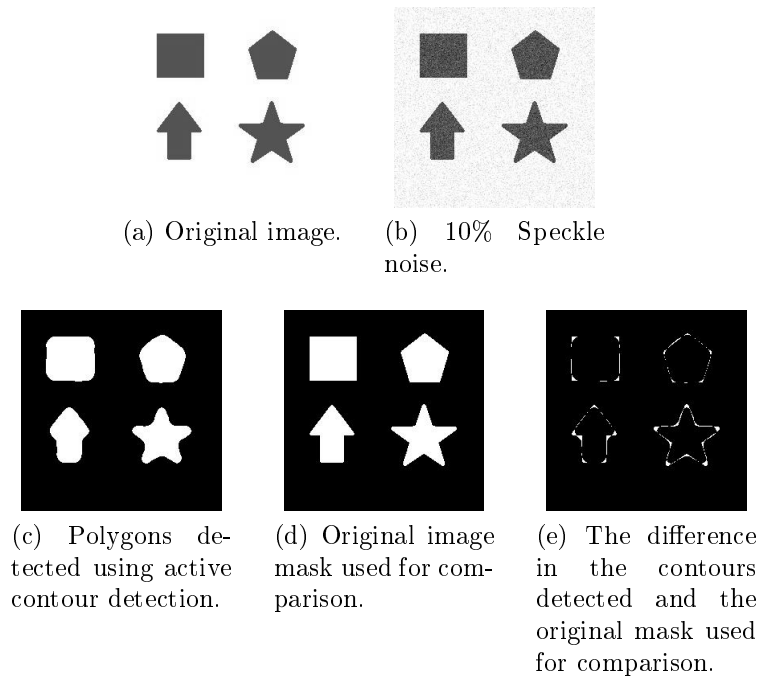


Figure 4.5: 10% Speckle noise, no smoother and active contour detection algorithm. $FPFN = 92.94\%$

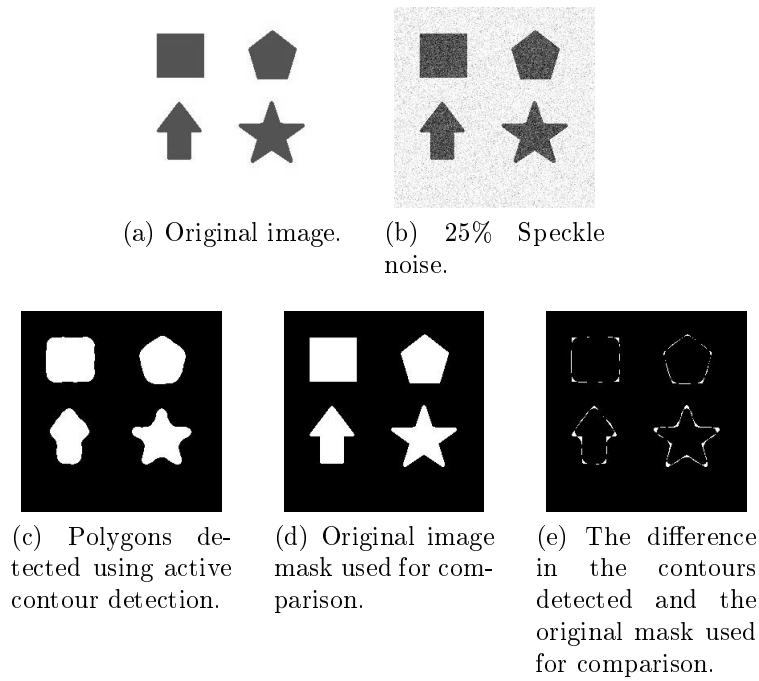


Figure 4.6: 25% Speckle noise, no smoother and active contour detection algorithm. $FPFN = 92.47\%$

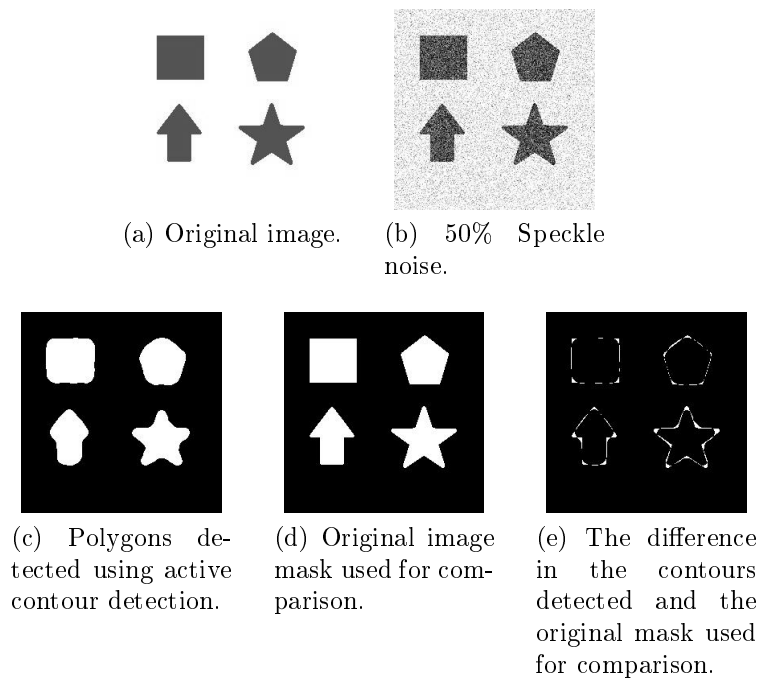


Figure 4.7: 50% Speckle noise, no smoother and active contour detection algorithm. $FPFN = 91.67\%$

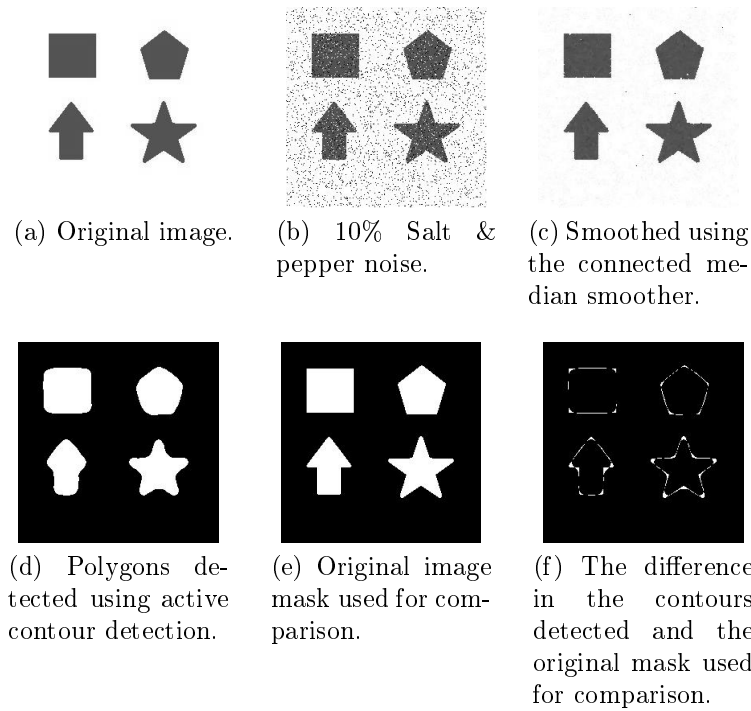


Figure 4.8: 10% Salt & pepper noise, connected median smoother and active contour detection algorithm. $FPFN = 92.48\%$

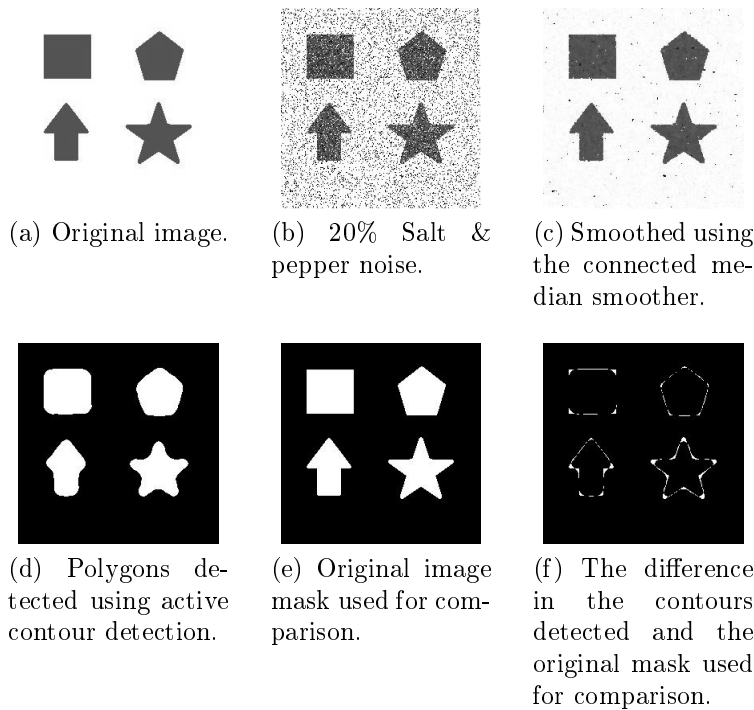


Figure 4.9: 20% Salt & pepper noise, connected median smoother and active contour detection algorithm. $FPFN = 92.63\%$

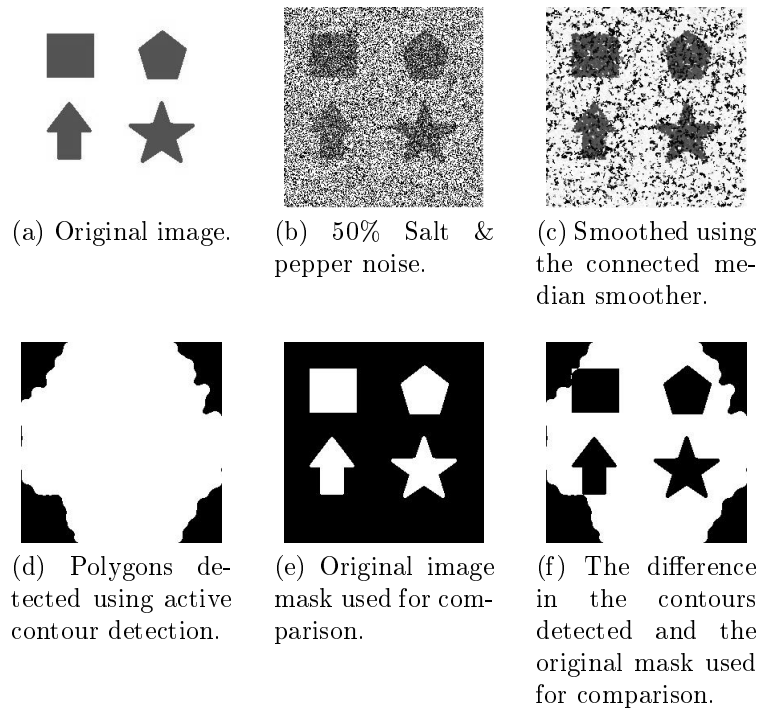


Figure 4.10: 50% Salt & pepper noise, connected median smoother and active contour detection algorithm. $FPFN = 20.86\%$

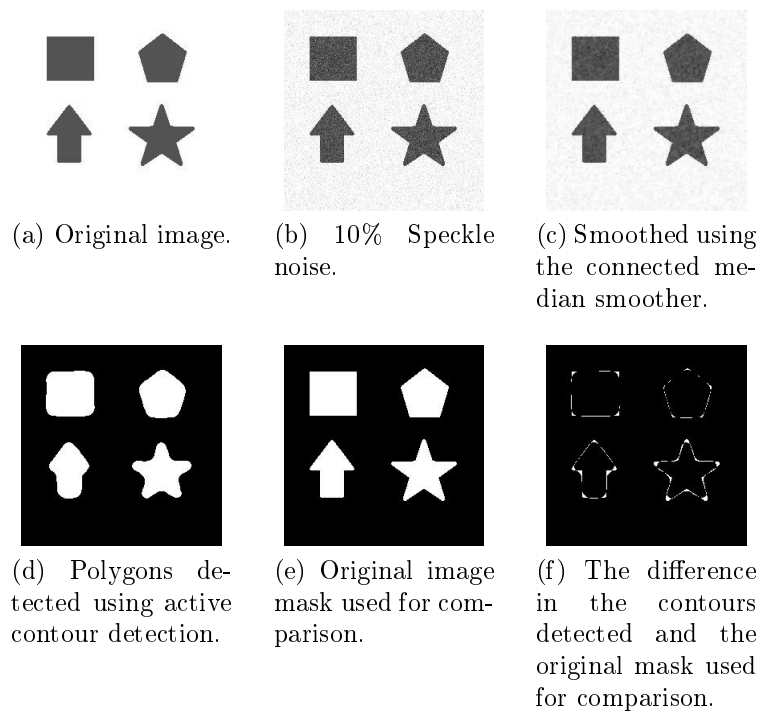


Figure 4.11: 10% Speckle noise, connected median smoother and active contour detection algorithm. $FPFN = 92.90\%$

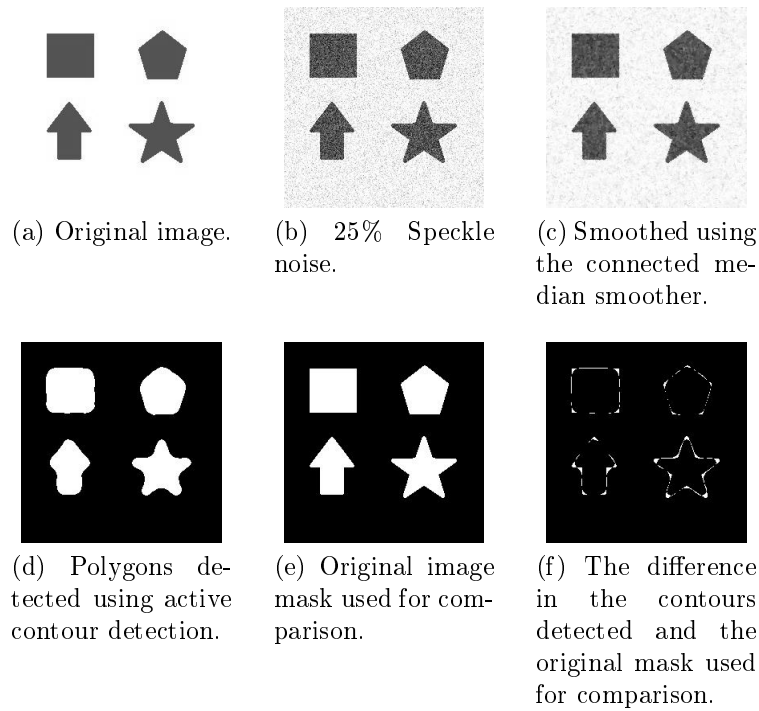


Figure 4.12: 25% Speckle noise, connected median smoother and active contour detection algorithm. $FPFN = 92.66\%$

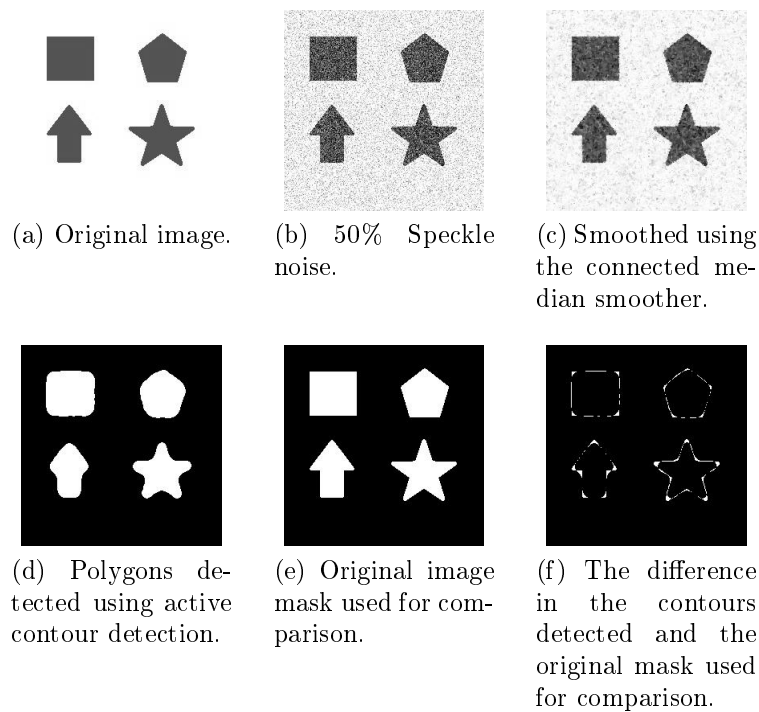


Figure 4.13: 50% Speckle noise, connected median smoother and active contour detection algorithm. $FPFN = 92.41\%$

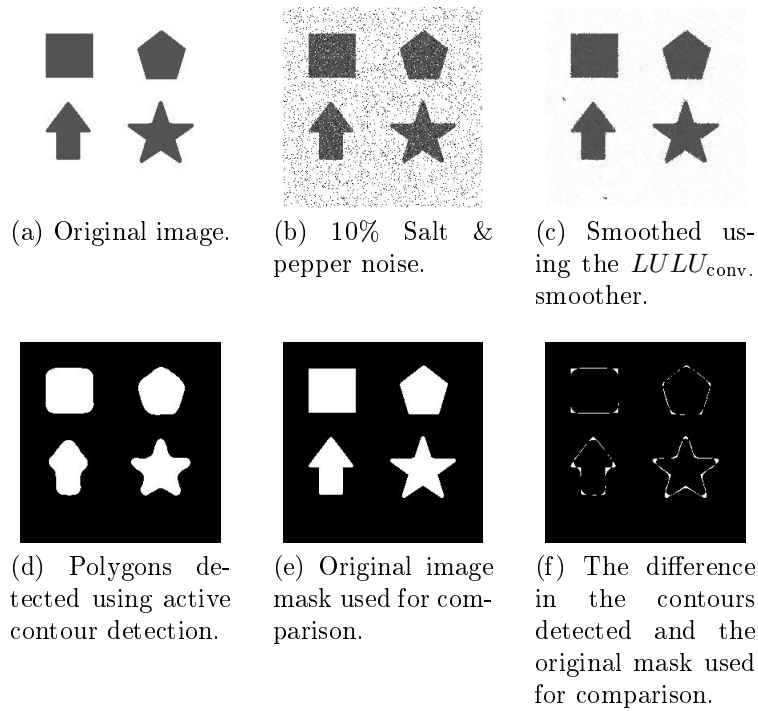


Figure 4.14: 10% Salt & pepper noise, $LULU_{conv.}$ smoother and active contour detection algorithm. $FPFN = 92.59\%$

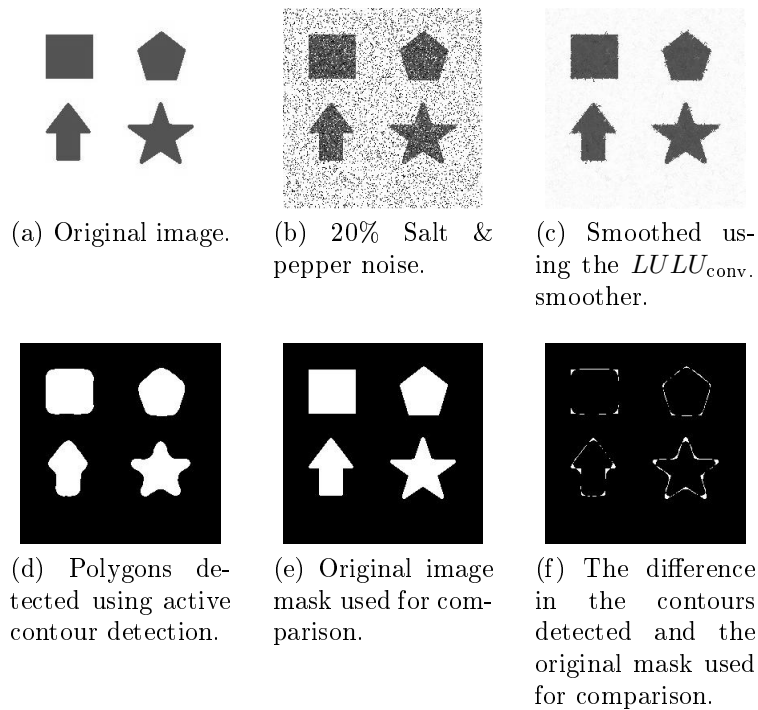


Figure 4.15: 20% Salt & pepper noise, $LULU_{conv.}$ smoother and active contour detection algorithm. $FPFN = 92.70\%$

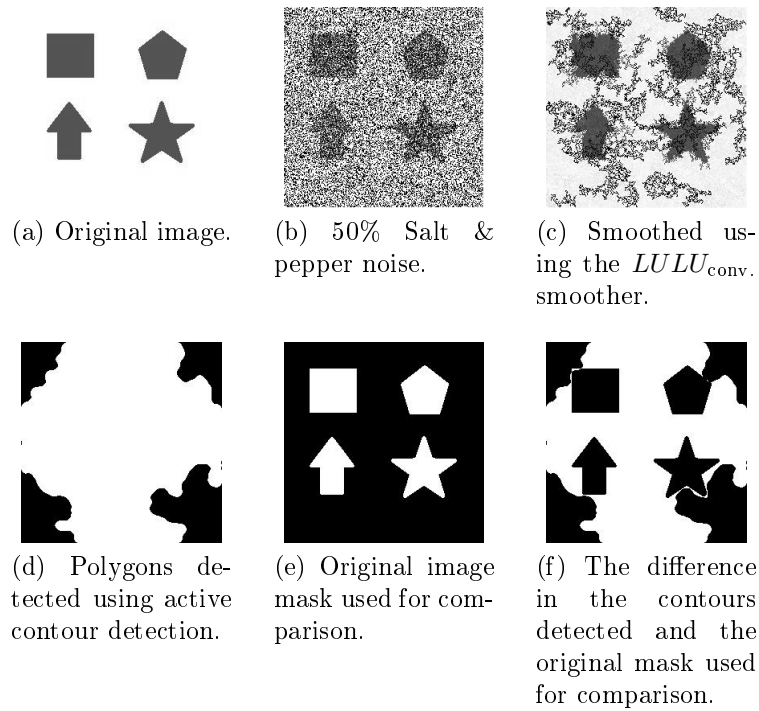


Figure 4.16: 50% Salt & pepper noise, $LULU_{conv.}$ smoother and active contour detection algorithm. $FPFN = 22.21\%$

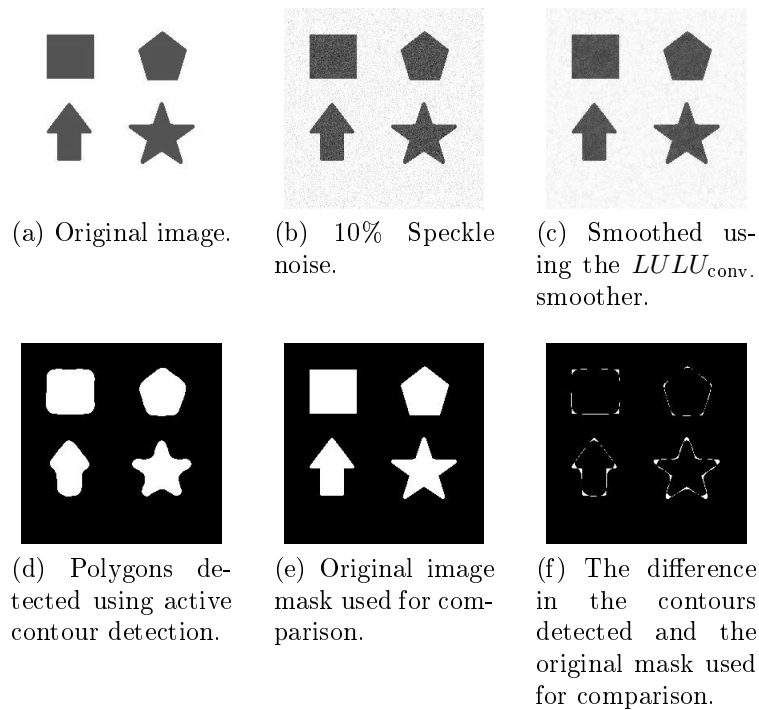


Figure 4.17: 10% Speckle noise, $LULU_{conv.}$ smoother and active contour detection algorithm. $FPFN = 93.06\%$

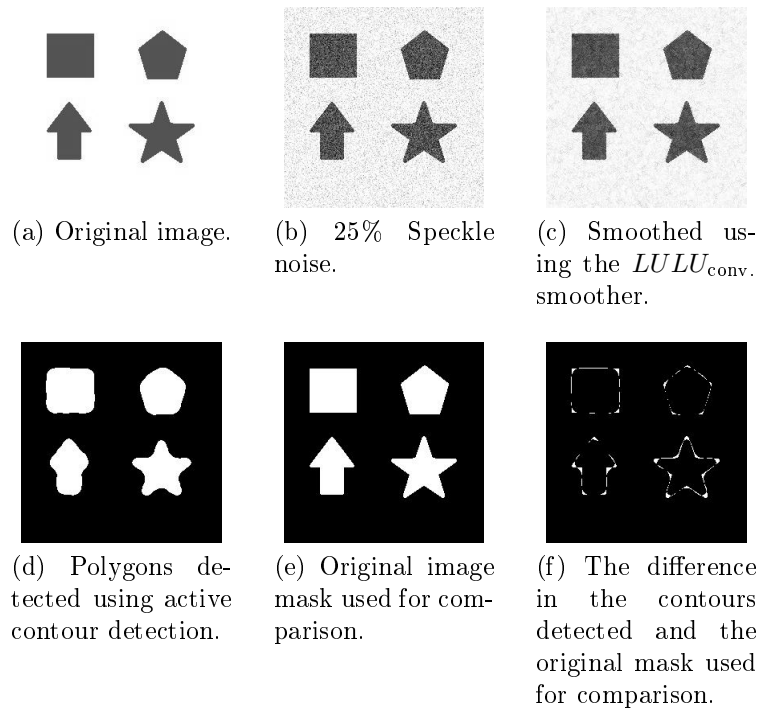


Figure 4.18: 25% Speckle noise, $LULU_{conv}$ smoother and active contour detection algorithm. $FPFN = 92.71\%$

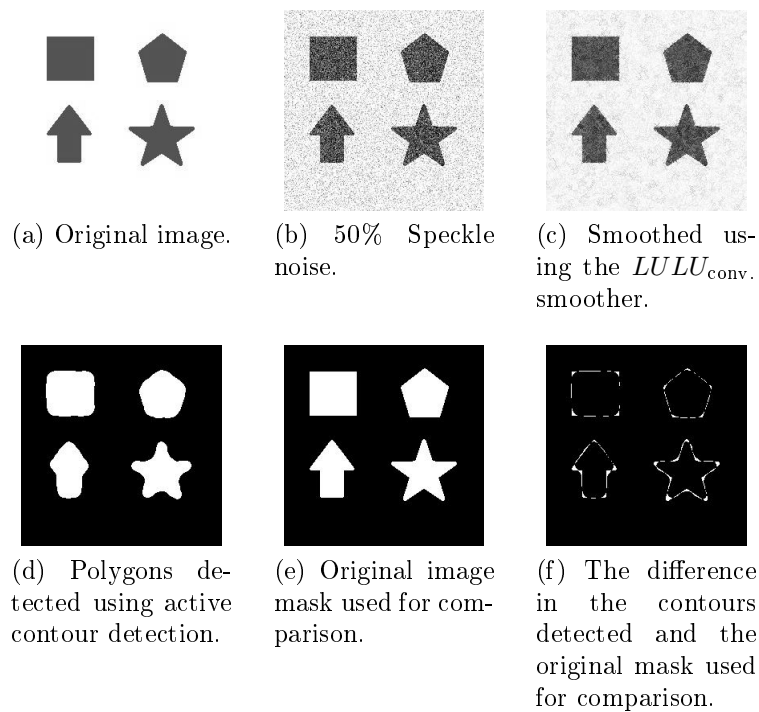


Figure 4.19: 50% Speckle noise, $LULU_{conv}$ smoother and active contour detection algorithm. $FPFN = 92.05\%$

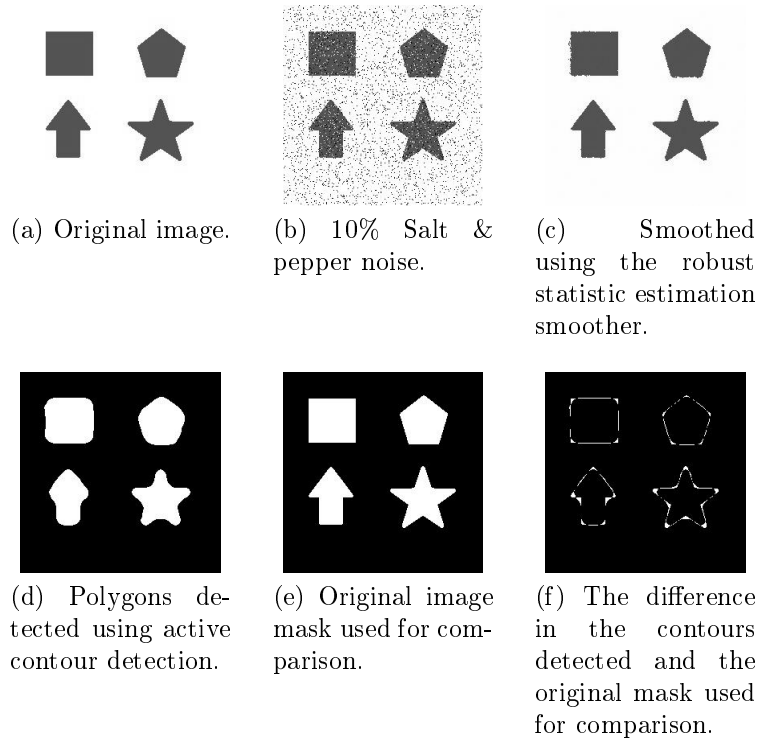


Figure 4.20: 10% Salt & pepper noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 91.68\%$

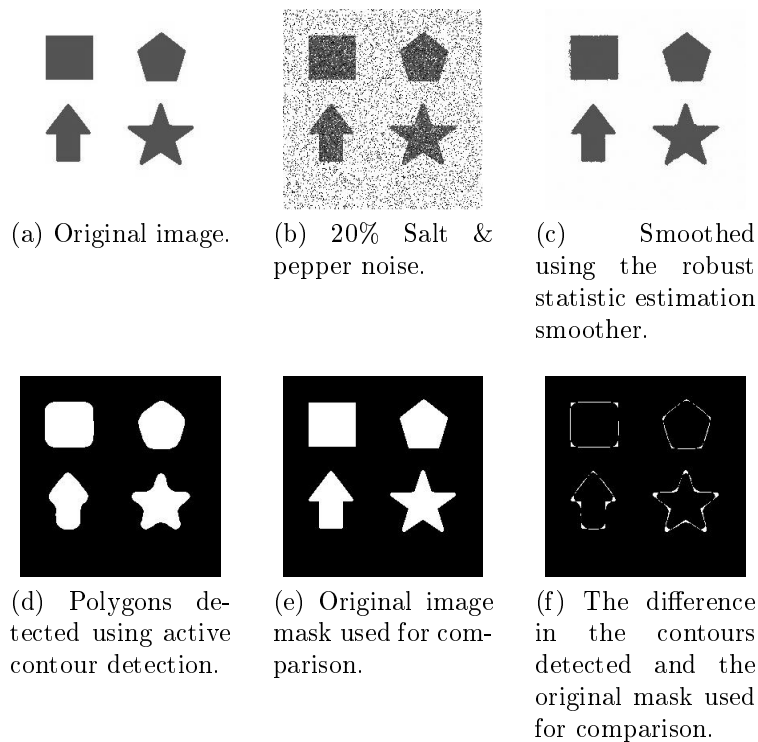


Figure 4.21: 20% Salt & pepper noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 91.61\%$

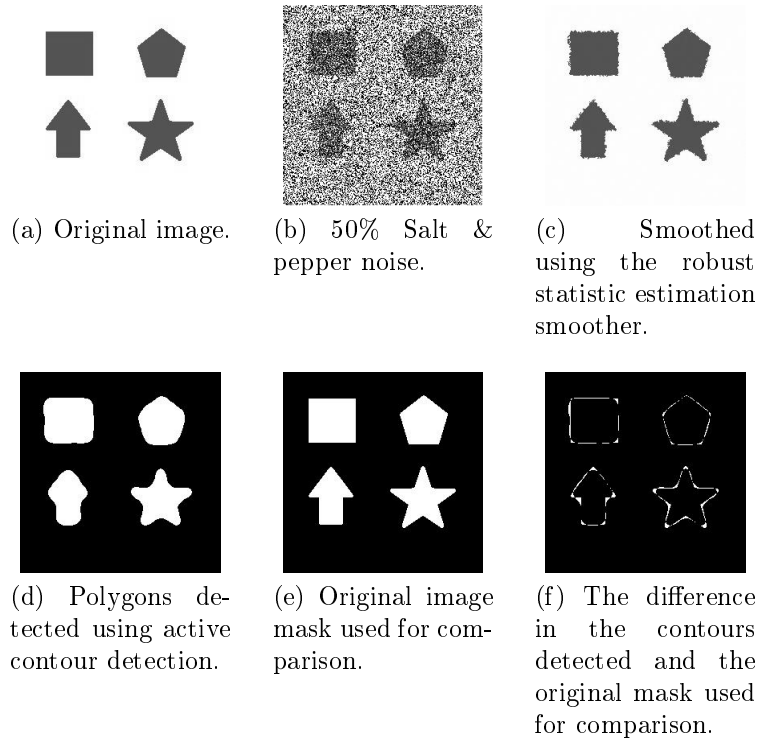


Figure 4.22: 50% Salt & pepper noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 91.27\%$

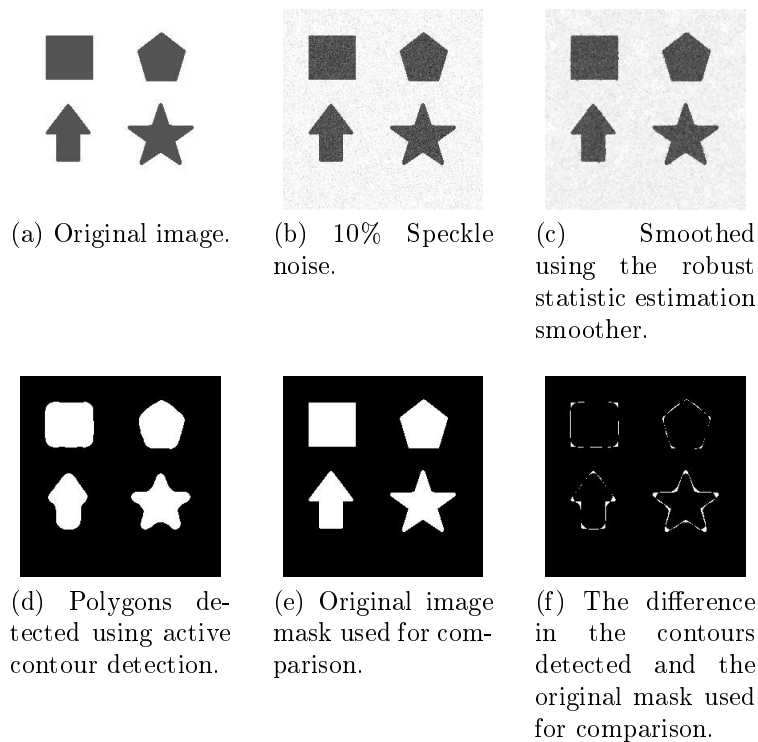


Figure 4.23: 10% Speckle noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 92.37\%$

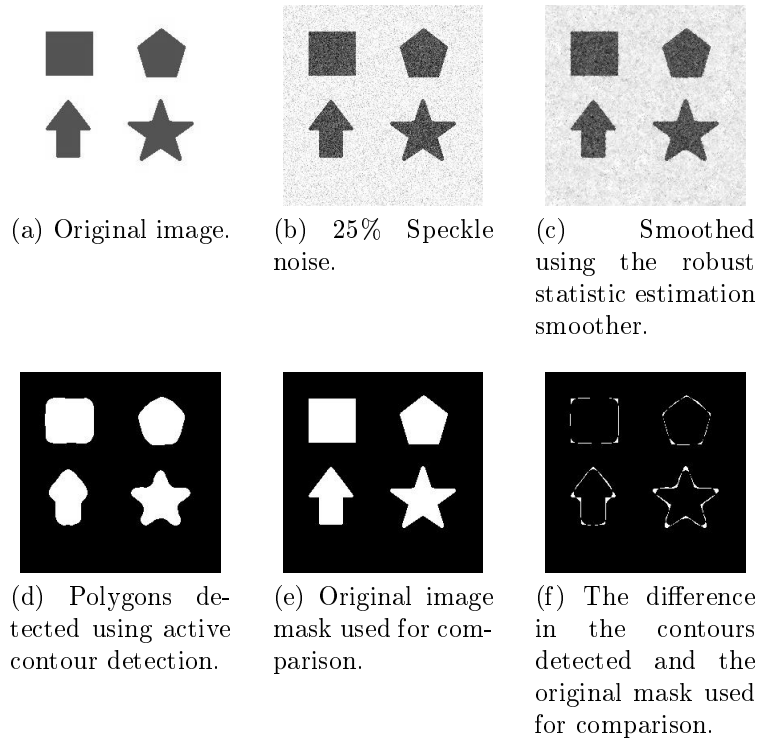


Figure 4.24: 25% Speckle noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 91.36\%$

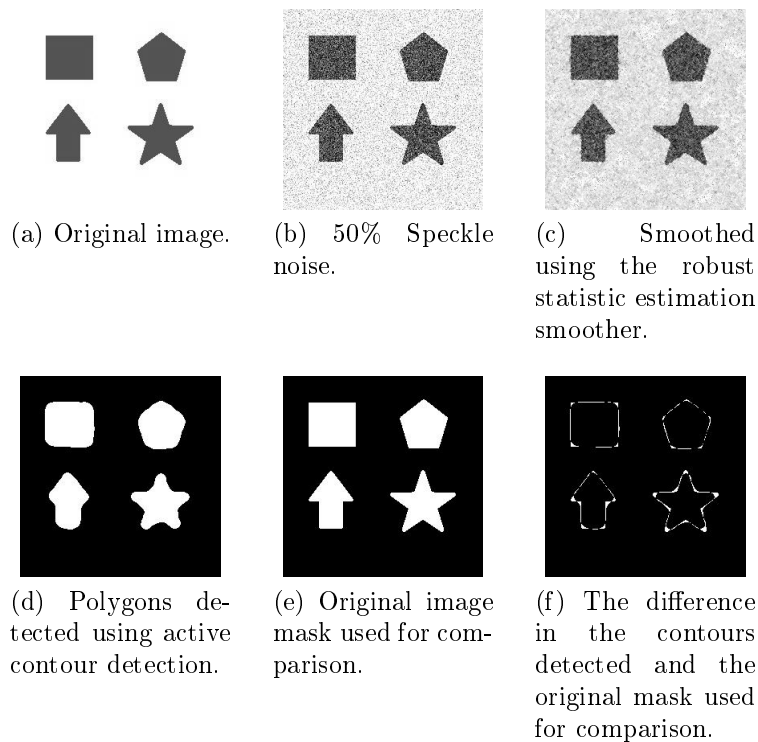


Figure 4.25: 50% Speckle noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 91.44\%$

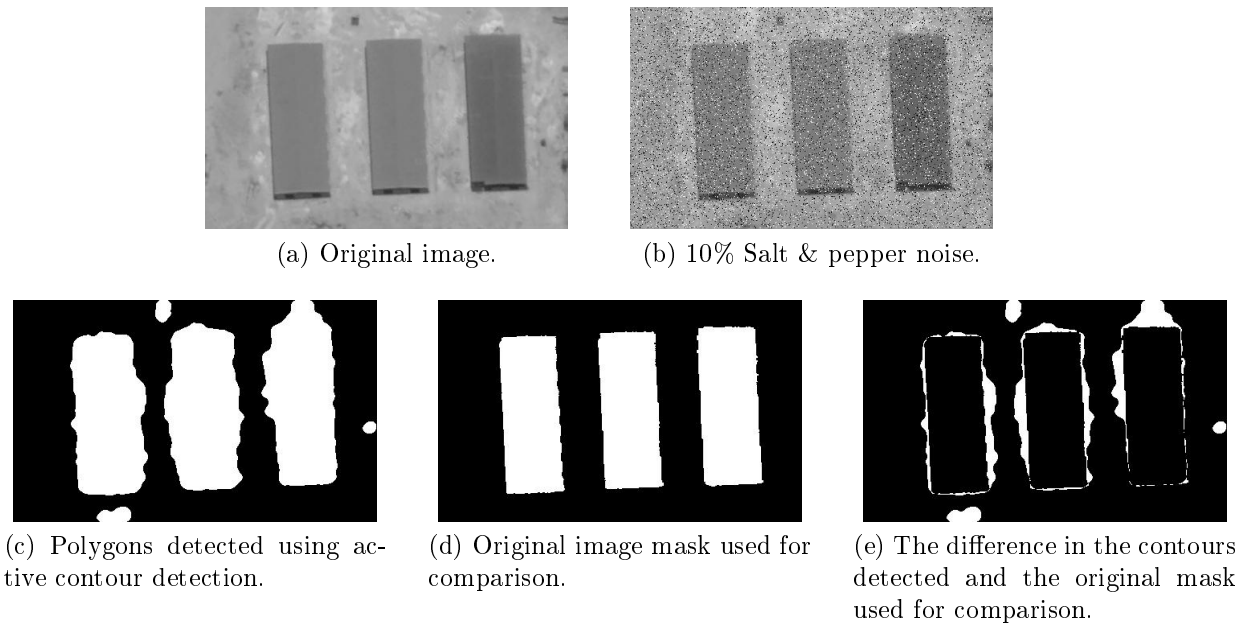


Figure 4.26: 10% Salt & pepper noise, no smoother and active contour detection algorithm. $FPFN = 81.04\%$

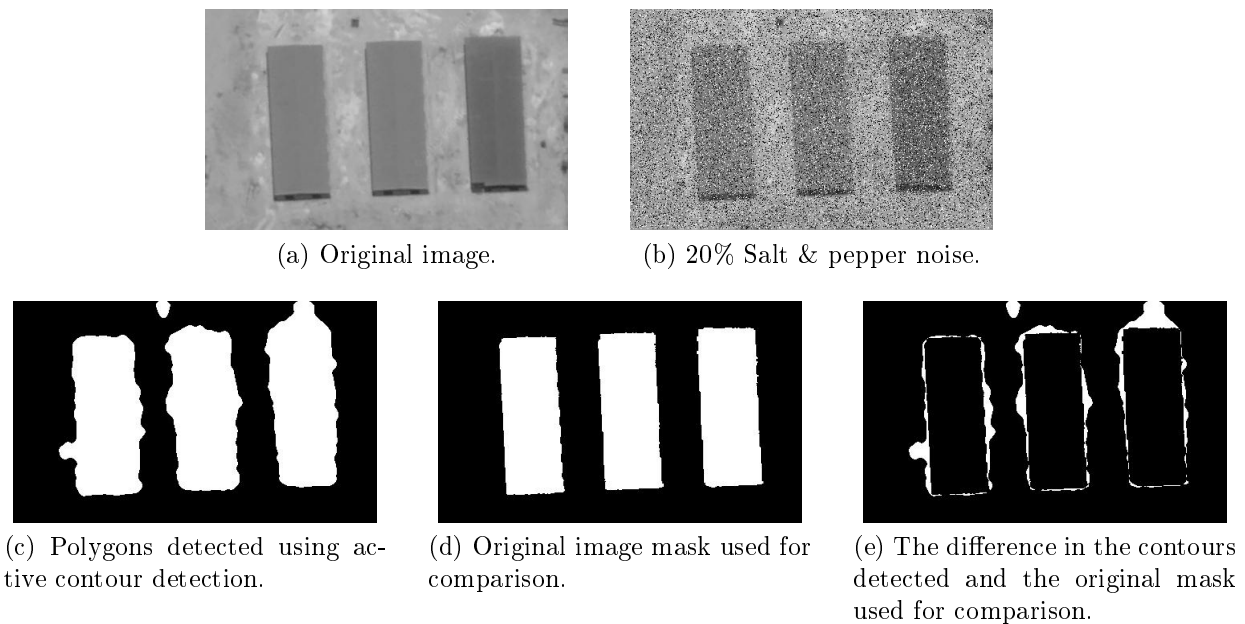


Figure 4.27: 20% Salt & pepper noise, no smoother and active contour detection algorithm. $FPFN = 85.31\%$

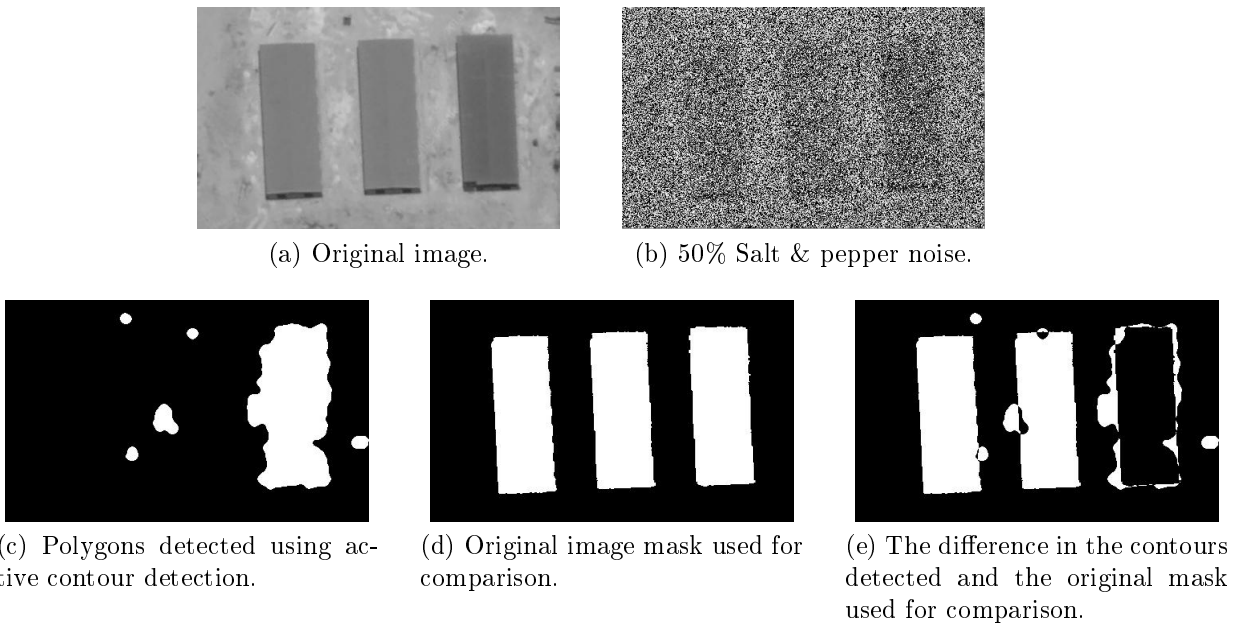


Figure 4.28: 50% Salt & pepper noise, no smoother and active contour detection algorithm. $FPFN = 30.54\%$

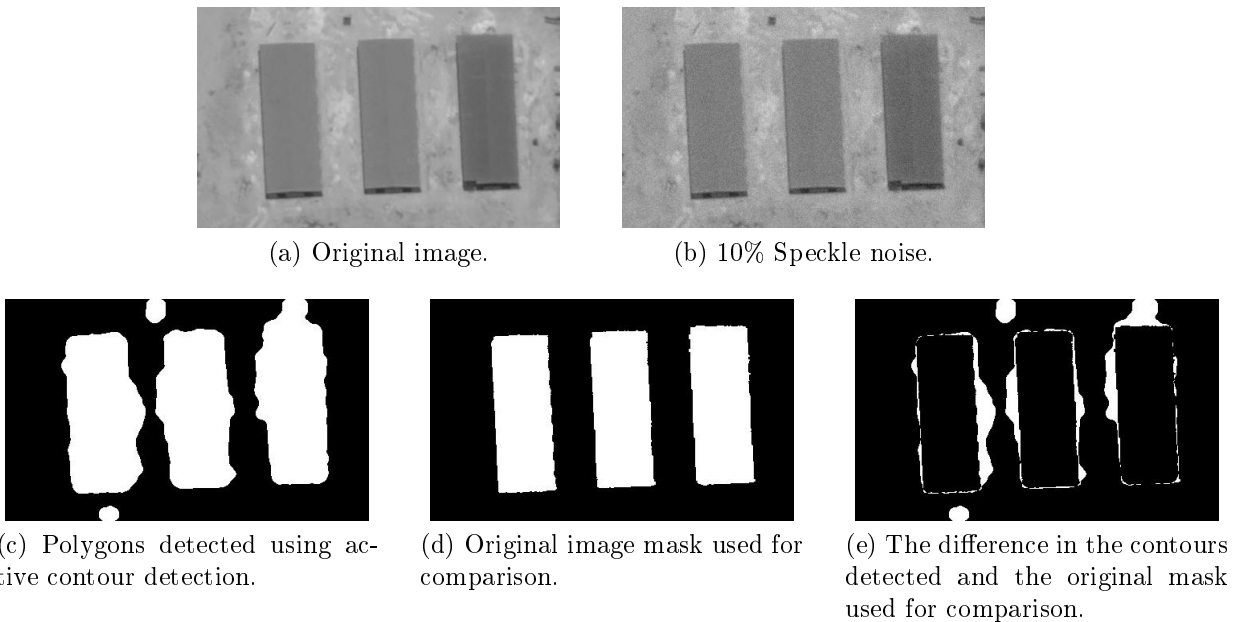


Figure 4.29: 10% Speckle noise, no smoother and active contour detection algorithm. $FPFN = 81.77\%$

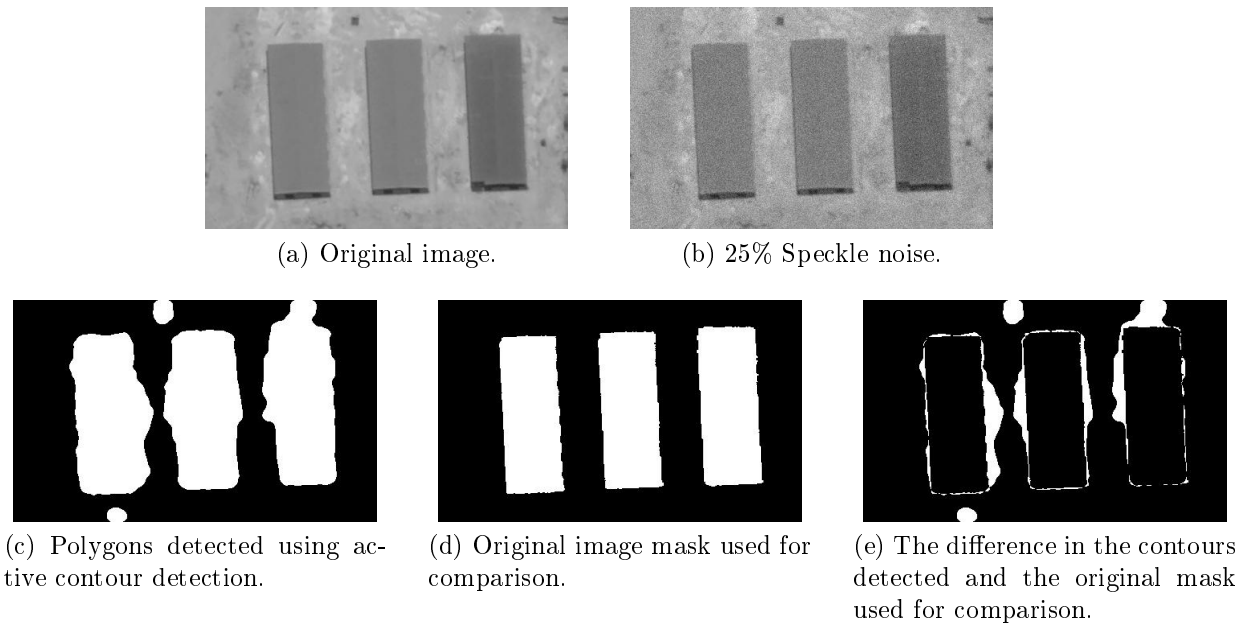


Figure 4.30: 25% Speckle noise, no smoother and active contour detection algorithm. $FPFN = 81.57\%$

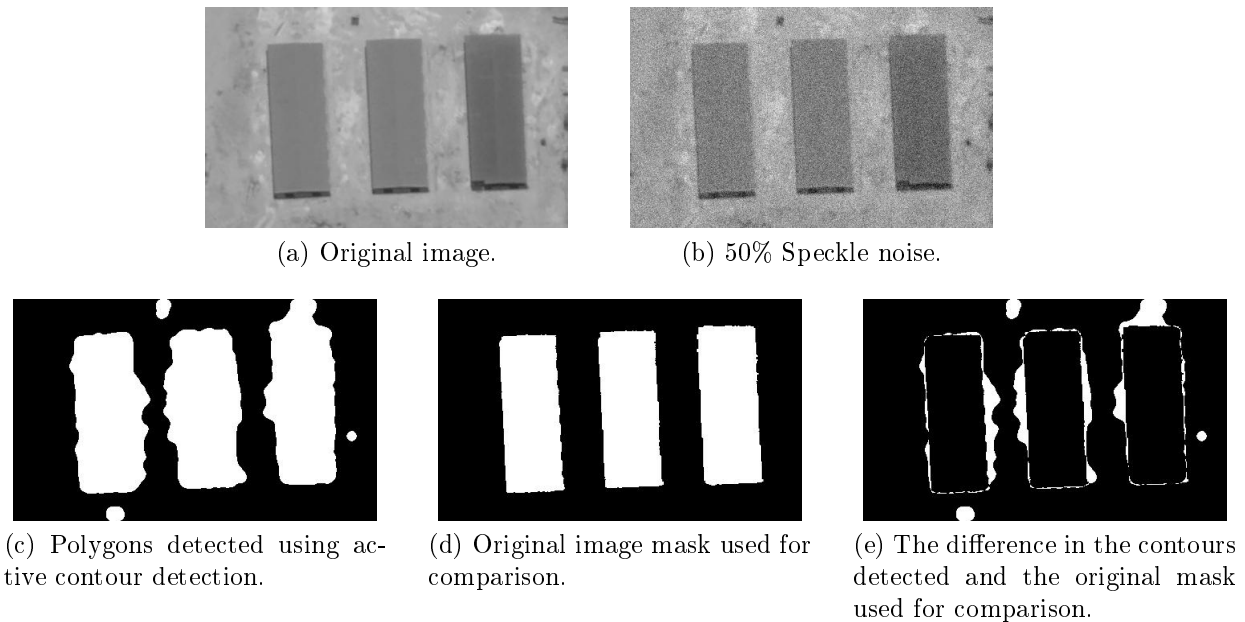


Figure 4.31: 50% Speckle noise, no smoother and active contour detection algorithm. $FPFN = 82.85\%$

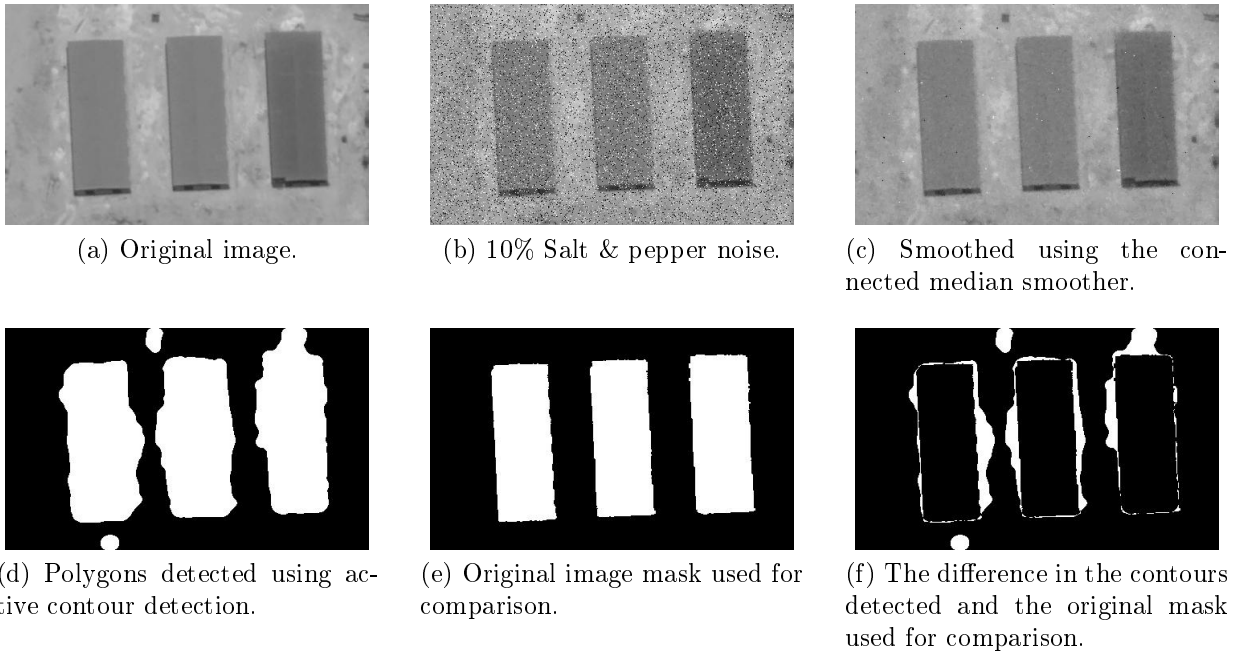


Figure 4.32: 10% Salt & pepper noise, connected median smoother and active contour detection algorithm. $FPFN = 80.92\%$

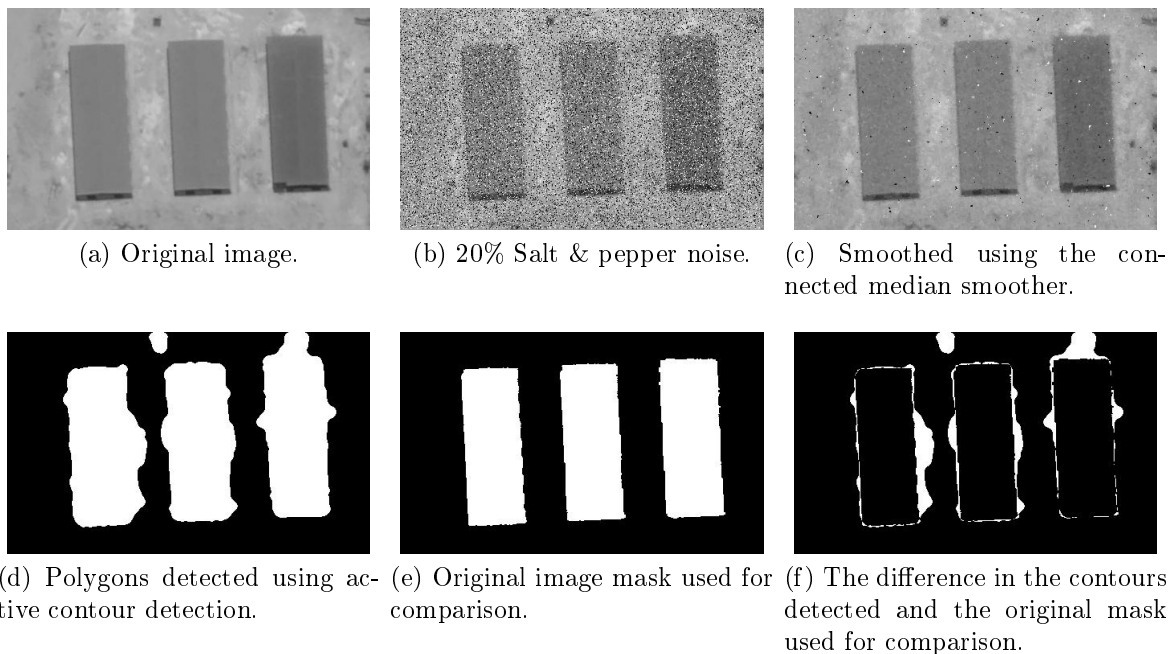
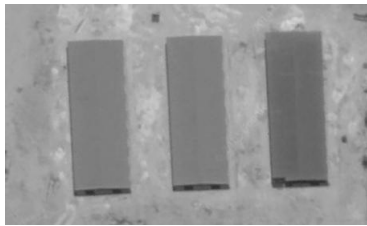
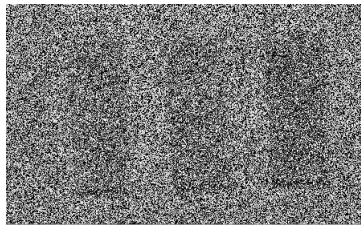


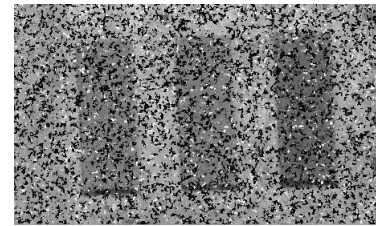
Figure 4.33: 20% Salt & pepper noise, connected median smoother and active contour detection algorithm. $FPFN = 84.70\%$



(a) Original image.



(b) 50% Salt & pepper noise.



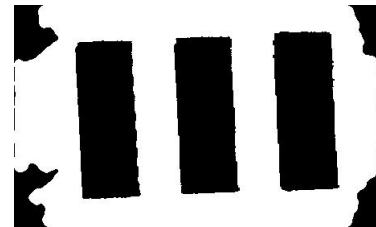
(c) Smoothed using the connected median smoother.



(d) Polygons detected using active contour detection.

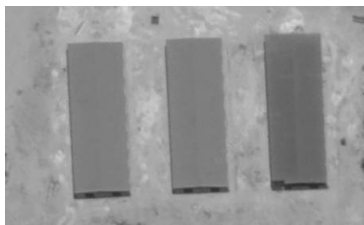


(e) Original image mask used for comparison.

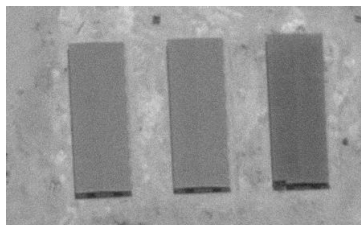


(f) The difference in the contours detected and the original mask used for comparison.

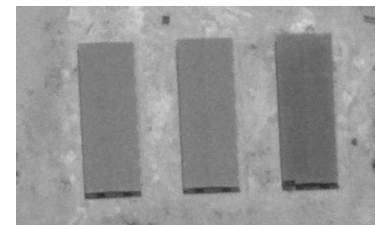
Figure 4.34: 50% Salt & pepper noise, connected median smoother and active contour detection algorithm. $FPFN = 35.61\%$



(a) Original image.



(b) 10% Speckle noise.



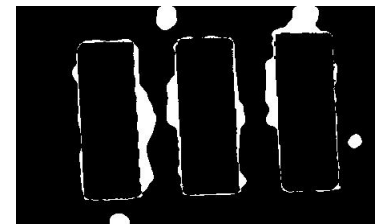
(c) Smoothed using the connected median smoother.



(d) Polygons detected using active contour detection.



(e) Original image mask used for comparison.



(f) The difference in the contours detected and the original mask used for comparison.

Figure 4.35: 10% Speckle noise, connected median smoother and active contour detection algorithm. $FPFN = 81.50\%$

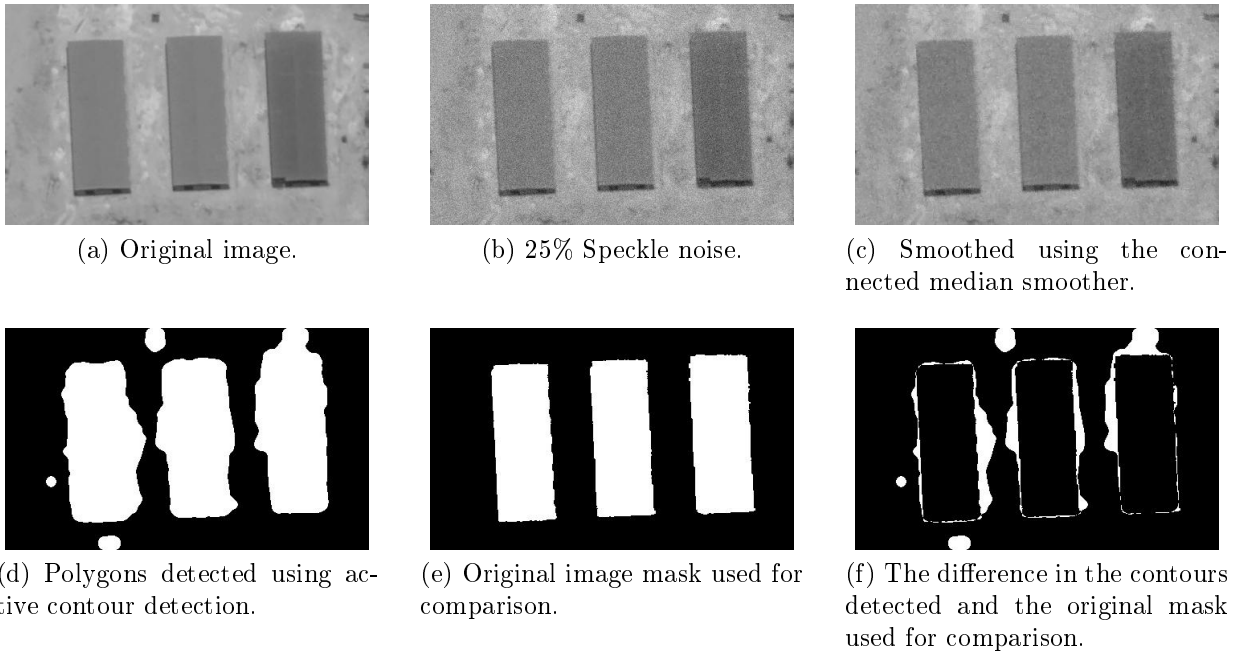


Figure 4.36: 25% Speckle noise, connected median smoother and active contour detection algorithm. $FPFN = 80.69\%$

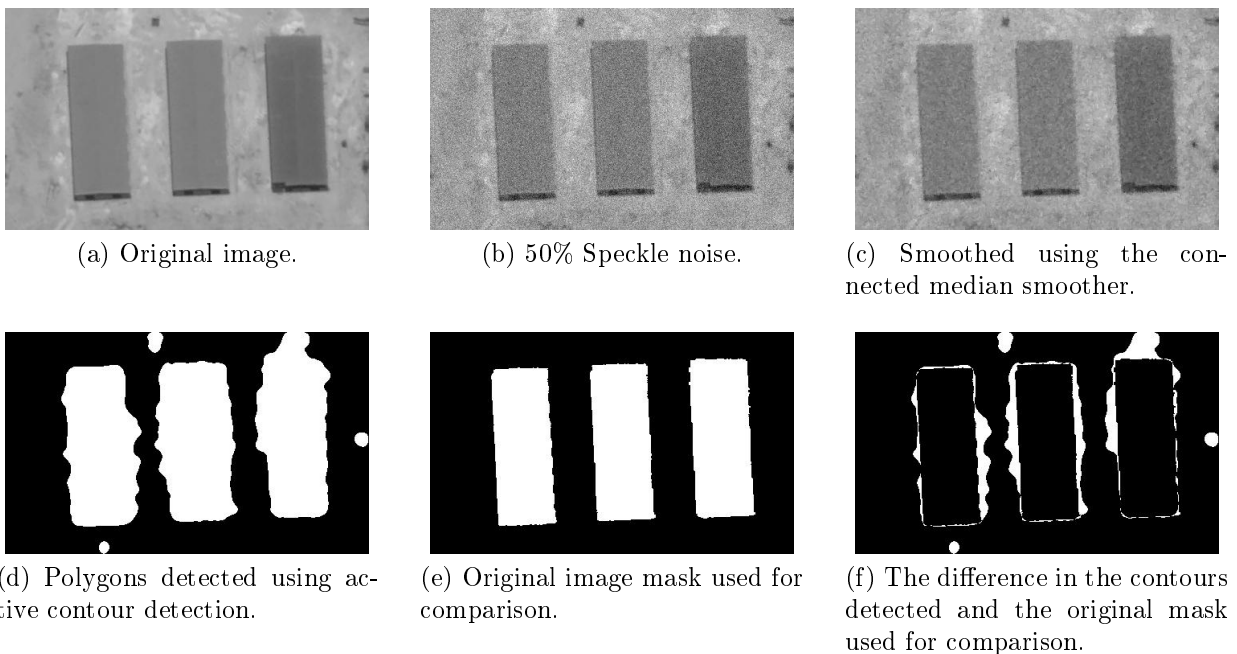
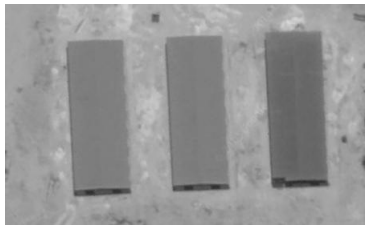
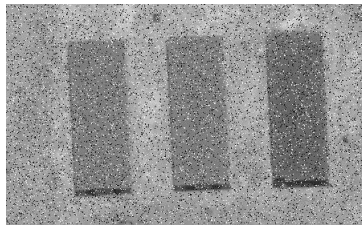


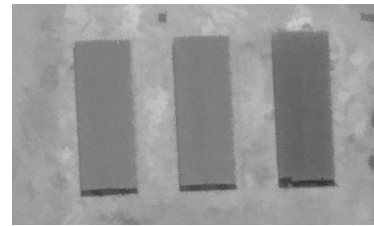
Figure 4.37: 50% Speckle noise, connected median smoother and active contour detection algorithm. $FPFN = 81.18\%$



(a) Original image.



(b) 10% Salt & pepper noise.



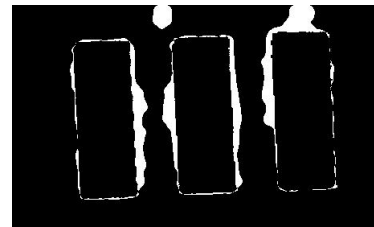
(c) Smoothed using the $LULU_{conv}$ smoother.



(d) Polygons detected using active contour detection.

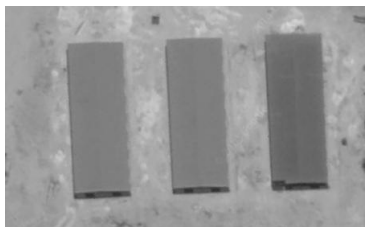


(e) Original image mask used for comparison.

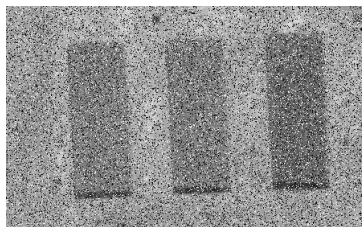


(f) The difference in the contours detected and the original mask used for comparison.

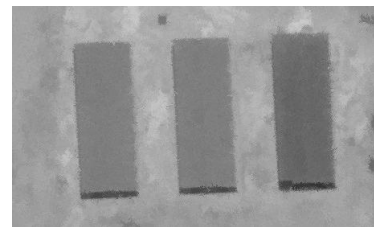
Figure 4.38: 10% Salt & pepper noise, $LULU_{conv}$ smoother and active contour detection algorithm. $FPFN = 82.47\%$



(a) Original image.



(b) 20% Salt & pepper noise.



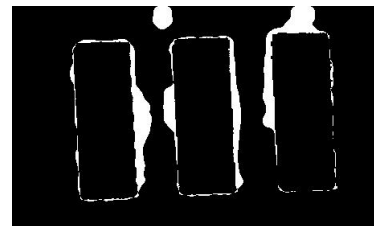
(c) Smoothed using the $LULU_{conv}$ smoother.



(d) Polygons detected using active contour detection.

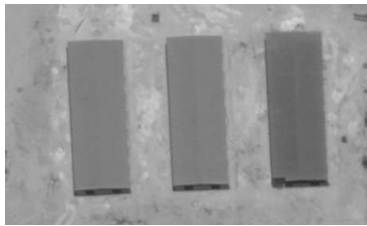


(e) Original image mask used for comparison.

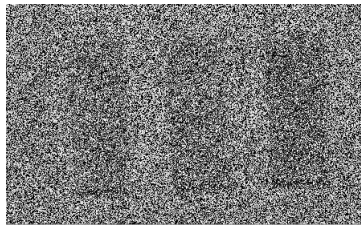


(f) The difference in the contours detected and the original mask used for comparison.

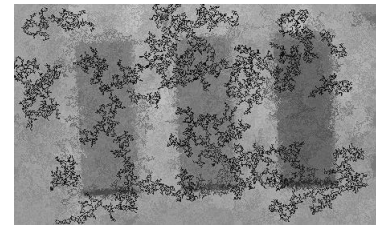
Figure 4.39: 20% Salt & pepper noise, $LULU_{conv}$ smoother and active contour detection algorithm. $FPFN = 82.59\%$



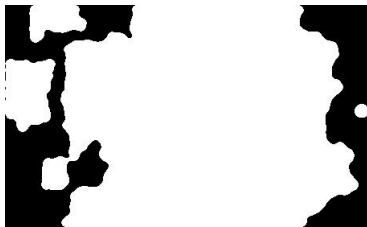
(a) Original image.



(b) 50% Salt & pepper noise.



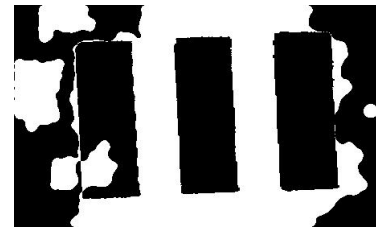
(c) Smoothed using the $LULU_{conv}$ smoother.



(d) Polygons detected using active contour detection.

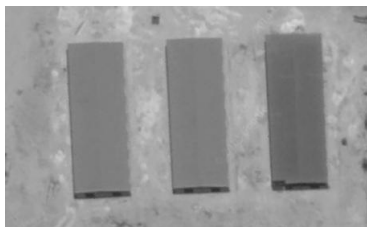


(e) Original image mask used for comparison.

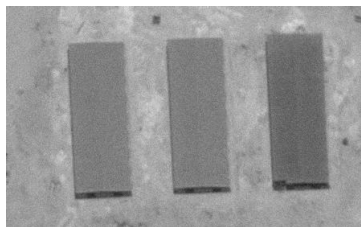


(f) The difference in the contours detected and the original mask used for comparison.

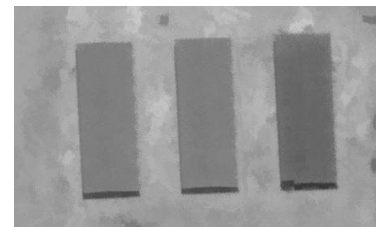
Figure 4.40: 50% Salt & pepper noise, $LULU_{conv}$ smoother and active contour detection algorithm. $FPFN = 41.46\%$



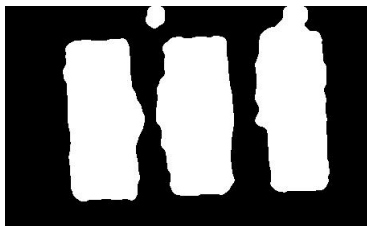
(a) Original image.



(b) 10% Speckle noise.



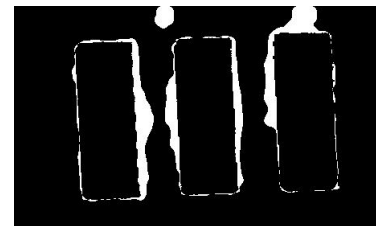
(c) Smoothed using the $LULU_{conv}$ smoother.



(d) Polygons detected using active contour detection.



(e) Original image mask used for comparison.



(f) The difference in the contours detected and the original mask used for comparison.

Figure 4.41: 10% Speckle noise, $LULU_{conv}$ smoother and active contour detection algorithm. $FPFN = 81.37\%$

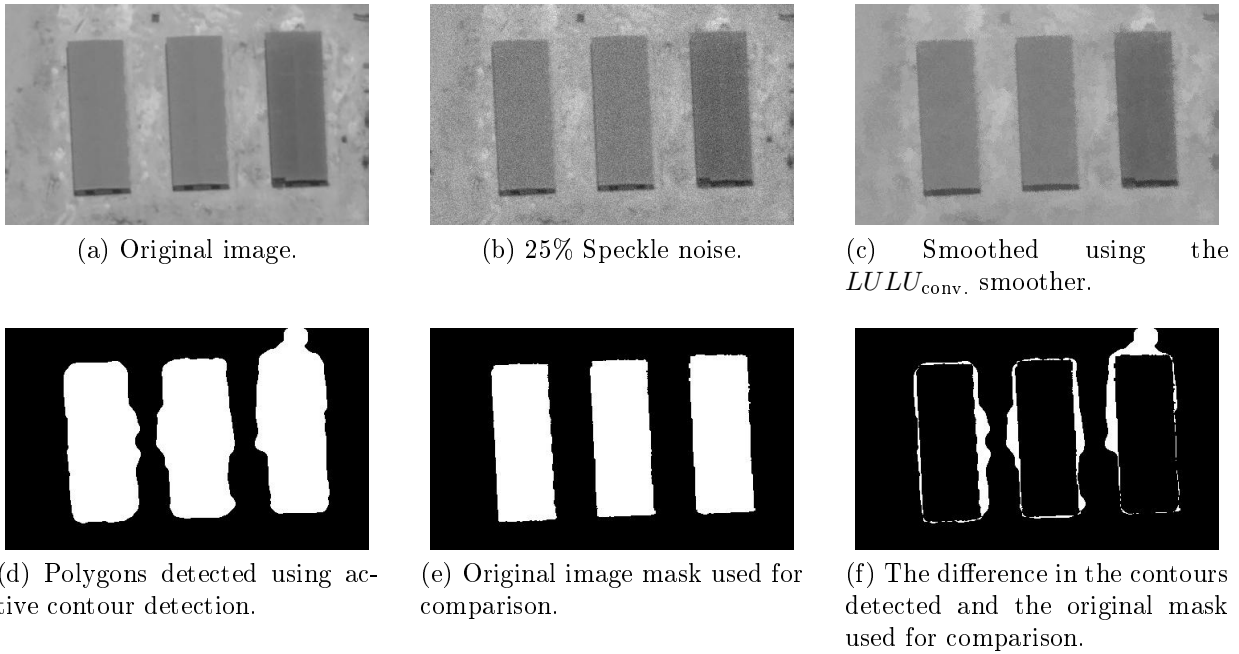


Figure 4.42: 25% Speckle noise, $LULU_{conv.}$ smoother and active contour detection algorithm. $FPFN = 83.32\%$

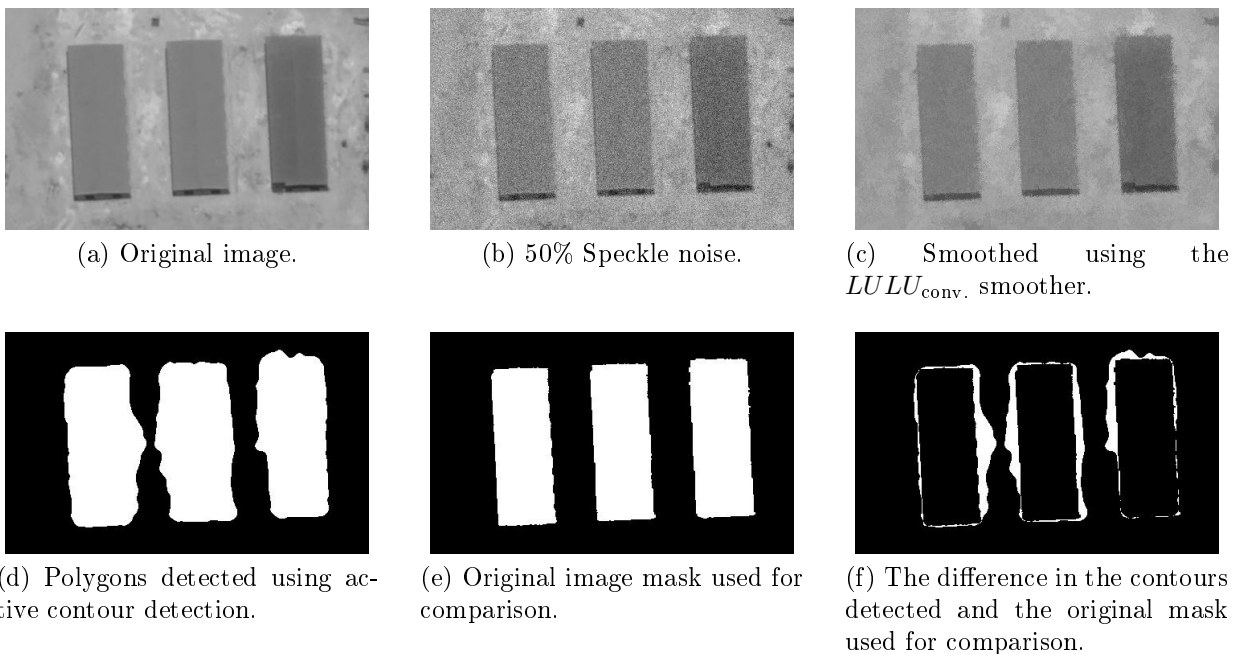


Figure 4.43: 50% Speckle noise, $LULU_{conv.}$ smoother and active contour detection algorithm. $FPFN = 82.70\%$

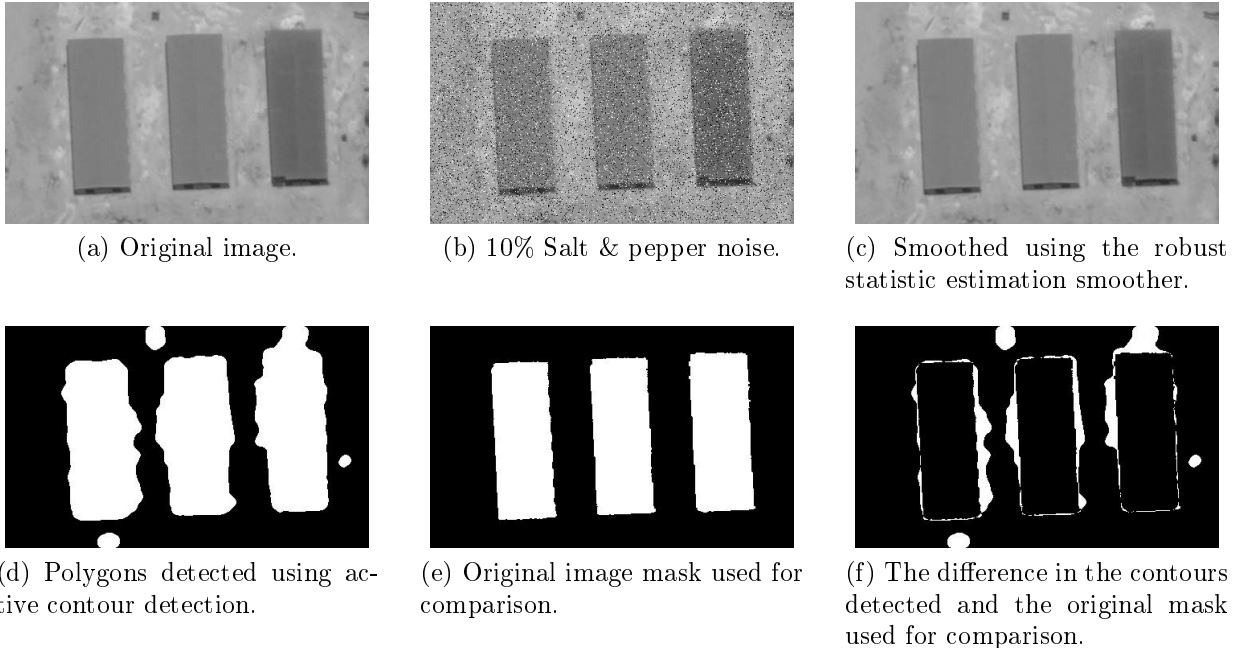
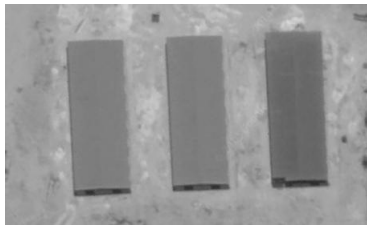


Figure 4.44: 10% Salt & pepper noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 81.46\%$

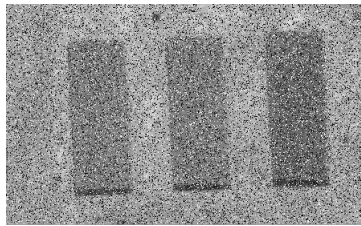
When comparing the three different contour detection algorithms over the various smoothers we make the following observations from Tables 4.1 to 4.3.

Watershed segmentation

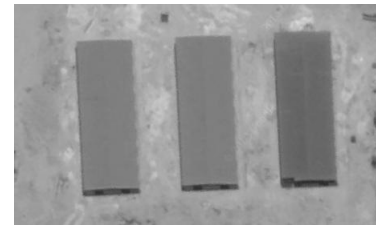
The watershed algorithm outperformed the other algorithms when 10% of salt & pepper noise was present in the polygon image. In almost all cases watershed algorithm performs either really well (close to perfect) or it doesn't perform well at all. This is due to the fact that the watershed algorithm didn't detect some objects at all under certain conditions. The $FPFN$ ratio was very low when the watershed algorithm was applied to the building image smoothed by the connected median filter when salt & pepper noise was present. Although the robust statistic based algorithm didn't get the best average rank score with the polygon image, it was the most consistent. Its performance deteriorated in the presence of speckle noise and ended in the worst performance of all the combinations at 50% speckle noise on the building image, but it was expected since the filter was explicitly designed for salt & pepper noise. The median filter performed well with the building image when speckle noise was present, but it was also to be expected since the noise followed a Gaussian distribution. The watershed segmentation did not consistently perform better than the other contour detection techniques on either of the two images. The watershed segmentation algorithm is not recommended if nothing is known about the type of noise present in the image. If the type of image is known and the type of noise is known then the watershed segmentation algorithm is highly recommended. For an image similar to our polygon toy image the watershed segmentation algorithm should be used together with either a the $LULU_{conv}$ smoother if speckle noise is present or the robust statistic based algorithm when salt & pepper noise is present. For images similar to our building image it is recommended that either the median image smoother



(a) Original image.



(b) 20% Salt & pepper noise.



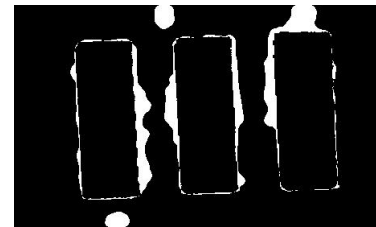
(c) Smoothed using the robust statistic estimation smoother.



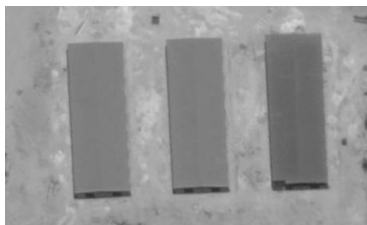
(d) Polygons detected using active contour detection.



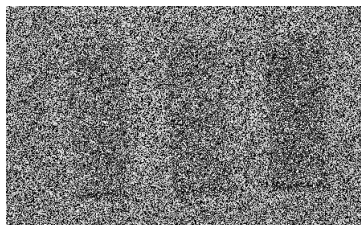
(e) Original image mask used for comparison.



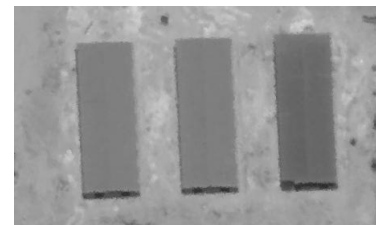
(f) The difference in the contours detected and the original mask used for comparison.

 Figure 4.45: 20% Salt & pepper noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 81.57\%$


(a) Original image.



(b) 50% Salt & pepper noise.



(c) Smoothed using the robust statistic estimation smoother.



(d) Polygons detected using active contour detection.

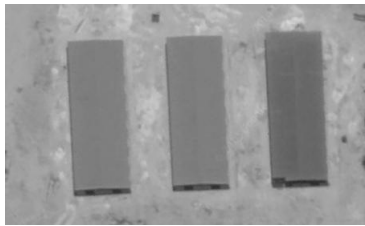


(e) Original image mask used for comparison.

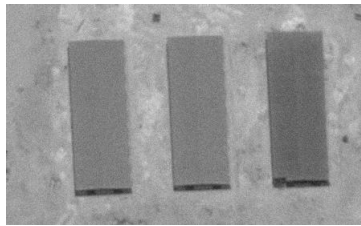


(f) The difference in the contours detected and the original mask used for comparison.

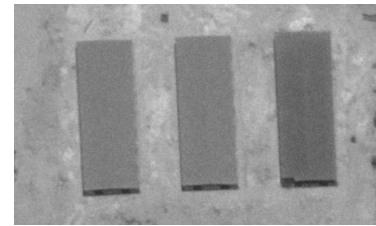
 Figure 4.46: 50% Salt & pepper noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 81.39\%$



(a) Original image.



(b) 10% Speckle noise.



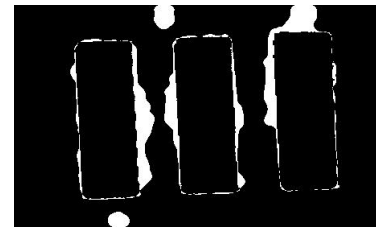
(c) Smoothed using the robust statistic estimation smoother.



(d) Polygons detected using active contour detection.

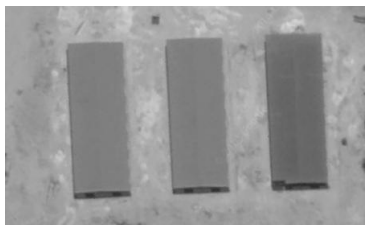


(e) Original image mask used for comparison.

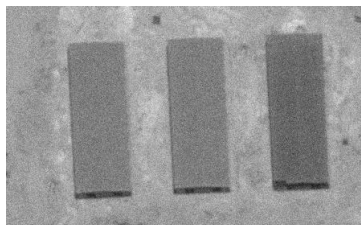


(f) The difference in the contours detected and the original mask used for comparison.

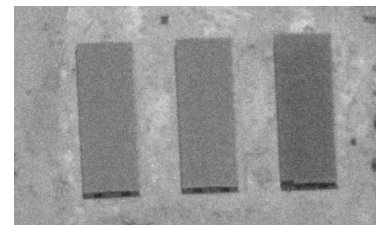
Figure 4.47: 10% Speckle noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 81.62\%$



(a) Original image.



(b) 25% Speckle noise.



(c) Smoothed using the robust statistic estimation smoother.



(d) Polygons detected using active contour detection.



(e) Original image mask used for comparison.



(f) The difference in the contours detected and the original mask used for comparison.

Figure 4.48: 25% Speckle noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 81.22\%$

FPFN (Image = Polygons)		Active Contours											
		FPFN of smoothed image					Rank FPFN of smoothed image						
		S&P		Speckle			Ranks		Ranks				
Filter/Noise	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	25%	50%	Average Rank
Median	92.63%	92.46%	20.70%	92.63%	92.46%	92.24%	2	4	6	5	4	2	3.83
Adaptive Median	92.93%	92.49%	45.16%	92.97%	92.08%	91.87%	1	3	2	2	5	4	2.83
Connected Median	92.48%	92.63%	20.86%	92.90%	92.66%	92.41%	4	2	5	4	2	1	3.00
LULU	92.59%	92.70%	22.21%	93.06%	92.71%	92.05%	3	1	4	4	1	3	2.17
Robust Statistic Estimate	91.68%	91.61%	91.27%	92.37%	91.36%	91.44%	6	5	1	6	6	6	5.00
None	92.45%	89.53%	24.01%	92.94%	92.47%	91.67%	5	6	3	3	3	5	4.17

(a)

FPFN (Image = Buildings)		Active Contours											
		FPFN of smoothed image					Rank FPFN of smoothed image						
		S&P		Speckle			Ranks		Ranks				
Filter/Noise	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	25%	50%	Average Rank
Median	81.19%	82.39%	40.98%	80.43%	80.60%	80.08%	4	5	4	5	6	6	5.00
Adaptive Median	81.28%	82.78%	61.30%	80.33%	82.26%	81.73%	3	3	2	6	2	4	3.33
Connected Median	80.92%	84.70%	35.61%	81.50%	80.69%	81.18%	6	2	5	3	5	5	4.33
LULU	82.47%	82.59%	41.46%	81.37%	83.32%	82.70%	1	4	3	4	1	2	2.50
Robust Statistic Estimate	81.46%	81.57%	81.39%	81.62%	81.22%	81.86%	2	6	1	2	4	3	3.00
None	81.04%	85.31%	30.54%	81.77%	81.57%	82.85%	5	1	6	1	3	1	2.83

(b)

Table 4.1: a.) False positive false negative ratio results for the active contour detection algorithm applied to the polygon image.
 b.) False positive false negative ratio results for the active contour detection algorithm applied to the real life buildings image.

FPFN (Image = Polygons)		Suzuki											
		FPFN of smoothed image					Rank FPFN of smoothed image						
		S&P		Speckle			Ranks						
Filter/Noise	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	25%	50%	Average Rank
Median	92.07%	87.21%	39.09%	93.50%	93.41%	92.48%	5	5	5	4	2	3	4.00
Adaptive Median	92.76%	91.93%	70.47%	93.42%	93.34%	92.82%	2	2	2	5	3	1	2.50
Connected Median	92.80%	91.27%	46.53%	93.51%	93.44%	92.63%	1	3	4	2	1	2	2.17
LULU	92.29%	91.05%	48.86%	93.51%	93.32%	92.43%	4	4	3	2	4	4	3.50
Robust Statistic Estimate	92.62%	92.50%	89.14%	92.81%	92.27%	88.07%	3	1	1	6	5	5	3.50
None	71.88%	56.81%	27.36%	93.51%	91.63%	77.82%	6	6	6	1	6	6	5.17

(a)

FPFN (Image = Buildings)		Suzuki											
		FPFN of smoothed image					Rank FPFN of smoothed image						
		S&P		Speckle			Ranks						
Filter/Noise	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	25%	50%	Average Rank
Median	92.02%	91.52%	11.48%	92.16%	91.53%	90.39%	2	2	6	2	1	1	2.33
Adaptive Median	91.56%	90.89%	76.25%	91.75%	91.04%	90.01%	4	4	2	3	3	2	3.00
Connected Median	90.72%	88.03%	19.80%	91.06%	89.42%	85.16%	5	5	4	4	4	4	4.33
LULU	92.12%	90.97%	17.24%	92.49%	91.28%	88.45%	1	3	5	1	2	3	2.50
Robust Statistic Estimate	91.69%	91.58%	91.08%	90.52%	86.37%	75.27%	3	1	1	5	5	5	3.33
None	9.18%	13.99%	32.96%	5.45%	9.73%	16.79%	6	6	3	6	6	6	5.50

(b)

Table 4.2: a.) False positive, false negative ratio results for the border following algorithm applied to the polygon image. b.) False positive false negative ratio results for the border following algorithm applied to the real life buildings image.

FPFN (Image = Polygons)		Watershed													
		FPFN of smoothed image							Rank FPFN of smoothed image						
		S&P			Speckle				Ranks			Ranks			
Filter/Noise	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	25%	50%	Average Rank		
Median	99.14%	18.12%	18.19%	18.19%	99.91%	99.77%	99.10%	3	6	4	4	3	4	4.00	
Adaptive Median	99.23%	98.25%	83.10%	99.88%	99.74%	99.22%	2	4	2	2	5	5	2	3.33	
Connected Median	99.31%	98.66%	18.15%	99.97%	99.77%	99.16%	1	2	5	1	2	3	3	2.33	
LULU	99.09%	98.34%	18.32%	99.95%	99.79%	99.30%	4	3	3	3	1	1	1	2.50	
Robust Statistic Estimate	99.05%	98.86%	95.79%	99.24%	98.76%	96.37%	5	1	1	6	6	6	6	4.17	
None	18.15%	18.13%	18.11%	99.96%	99.77%	96.78%	6	5	6	2	4	5	5	4.67	

(a)

FPFN (Image = Buildings)		Watershed													
		FPFN of smoothed image							Rank FPFN of smoothed image						
		S&P			Speckle				Ranks			Ranks			
Filter/Noise	10%	20%	50%	10%	25%	50%	10%	20%	50%	10%	25%	50%	Average Rank		
Median	96.47%	95.24%	34.17%	96.15%	94.97%	95.34%	1	2	4	1	1	1	1	1.67	
Adaptive Median	94.92%	94.15%	35.17%	93.82%	92.65%	62.67%	4	3	2	5	4	3	3	3.50	
Connected Median	32.80%	32.21%	33.31%	92.67%	94.29%	31.62%	6	6	5	6	2	5	5	5.00	
LULU	95.91%	93.86%	32.88%	95.39%	93.86%	91.70%	2	4	6	2	3	2	2	3.17	
Robust Statistic Estimate	95.57%	95.46%	94.75%	94.62%	63.34%	2.90%	3	1	1	3	5	6	6	3.17	
None	35.14%	37.46%	35.13%	94.29%	31.35%	33.33%	5	5	3	4	6	4	4	4.50	

(b)

Table 4.3: a.) False positive, false negative ratio results for the watershed algorithm applied to the polygon image. b.) False positive false negative ratio results for the watershed algorithm applied to the real life buildings image.

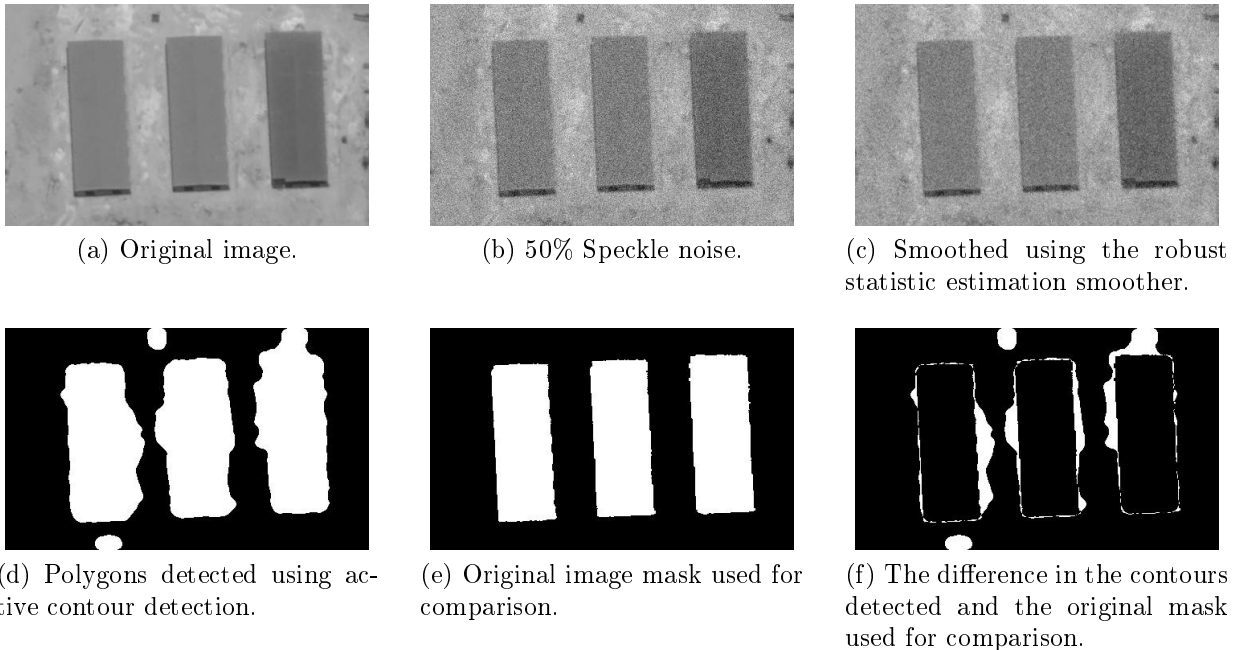


Figure 4.49: 50% Speckle noise, robust statistic estimation smoother and active contour detection algorithm. $FPFN = 82.85\%$

should be used if speckle noise is present or the robust statistic based algorithm should be used if salt & pepper noise is present together with the watershed segmentation algorithm.

Active contours

Active contours gave the most consistent performance on the building image with 50% salt & pepper noise. Relative high $FPFN$ ratios were also obtained when high amounts of speckle noise was present in the building image. We note again that the robust statistic based algorithm was again the only image smoother which continued to do well even with 50% salt & pepper noise in both images. We see that active contours did worse than the watershed segmentation algorithm and the border following algorithm in the cases where they didn't completely malfunction because of the image noise, but the extent to which the active contour algorithm failed is a lot less compared to the other algorithms leading us to the conclusion that it is a robust contour detection technique. The consistency obtained on the building image when used in conjunction with either the $LULU_{conv}$ or the robust statistics based algorithm makes this contour detection technique our preferred choice when extracting object from images similar to our building image if nothing is known about the type of noise in the image.

Border following algorithm (BFA) by Suzuki

With the polygon image the BFA's worst performance was almost twice as good as the other contour detection techniques' worst performance. Under lower noise condition the BFA performed better than the active contour algorithm but worse than the watershed segmentation algorithm. With the building image the BFA performed a lot worse compared to the other algorithms, failing

a lot more and obtaining very low minimum *FPFN* ratios. This is a good example of a contour detection technique that performs very well on certain types of images and not well at all on others. The adaptive median and robust statistic based algorithm performed best on the polygon toy image and together with those two image smoothing techniques they form our recommendation of algorithms to use to extract objects from images that are similar to our polygon toy image if nothing is known about the type of noise in the image.

4.3 Conclusion

In this chapter we combined various images with two different types of image noise at three levels of noise intensity. We applied five image smoothing techniques in order to remove some of the applied noise and used three contour detection techniques to extract the objects in the images. There were 180 results in total and we compared them in terms of how accurately the object were extracted from the images using a false positive, false negative ratio. It was seen that some of the image smoothers give better results under certain circumstances and the same with the various contour detection algorithms. The results of the the artificial polygon image were in most cases better and more consistent compared to the real life building image. Our recommendation is that if there is no prior knowledge of what kind of noise is present in the image then either the $LULU_{conv}$ or the robust statistics based algorithm, together with active contour detection algorithm should be used when extracting object from images similar to our building image and the adaptive median or robust statistic based algorithm should be used, together with the border following algorithm by Suzuki, to extract objects from images that are similar to our polygon toy image. If it is known what kind of noise presents itself in the image then the watershed segmentation algorithm should be preferred together with the optimal image smoother as discussed in Section 4.2. Even though some combinations of algorithms compared better than other, the exact magnitude and significance should be weighed against the computational cost and tractability of the algorithms involved. In Tables 4.1, 4.2 and 4.3 an performance rank was given to each combination of algorithms, but this should be considered together with the actual performance of using a specific combination of algorithm to make a final decision about which combination to use. This result will enables future researchers to use an optimal combination of image smoothing techniques and contour detection algorithms when they know the type of image and type of image noise they are dealing with.

Chapter 5

Conclusion

A brief overview of the field of mathematical morphology was given in Chapter 2. We discussed five different image smoothing methods to be used to decrease the two types of noise in images at different intensity levels. A new image smoother was introduced called the connected median filter for images. We compared the new image smoother with existing image smoothers and found its performance to be similar and in some instances better. Three methods that can be used to detect and extract contours in images were investigated in Chapter 3. We combined two different types of noise, each at three different intensity level with two separate types of images. These images were then smoothed using five image smoothers including our new proposed image smoother. The smoothed images were then subjected to the three contour detection algorithms and we compared the results to the ground truth images using a false positive, false negative accuracy method in Chapter 4. The results showed that prior information of image type and image noise play a big part in detecting and extracting objects in images successfully and accurately. The results were summarised so that future researchers can easily identify a suitable combination of algorithms to use subject to their circumstances. In the future we wish to expand the variety of noise in our investigation with various types of noise distributions when applying these algorithms. The identification of groups of building with walls and other regular objects in images should also be addressed in future work. A greater sample of input images should also be used to better understand the affects and failure modes of different kinds of images. The connected median filter algorithm could also be adapted to include elements of the robust statistic estimation algorithm to perform similarly in conditions where salt & pepper noise is present. This study should also be repeated with many images in order to obtain distributions of false positive, false negative ratios to determine if the difference in performance between different combinations of algorithms are significant or not.

Bibliography

- [1] S S Al-amri, N V Kalyankar Kalyankar, and S D Khamitkar. A comparative study of removal noise from remote sensing image. *International Journal of Computer Science Issues (IJCSI)*, Vol. 7, Issue. 1, No. 1, January 2010, 7(1), 2010.
- [2] Shaukat Ali, Ram M Tiwari, and Gordon B Snow. False-positive and false-negative neck nodes. *Head & Neck Surgery*, 8(2):78–82, 1985.
- [3] Philippe Andrey and Thomas Boudier. Adaptive active contours (snakes) for the segmentation of complex structures in biological images. In *Proceedings of ImageJ Conference, Luxembourg, Belgium*, page 18, 2006.
- [4] R Anguelov and I N Fabris-Rotelli. LULU operators and discrete pulse transform for multi-dimensional arrays. *Image Processing, IEEE Transactions on*, 19(11):3012–3023, 2010.
- [5] Gilles Aubert, Michel Barlaud, Olivier Faugeras, and Stéphanie Jehan-Besson. Image segmentation using active contours: Calculus of variations or shape gradients? *SIAM Journal on Applied Mathematics*, 63(6):2128–2154, 2003.
- [6] Serge Beucher. Watersheds of functions and picture segmentation. In *Acoustics, Speech, and Signal Processing, Proceedings of IEEE International Conference on ICASSP'82.*, volume 7, pages 1928–1931. IEEE, 1982.
- [7] Serge Beucher and Christian Lantuejoul. Use of watersheds in contour detection. In *Proceedings of International Workshop Image Processing, Real-Time Edge and Motion Detection/Estimation*, Rennes, France, September 1979.
- [8] Serge Beucher and Fernand Meyer. The morphological approach to segmentation: the watershed transformation. In *Mathematical morphology in image processing*, pages 433–481. 1993.
- [9] André Bleau, Jacques de Guise, A Leblanc, et al. A new set of fast algorithms for mathematical morphology: I. idempotent geodesic transforms. *CVGIP: Image Understanding*, 56(2):178–209, 1992.
- [10] Alan C Bovik. *Handbook of Image and Video Processing*. Academic Press, 2010.
- [11] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of Graphics, GPU, and Game Tools*, 12(2):13–21, 2007.

- [12] Raymond H Chan, Chung-Wa Ho, and Mila Nikolova. Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization. *Image Processing, IEEE Transactions on*, 14(10):1479–1485, 2005.
- [13] S Grace Chang, Bin Yu, and Martin Vetterli. Adaptive wavelet thresholding for image denoising and compression. *Image Processing, IEEE Transactions on*, 9(9):1532–1546, 2000.
- [14] Willie J Conradie and Tertius De Wet. LULU smoothers on online data. *South African Statistical Journal*, 46(2):185–220, 2012.
- [15] WJ Conradie, T De Wet, and M Jankowitz. Exact and asymptotic distributions of LULU smoothers. *Journal of Computational and Applied Mathematics*, 186(1):253–267, 2006.
- [16] Sanford Eisenhandler, Paul W Lyons, Stuart S Perlman, and Michael J Shumila. Adaptive median filter system, July 21 1987. US Patent 4,682,230.
- [17] Inger Nicolette Fabris-Rotelli. *LULU operators on multidimensional arrays and applications*. PhD thesis, University of Pretoria, 2009.
- [18] Hironobu Fujiyoshi and Alan J Lipton. Real-time human motion analysis by image skeletonization. In *Proceedings of Applications of Computer Vision, 1998. WACV'98. , Fourth IEEE Workshop on*, pages 15–21. IEEE, 1998.
- [19] Joseph W Goodman. Some fundamental properties of speckle. *JOSA*, 66(11):1145–1150, 1976.
- [20] Hugo Hadwiger. *Vorlesung über Inhalt, Oberfläche und Isoperimetrie*. Springer-Verlag, 1957.
- [21] TD Haig, Y Attikiouzel, and MD Alder. Border following: new definition gives improved borders. In *Communications, Speech and Vision, IEE Proceedings I*, volume 139, pages 206–211. IET, 1992.
- [22] Geoffrey Hellman. Mathematical constructivism in spacetime. *The British Journal for the Philosophy of Science*, 49(3):425–450, 1998.
- [23] Ren Honge, Zhong Qiubo, and Kang Junfeng. Object recognition algorithm research based on variable illumination. In *Automation and Logistics, 2009. ICAL'09. Proceedings of IEEE International Conference on*, pages 1609–1613. IEEE, 2009.
- [24] T Huang, G Yang, and G Tang. A fast two-dimensional median filtering algorithm. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 27(1):13–18, 1979.
- [25] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.
- [26] Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics Letters*, 44(13):800–801, 2008.
- [27] Humor Hwang and R Haddad. Adaptive median filters: new algorithms and results. *Image Processing, IEEE Transactions on*, 4(4):499–502, 1995.
- [28] Anil K Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Inc., 1989.

- [29] Ramesh Jain, Rangachar Kasturi, and Brian G Schunck. *Machine Vision*, volume 5. McGraw-Hill New York, 1995.
- [30] Maria Dorothea Jankowitz. *Some statistical aspects of LULU smoothers*. PhD thesis, Stellenbosch: University of Stellenbosch, 2007.
- [31] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2015-06-01].
- [32] BI Justusson. *Median Filtering: Statistical Properties*. Springer, 1981.
- [33] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.
- [34] Daniel Kelly, John McDonald, and Charles Markham. A person independent system for recognition of hand postures used in sign language. *Pattern Recognition Letters*, 31(11):1359–1368, 2010.
- [35] T Yung Kong and Azriel Rosenfeld. Digital topology: introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, 1989.
- [36] Dirk P Laurie and Carl H Rohwer. Fast implementation of the discrete pulse transform. In *Proceedings of International Conference Numerical Analysis and Applied Mathematics*, pages 15–19. Weinheim, Germany, 2006.
- [37] Frederic Leymarie and Martin D. Levine. Tracking deformable objects in the plane using an active contour model. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(6):617–634, 1993.
- [38] Georges Matheron and Jean Serra. The birth of mathematical morphology. In *Proceedings of the 6th International Symposium of Mathematical Morphology*, pages 1–16. Sydney, Australia, 2002.
- [39] François Mendels, Conor Heneghan, and Jean-Philippe Thiran. Identification of the optic disk boundary in retinal images using active contours. In *Proceedings of the Irish Machine Vision and Image Processing Conference*, pages 103–115. Citeseer, 1999.
- [40] Fernand Meyer and Serge Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–46, 1990.
- [41] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311–324, 2007.
- [42] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.
- [43] Theodosios Pavlidis. *Algorithms for Graphics and Image Processing*. Computer science press, 1982.

- [44] Shengcai Peng and Lixu Gu. A novel implementation of watershed transform using multi-degree immersion simulation. In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. Proceedings of 27th Annual International Conference of the*, pages 1754–1757. IEEE, 2006.
- [45] Roushanak Rahmat, Aamir Saeed Malik, and Nidal Kamel. Comparison of LULU and median filter for image denoising. *International Journal of Computer & Electrical Engineering*, 5(6), 2013.
- [46] John A Richards and Xiuping Jia. *Remote Sensing Digital Image Analysis*, volume 3. Springer, 1999.
- [47] Carl H Rohwer and Marcel Wild. LULU theory, idempotent stack filters, and the mathematics of vision of marr. *Advances in Imaging and Electron Physics*, 146:58, 2007.
- [48] CH Rohwer. Idempotent one-sided approximation of median smoothers. *Journal of Approximation Theory*, 58(2):151–163, 1989.
- [49] CH Rohwer and DP Laurie. The discrete pulse transform. *SIAM Journal on Mathematical Analysis*, 38(3):1012–1034, 2006.
- [50] CH Rohwer and LM Toerien. Locally monotone robust approximation of sequences. *Journal of Computational and Applied Mathematics*, 36(3):399–408, 1991.
- [51] CH Rohwer and M Wild. Natural alternatives for one dimensional median filtering. *Quaestiones Mathematicae*, 25(2):135–162, 2002.
- [52] Azriel Rosenfeld and Avinash C Kak. *Digital Picture Processing*, volume 1. Elsevier, 2014.
- [53] Franz Rottensteiner, John Trinder, Simon Clode, and Kurt Kubik. Using the dempstershafer method for the fusion of lidar data and multi-spectral images for building detection. *Information Fusion*, 6(4):283 – 300, 2005. Fusion of Remotely Sensed Data over Urban Areas.
- [54] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.
- [55] I Saارينen. Color image segmentation by a watershed algorithm and region adjacency graph processing. In *Image processing, 1994. ICIP-94., Proceedings of IEEE international conference*, volume 3, pages 1021–1025. IEEE, 1994.
- [56] Jean Serra. Mathematical morphology for boolean lattices. *Image Analysis and Mathematical Morphology, II: Theoretical Advances*, pages 37–58, 1988.
- [57] F.Y. Shih. *Image Processing and Pattern Recognition: Fundamentals and Techniques*. Wiley, 2010.
- [58] P. Soille. *Morphological Image Analysis: Principles and Applications*. Springer, 1999.
- [59] Satoshi Suzuki. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

- [60] John W Tukey. *Exploratory Data Analysis*. Reading, Mass., 1977.
- [61] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.
- [62] VR Vijaykumar, PT Vanathi, P Kanagasabapathy, and D Ebenezer. Robust statistics based algorithm to remove salt and pepper noise in images. *International Journal of Information and Communication Engineering*, 5(3):164–173, 2009.
- [63] Luc Vincent and Pierre Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, 1991.
- [64] Changhong Wang, Taoyi Chen, and Zhenshen Qu. A novel improved median filter for salt-and-pepper noise from highly corrupted images. In *Systems and Control in Aeronautics and Astronautics (ISSCAA), Proceedings of 2010 3rd International Symposium on*, pages 718–722. IEEE, 2010.
- [65] Chenyang Xu, Anthony Yezzi Jr, and Jerry L Prince. On the relationship between parametric and geometric active contours. In *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on*, volume 1, pages 483–489. IEEE, 2000.
- [66] O Yli-Harja, J Astola, and Y Neuvo. Statistical properties of median, weighted median and stack filters. In *Circuit Theory and Design, 1989., Proceedings of European Conference on*, pages 354–357. IET, 1989.
- [67] Shigeki Yokoi, Jun-Ichiro Toriwaki, and Teruo Fukumura. An analysis of topological properties of digitized binary pictures using local features. *Computer Graphics and Image Processing*, 4(1):63–73, 1975.
- [68] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively-variant textures on arbitrary surfaces. In *Proceedings of ACM Transactions on Graphics (TOG)*, volume 22, pages 295–302. ACM, 2003.
- [69] Xiangrong Zhou, Takeshi Hara, Hiroshi Fujita, Ryujiro Yokoyama, Takuji Kiryu, and Hiroaki Hoshi. Automated segmentations of skin, soft-tissue, and skeleton, from torso ct images. In *Proceedings of Medical Imaging 2004*, pages 1634–1639. International Society for Optics and Photonics, 2004.

Appendix

All code available at <https://github.com/Magnusvann/WST895>.

Binary mathematical morphology code [61, 25, 31]

```
from demo import load_image
import matplotlib.pyplot as plt
from skimage
import morphology as m
import skimage import numpy as np
img = load_image('woofs2.jpg')
img = img[0:100,0:100]
img_bin = img>120
plt.subplot(3, 2, 1)
plt.imshow(img, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Original Image')
plt.xticks([])
plt.yticks([])
plt.subplot(3, 2, 2)
plt.imshow(img_bin, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Binary Image')
plt.xticks([])
plt.yticks([])
SE = np.array([[0,1,0],
               [1,1,1],
               [0,1,0]], dtype=bool)
img_erode = m.binary_erosion(img_bin, SE)
plt.subplot(3, 2, 3)
plt.imshow(img_erode, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Eroded Image')
plt.xticks([])
plt.yticks([])
img_dilate = m.binary_dilation(img_bin, SE)
plt.subplot(3, 2, 4)
plt.imshow(img_dilate, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Dilated Image')
plt.xticks([])
```

```

plt.yticks ([])
img_opened = m.binary_opening(img_bin,SE)
plt.subplot(3, 2, 5)
plt.imshow(img_opened, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Opened Image')
plt.xticks ([])
plt.yticks ([])
img_closed = m.binary_closing(img_bin,SE)
plt.subplot(3, 2, 6)
plt.imshow(img_closed, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Closed Image')
plt.xticks ([])
plt.yticks ([])
plt.show()

```

Grey scale mathematical morphology code[61, 25, 31]

```

from demo import load_image
import matplotlib.pyplot as plt
import pymorph as pm from skimage.io
import imsave
img = load_image('woofs2.jpg')
img = img[0:100,0:100]
se = pm.secross(r=1)
#Mathematical Morphology Operator
img_result = pm.erode(img, b=se)
#img_result = pm.dilate(img, B=se)
#img_result = pm.open(img, b = se)
#img_result = pm.close(img, Bc = se)
imsave('output/img.jpg', img)
#plt.imshow('output/se2.jpg', se, cmap=plt.cm.gray)
imsave('output/erode_se1.jpg', img_result)

```

Robust statistic based algorithm code[61, 25, 31]

```

import skimage.morphology as mor import numpy as np
test_img = np.array([ [9, 9, 8, 2, 6, 6],
                      [9, 9, 2, 10, 8, 6],
                      [10, 10, 10, 10, 10, 3],
                      [6, 3, 2, 10, 2, 10],
                      [6, 6, 4, 9, 10, 10] ])
def RobustStatisticBased(img = test_img, zeta = 0.3, L_max = 3, prints \
= False):
    new_img = np.zeros_like(img)
    for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        for L in range(1, L_max):

```

```

    eval = img[max(0,i-L):(min(img.shape[0],i+L)+1), \
max(0,j-L): min(img.shape[1],j+L)+1]
        if prints == True:
            print 'i,j,L = ',i,j,L
            print 'eval: \n',eval
            eval_list = eval#[eval>0]
if prints == True:
    print 'eval_list: \n',eval_list
    f_ij = img[i][j]
if prints == True:
    print 'f_ij:',f_ij
    #if len(np.unique(eval_list)) > 0:
eval_min = np.min(eval_list)
eval_max = np.max(eval_list)
eval_med = np.median(eval_list)
#else:
#    eval_min = f_ij
#    eval_max = f_ij
#    eval_med = f_ij
if prints == True:
    print 'eval_min: ', eval_min
    print 'eval_med: ', eval_med
    print 'eval_max: ', eval_max
if eval_min < eval_med < eval_max:
    if eval_min < f_ij < eval_max:
        if prints == True:
            print 'f_ij unchanged!'
        break
    else:
        mid = eval_list[eval_list > eval_min]
        mid = mid[mid < eval_max]
        if prints == True:
            print 'mid',mid
        if np.unique(mid).shape[0] <= 1 and L < L_max:
            L = L + 1
            if prints == True:
                print 'Increasing L to L =',L
    else:
        tau = zeta*mid.std()
        if prints == True:
            print 'tau:',tau.astype(np.float16)
        t = tau/np.sqrt(2)
        if prints == True:
            print 't:',t
        x = mid - np.median(eval_list)
        if prints == True:

```

```

        print 'x:', x
        mid = (x!=0)*mid
        mid = mid[mid!=0]
        if prints == True:
            print 'mid:', mid
        x = x[x!=0]
        if prints == True:
            print 'x:', x
        r = 2*x/(t**2+x**2)
        if prints == True:
            print 'r:', r
        S1 = np.sum(mid*r/x)
        if prints == True:
            print 'S1:', S1
        S2 = np.sum(r/x)
        if prints == True:
            print 'S2:', S2
        f_ij = np.round(S1/S2)
        if prints == True:
            print 'long method. f_ij changed \
to: ', f_ij
            break
    else:
        mid = eval_list[eval_list > eval_min]
        mid = mid[mid < eval_max]
        if prints == True:
            print 'mid', mid
        if np.unique(mid).shape[0] <= 1 and L < L_max:
            L = L + 1
            if prints == True:
                print 'Increasing L to L =', L
        else:
            tau = zeta*mid.std()
            if prints == True:
                print 'tau:', tau.astype(np.float16)
            t = tau/np.sqrt(2)
            if prints == True:
                print 't:', t
            x = mid - np.median(eval_list)
            if prints == True:
                print 'x:', x
            mid = (x!=0)*mid
            mid = mid[mid!=0]
            if prints == True:
                print 'mid:', mid
            x = x[x!=0]

```

```

        if prints == True:
            print 'x:',x
        r = 2*x/(t**2+x**2)
        if prints == True:
            print 'r:',r
        S1 = np.sum(mid*r/x)
        if prints == True:
            print 'S1:',S1
        S2 = np.sum(r/x)
        if prints == True:
            print 'S2:',S2
        f_ij = np.round(S1/S2)
        if prints == True:
            print 'long method. f_ij changed to: ',f_ij
        break
    new_img[i][j] = f_ij
    new_img = new_img.astype(int)
if prints == True:
    print 'New image: \n', new_img
    if np.array_equal(img,test_img):
        assert (new_img.all() == np.array([ \
            [ 9,  9,  8,  6,  6,  6],
            [ 9,  9,  8,  6,  8,  6],
            [ 6,  8,  8,  7,  7,  3],
            [ 6,  3,  5,  6,  6, 10],
            [ 6,  3,  4,  9, 10, 10]]).all()), "BROKEN!")
if prints == True:
    print "Success!"
return new_img

```

LULU smoother code[61, 25, 31]

```

import demo as d import lulu import numpy as np
def lulu_filter(noisy,sf):
    prev_smoothed = noisy
    pulses = lulu.decompose(img=noisy,quiet=False,operator='LU')
    smoothness_inc = 0.01
    pulse_len = len(pulses.keys())
    pulses
    i=2
    while 1-smoothness_inc > sf and i < pulse_len:
        lulu_smoothed, areas, area_count = \
lulu.reconstruct(dict((k, pulses[k]) for k in pulses.keys()[0:i]), \
noisy.shape)
        lulu_smoothed = noisy - lulu_smoothed
        smoothness_inc = np.var(lulu_smoothed)/np.var(prev_smoothed)

```

```

    prev_smoothed = lulu_smoothed
    i = i + 1
return lulu_smoothed

```

Connected median filter code[61, 25, 31]

```

import skimage.measure as meas
import skimage.morphology as mor
import numpy as np
from scipy.stats
import itemfreq
test_img = np.array([[9, 9, 8, 2, 6, 6],
                    [9, 9, 2, 10, 8, 6],
                    [10, 10, 10, 10, 10, 3],
                    [6, 3, 2, 10, 2, 10],
                    [6, 6, 4, 9, 10, 10]])

def p_list(connpnts,p):
list = []
freq = itemfreq(connpnts)
for i in range(freq.shape[0]):
    if freq[i][1] == p:
        list.append(i)
return list

def ConnectedMedian(img = test_img ,p_max=10, n_max=5,connectivity=4):
    #print "Original image: \n", img
img = img + (img == 0)*1
ori_img = img.copy()
for p in range(1,p_max+1):
    #print "For p =",p
    connpnts = meas.label(img, neighbors=connectivity)
    #print "Connected sets matrix: \n", connpnts
    connpnts_list = p_list(connpnts,p)
    #print "Connected sets of size ",p,":\n",connpnts_list
    new_img = np.zeros_like(img)
        #Loop through the connected sets of size p
    for i in connpnts_list:
        #Isolate a connected set of size p
        conn_comp_bin = (connpnts == i)
        #print "Binary mask of connected set: \n", conn_comp_bin
        for n in range(1,n_max+1): #Change SE size if required
            #print "for n =",n
            bindil = mor.binary_dilation(conn_comp_bin, mor.disk(n))
            #print "Binary mask after dilation: \n", bindil
                #Image data for connected set
            conn_comp_img = conn_comp_bin*img
            #print "Binary mask applied to image: \n", conn_comp_img

```



```

conn_comp_img_dil = bindil*img
a_list = conn_comp_img_dil[conn_comp_img_dil!=0]
#print "List of pixel values covered by the mask: \n" \
      , a_list
f_ij = np.max(conn_comp_img)
#print "f_ij: \n", f_ij
if len(np.unique(a_list)) > 0:
    K_min = np.min(a_list)
    K_med = np.median(a_list)
    K_max = np.max(a_list)
else:
    break
#print "K_min: \n", K_min
#print "K_med: \n", K_med.astype(int)
#print "K_max: \n", K_max
#Decision rules.
if K_min < K_med < K_max:
    if f_ij == K_max or f_ij == K_min:
        new_f_ij = K_med
        #print "f_ij changed! New f_ij =" \
              ,new_f_ij.astype(int)
        conn_comp_img = (conn_comp_bin==1)*new_f_ij
    #else:
        #print "f_ij did not change."
        break
elif n == n_max:
    new_f_ij = K_med
    #print "f_ij changed! New f_ij=", \
          new_f_ij.astype(int)
    conn_comp_img = (conn_comp_bin==1)*new_f_ij
    new_img = new_img + conn_comp_img
    #print "New Image: \n",new_img.astype(int)
    img = new_img.astype(int) + (new_img.astype(int) == 0)*img
    #print "Final Image after p=",p,"was processed.\n",img.astype(int)
if np.array_equal(ori_img, test_img):
    assert (img == np.array( [[9, 9, 8, 7, 6, 6],
                             [9, 9, 9, 9, 8, 6],
                             [9, 9, 9, 9, 9, 8],
                             [6, 3, 4, 9, 8, 9],
                             [6, 6, 4, 9, 9, 9]])).all(), "BROKEN!"

    #print "Success!"
return img

```

Batch run smoother code[61, 25, 31]

```
import demo as d
```

```

import os from skimage.io
import imsave from skimage.filter.rank
import median from skimage.morphology
import disk import numpy as np
import lulu_filter as lulu
infolder = "./data/"
for root, dirs, files in os.walk(infolder):
    for name in files:
        print name
        img = d.load_image('../data/'+name)
        lulu_smoothed = lulu.lulu_filter(img,0.5)
        imsave('../data/' + name,lulu_smoothed)

```

PSNR code[61, 25, 31]

```

import demo as d
import os from math
import log10
import numpy as np
infolder = "./PSNR/polys/"
for root, dirs, files in os.walk(infolder):
    for name in files:
        img = d.load_image('../PSNR/original_polys.jpg')
        smoothed = d.load_image('../PSNR/polys/'+name)
        SNR = 10*log10(255**2/(float(1)/float(img.size) \
*np.sum((smoothed-img)*(smoothed-img))))
        print name, SNR
        infolder = "./PSNR/buildings/"
for root, dirs, files in os.walk(infolder):
    for name in files:
        img = d.load_image('../PSNR/original_buildings.jpg')
        smoothed = d.load_image('../PSNR/buildings/'+name)
        SNR = 10*log10(255**2/(float(1)/float(img.size) \
*np.sum((smoothed-img)*(smoothed-img))))
        print name, SNR

```

Border following algorithm (Suzuki) code[61, 25, 31]

```

import numpy as np
from matplotlib import pyplot as plt
import demo as d
import cv2 from skimage.filter
import threshold_otsu
img = d.load_image("buildings.jpg").astype(np.uint8)
bin_img_ori = (d.load_image("buildings_bin_ori.jpg")>125)*1
noisy = d.load_image("../output/smoothed/rsb_buildings_S&P_10_noisy.jpg")
threshold_global_otsu = threshold_otsu(img)

```

```

suz = np.zeros_like(noisy)
threshold_global_otsu = threshold_otsu(noisy)
bin_img = ((noisy>threshold_global_otsu)*255).astype(np.uint8)
contours, hierarchy= \
cv2.findContours(bin_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(suz, contours, -1, (250, 250, 250), -1)
#Last argument 2 if only the border is required suz = ((suz == 0)*1)
suz[245:250, :] = bin_img_ori[245:250, :]
comb = np.clip(bin_img_ori + suz, 0, 1)
sens = float(np.sum(np.logical_and(suz == bin_img_ori, bin_img_ori!=0))) \
/float(np.sum(comb))
diff = comb - np.logical_and(suz == bin_img_ori, bin_img_ori!=0)
plt.subplot(221)
plt.imshow(bin_img_ori, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Original Image') plt.xticks([]) plt.yticks([])
plt.subplot(222)
plt.imshow(suz, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Border following algorithm')
plt.xticks([]) plt.yticks([])
plt.subplot(223)
plt.imshow(comb, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Combined')
plt.xticks([])
plt.yticks([])
plt.subplot(224)
plt.imshow(diff, interpolation='nearest', cmap=plt.cm.gray)
plt.title('Difference')
plt.xticks([])
plt.yticks([])
plt.show()
print sens

```

Watershed algorithm code[61, 25, 31]

```

import matplotlib.pyplot as plt
import numpy as np
import demo as d
from skimage.filter import threshold_otsu, rank
from skimage.morphology import disk, watershed
from scipy import ndimage from skimage.io import imsave
img=d.load_image("am_buildings_S&P_20_noisy.jpg").astype(np.uint8)
bin_img_ori = (d.load_image("buildings_bin_ori.jpg")>125)*1
mark_grad = rank.gradient(img, disk(3))
imsave("./output/mark_grad.jpg", mark_grad)
mark_grad_thresh = ((mark_grad < 20)*1) #(threshold_otsu(mark_grad) )
#print mark_grad_thresh.shape,

```

```

mark_grad_thresh.dtype ,
np.min(mark_grad_thresh) ,
np.max(mark_grad_thresh)
imsave("./output/mark_grad_thresh.jpg", (mark_grad_thresh > 0) * 255)
markers = ndimage.label(mark_grad_thresh)[0]
imsave("./output/markers.jpg", markers * 255 / np.max(markers))
#print markers.shape, markers.dtype, np.min(markers), np.max(markers)
local_grad = rank.gradient(img, disk(1))
imsave("./output/local_grad.jpg", local_grad)
labels = watershed(local_grad, markers) != 1
imsave("./output/labels.jpg", labels * 255 / np.max(labels))
comb = np.clip(bin_img_ori + labels, 0, 1)
sens = float(np.sum(np.logical_and(labels == bin_img_ori, bin_img_ori != 0))) \
/ float(np.sum(comb))
diff = comb - \
np.logical_and(labels == bin_img_ori, bin_img_ori != 0)
print sens
imsave("./output/result.jpg", img * labels)

```

Active contour detection code[61, 25, 31]

```

import morphsnakes
from scipy.misc
import imread
import numpy as np from matplotlib
import pyplot as plt
import demo as d
from skimage.filter import threshold_otsu
def circle_levelset(shape, center, sqradius, scalerow=1.0):
    """Build a binary function with a circle as the 0.5-levelset."""
    R, C = np.mgrid[:shape[0], :shape[1]]
    phi = sqradius - (np.sqrt(scalerow * (R - center[0])**2 + \
        (C - center[1])**2))
    u = np.float_(phi > 0)
    return u
#Load the image. img = d.load_image("buildings.jpg")
#noisy = d.load_image("original.jpg")
#noisy = d.load_image("noisy.jpg")
#noisy = d.load_image("rsb_smoothed.jpg")
#noisy = d.load_image("cm_smoothed.jpg")
noisy = d.load_image("cm_buildings_S&P_10_noisy.jpg")
#noisy = d.load_image("lulu_smoothed.jpg")
ver_cen = int(noisy.shape[0]/2)
hor_cen = int(noisy.shape[1]/2)
# g(I) gI = morphsnakes.gborders(noisy, alpha=1000, sigma=4)
#use sigma = 2

```

```

# Morphological GAC. Initialization of the level-set.
mgac = morphsnakes.MorphGAC(gI, smoothing=3, threshold=0.3, balloon=-1)
#use smoothing=2 or 3 depending on noise level
mgac.levelset = circle_levelset(noisy.shape, (ver_cen, hor_cen), \
max(ver_cen, hor_cen), scalerow=0.75)
num_iters=500 msnake = mgac
# Iterate.
for i in xrange(num_iters):
    # Evolve.
    msnake.step()
    threshold_global_otsu = threshold_otsu(img)
    bin_img = (img<threshold_global_otsu)*1
    comb = np.clip(bin_img + msnake.levelset, 0, 1)
    sens = np.sum(np.logical_and(msnake.levelset==bin_img, bin_img!=0)) \
    /np.sum(comb)
    diff = comb - np.logical_and(msnake.levelset == bin_img, bin_img!=0)
    plt.subplot(221)
    plt.imshow(bin_img, interpolation='nearest', cmap=plt.cm.gray)
    plt.title('Original Image') plt.xticks([]) plt.yticks([])
    plt.subplot(222)
    plt.imshow(msnake.levelset, interpolation='nearest', cmap=plt.cm.gray)
    plt.title('Snakes')
    plt.xticks([])
    plt.yticks([])
    plt.subplot(223)
    plt.imshow(comb, interpolation='nearest', cmap=plt.cm.gray)
    plt.title('Combined')
    plt.xticks([])
    plt.yticks([])
    plt.subplot(224)
    plt.imshow(diff, interpolation='nearest', cmap=plt.cm.gray)
    plt.title('Difference')
    plt.xticks([])
    plt.yticks([])
    plt.show()
    print sens

```