# A comparison of image processing algorithms for edge detection, corner detection and thinning

SIDDHARTH AVINASH PAREKH

This thesis is presented for the degree of Master of Engineering Science of
THE UNIVERSITY OF WESTERN AUSTRALIA

CENTRE FOR INTELLIGENT INFORMATION PROCESSING SYSTEMS (CIIPS)
SCHOOL OF ELECTRICAL, ELECTRONIC AND COMPUTER ENGINEERING
THE UNIVERITY OF WESTERN AUSTRALIA

JULY 2004

## Abstract

Image processing plays a key role in vision systems. Its function is to extract and enhance pertinent *information* from raw *data*. In robotics, processing of real-time data is constrained by limited resources. Thus, it is important to understand and analyse image processing algorithms for accuracy, speed, and quality. The theme of this thesis is an implementation and comparative study of algorithms related to various image processing techniques like edge detection, corner detection and thinning. A re-interpretation of a standard technique, non-maxima suppression for corner detectors was attempted. In addition, a thinning filter, Hall-Guo, was modified to achieve better results. Generally, real time data is corrupted with noise. This thesis also incorporates few smoothing filters that help in noise reduction. Apart from comparing and analysing algorithms for these techniques, an attempt was made to implement correlation-based optic flow.

# Contents

# Acknowledgements

I am grateful to my supervisor Thomas Bräunl for his help and guidance throughout this work. He made sure that I was always on track and did not lose sight of my final goal.

I would like to thank Peter Kovesi, Mike Alder, and Les Kitchen for helping me understand concepts in corner detectors.

The help provided by the staff at CIIPS and School of Electrical, Electronic and Computer Engineering is greatly appreciated. Special thanks to Brenda, Janice, Doris, Clive, Vickie, Alex, and Sandra.

Margaret and Krystyna at UWA Students' Services assisted in proof reading and structuring on writing. Had I not taken their advice on writing, this work would have been very different.

It was a pleasure to work in the same lab as Joshua, Alistair, Adrian, Jia Lei, Christopher, Antonio, Jochen, Norman, Stefan, Pierre, Charlène, Andreas, Frédéric, Axelle, David, Nicholas, and James.

Talking about image processing with Martin and Tzu Yen was very valuable. Thanks for all your efforts and time. Thanks to Leon for upgrading ImprovQT to version 5.1 thus greatly simplifying my work.

This work would not have been possible without the support of my friends and family. I am very grateful to my parents for encouraging me at all stages of my life. Their support is immeasurable. Parija, you have been a great partner in my life. Thanks for everything.

# 1. Introduction

*"Image processing* is the capturing and manipulation of images in order to enhance or extract information" [1]. It is an important step for many different fields including robot vision. The images captured by a robot are subjected to various kinds of processing in order to obtain relevant information. The results from the image processing act as an input for the next phase of the application. These can be like navigation and localization, where captured images are a key source of input.

Image processing techniques operate on pixels, transforming them in some manner and producing the result. A *pixel*, short for picture elements, is the basic representation of image data in an electronic format. An *image* is simply a collection of data represented in arrays of pixels. Image processing techniques are generally called *filters*. Filters work upon areas in an image called neighbourhoods. A *neighbourhood* is a set of pixels surrounding one pixel that is of interest. Generally, the neighbourhood is of size 3 x 3 in which the centre pixel is the point of interest.

Most of the operations performed on neighbourhoods are based on the concept of convolution. Mathematically, *convolution* is "an integral that expresses the amount of overlap of one function g as it is shifted over another function *f*" [2].

$$f * g = \int_{0}^{t} f(t)\ g(t - t)\ dt$$

Equation 1: A convolution function.

In filtering, an image is convolved with a pre-defined kernel. A *kernel* is a template of size m x n representing a set of weights assigned to the pixels of a neighbourhood.

*Noise* is unwanted data within the image that distorts it. Noise can occur due to reasons like errors in digitization, and imperfect capturing of images. *Smoothing* is used to suppress noise and to blur an image. Smoothing filters are essentially a *low-pass* filter, which retain low values and suppress high values. An image contains locations where there are sharp intensity fluctuations between successive pixels. Blurring is the result of reduction in these fluctuations.

Sharpening filters are used to segment an image. It is essentially a *high-pass* filter, which attenuates low values and preserves high values. Low values result from differentiation of image sections where the spread of intensity values is within a constant range. In an image, an *edge* is an abrupt change in gray level intensity values of successive pixels. High-pass filtering enhances these features in an image. High values are the result of differentiation in those sections where there is a significant change in intensity values. In image processing, differentiation is approximated by taking differences. The basic types of differences are:

- **Forward Difference**

In *forward difference*, the current (previous) pixel is subtracted from the next (current) pixel and the result is assigned to the current pixel. In Equation 2, '$\partial I_{x1}$' is the derivative of the current pixel where the current pixel is subtracted from the next pixel. '$\partial I_{x2}$' is the derivative of the current pixel where the previous pixel is subtracted from the current pixel. The result is assigned to the current pixel. Hence, the direction of difference is from right to left.

$$\partial I_{x1} = I_{x+1} - I_{x}$$
$$\partial I_{x2} = I_{x} - I_{x-1}$$

Equation 2: Forward difference method.

The equation for '$\partial I_{x2}$' corresponds to the template shown in Equation 3.

$$H_x = \begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array} \qquad H_y = \begin{array}{|c|}\hline -1 \\ \hline 1 \\ \hline \end{array}$$

(a) (b)

Equation 3: Template for forward difference for '$\partial\ I_{x2}$'.

- **Backward Difference**

In *backward difference*, the next (current) pixel is subtracted from the current (previous) pixel and the result is assigned to the current pixel. In Equation 4, '$\partial\ I_{x3}$' is the derivative of the current pixel where the next pixel is subtracted from the current pixel. In case of '$\partial\ I_{x4}$', the current pixel is subtracted from the previous pixel. The result is assigned to the current pixel. Hence, the direction of difference is from left to right.

$$\begin{aligned} \partial\ I_{x3} &= I_{x} - I_{x+1} \\ \partial\ I_{x4} &= I_{x-1} - I_{x} \end{aligned}$$

Equation 4: Backward difference method.

The equation for '$\partial\ I_{x3}$' is corresponding to template shown in Equation 5.

$$H_x = \begin{array}{|c|c|}\hline 1 & -1 \\ \hline \end{array} \qquad H_y = \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline \end{array}$$

(a) (b)

Equation 5: Template for backward difference for '$\partial\ I_{x3}$'.

- **Central Difference**

In *central difference*, the previous half-a-pixel (next pixel) is subtracted from the next half-a-pixel (previous pixel) and the result is assigned to the current pixel. In Equation 6, '$\partial I_{x5}$' is the derivative of the current pixel where the next pixel is subtracted from the current pixel. In case of '$\partial I_{x6}$', the previous pixel is subtracted from the next pixel. The result is assigned to the current pixel. Hence, the direction of difference is from left to right.

$$\partial I_{x5} = I_{x+\frac{1}{2}} - I_{x-\frac{1}{2}}$$

$$\partial I_{x6} = \frac{1}{2}\left(I_{x+1} - I_{x-1}\right)$$

Equation 6: Central difference method.

The previous two difference methods compute approximations at the previous or next half-a-pixel location, as shown in Equation 3 and Equation 5. In the central difference method, approximation is made at the previous and next half-a-pixel location. This can be expanded to '$\partial I_{x7}$' which is the difference of the next pixel and the previous pixel as shown in Equation 8.

$$\partial I_{x7} = I_{x+1} - I_{x-1}$$

Equation 7: Re-interpretation of the central difference method.

$$H_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \qquad H_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

(a) (b)

Equation 8: Template for central difference for '$\partial I_{x7}$'.

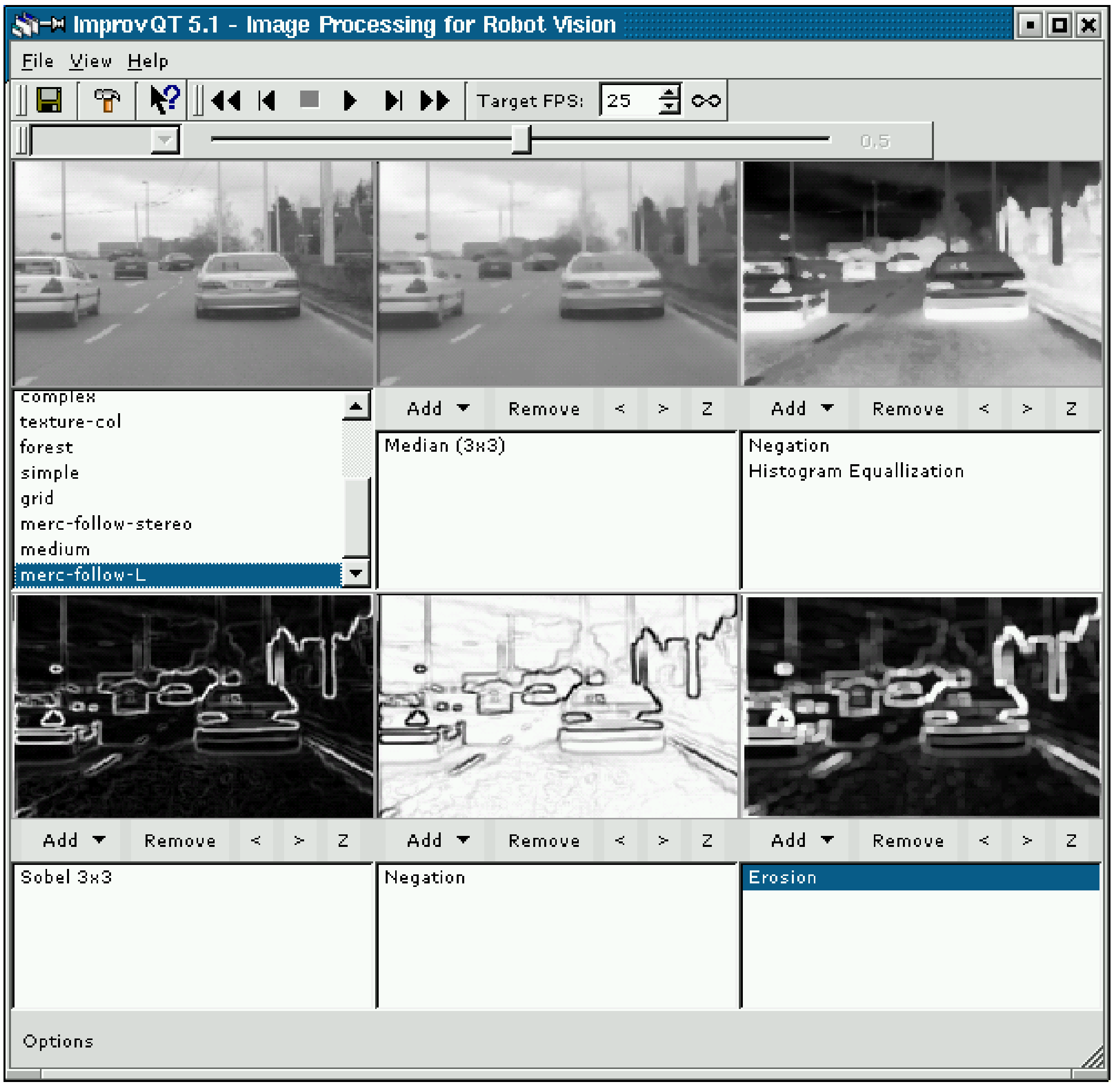
The term '$\frac{1}{2}$' in Equation 6 for '$\partial I_{x6}$' can be neglected. This is because the pixels in the real images have random intensity values. Hence, '$\partial I_{x7}$' was preferred over the discussed difference methods.

Often, when filters are applied to images, the values of pixels go out of a valid range, for example less than 0 or greater than gray-level value 255. In order to ensure that this does not happen, a factor is applied. *Normalization factor* is the process of bringing the pixel values of an image within a defined range. This concept is used in smoothing filters to ensure that on addition, values do not exceed gray-level value 255. In edge detectors, a *scale factor* may be applied to its result to ensure that the strengths of all the edges do not exceed the gray-level value 255. This simplifies distinguishing strong edges from weak edges.

In this study, the resources used were mainly computer based. A suite of image processing software called ImprovQT [3] was used for implementing various techniques and testing them.

Improv (Image Processing for Robot Vision), originally written by Dr. Thomas Bräunl at The University of Stuttgart, is currently at version 5.1. Improv is a Linux-based application. It is a tool for real-time image processing. It can work accurately on a low-resolution camera, which is required by mobile robots. It has the ability to use synthetic images or live images captured by a camera. Applying filters on images in Improv is as good as using them on robots.

The latest version of Improv comes with the ability to integrate plug-ins in an easy manner. This makes the image processing sequence completely customisable in a user-friendly manner. A screenshot of ImprovQT 5.1 is shown in Figure 1.



complex
texture-col
forest
simple
grid
merc-follow-stereo
medium
merc-follow-L

Median (3x3)

Negation
Histogram Equallization

Sobel 3x3

Negation

Erosion

Figure 1: ImprovQT 5.1 [3].

Figure 1 is an example of a simplest form of Improv. Improv can handle a number of sub-windows; default value is six.

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |

Figure 2: Order of sub-windows in ImprovQT 5.1.

There are many advantages of testing an image processing application on Improv before downloading it on robots. The main advantage is that intermediate steps involved in image processing can be systematically tested and modified. At any stage, a number of plug-ins can be added or removed from the processing to achieve better results. The output of image processing steps in Improv is the same as the output obtained on implementing the same steps on a robot.

As given in [4], "*fundamental steps in image processing are*

- *Acquisition – acquiring a digital image.*
- *Pre-processing – improving the images in ways that increase the chance for success of the other processes.*
- *Segmentation – partitioning an image into its constituent parts or objects.*
- *Representation (boundary and regional) and description (feature selection) – Representation converts the data to a form suitable for computer processing. Description extracts features for differentiating one class of object from another.*
- *Recognition and interpretation – Recognition assigns a label to an object based on information provided by its descriptors. Interpretation assigns meaning to an ensemble of recognised objects.*"

This thesis focuses on image pre-processing and segmentation. The different types of filters experimented and listed here are smoothing filters, edge detectors, corner detectors, and thinning filters.

An ideal image is one that is free from noise. Smoothing is the process of blurring and noise reduction. Types of smoothing filters implemented in this thesis are:

- Mean filter,
- Gaussian filter, and
- Median filter.

Edge detection is the process of highlighting edges in an image. Edges correspond to the outline of an object. This property of an object is useful in object detection. However, in many cases edges do not correspond to object boundaries, for example shadows, and textures. In such cases, false edges are detected. Even when edges correspond to boundaries, detection techniques fail if two pixels of different overlapping objects have similar intensity values. Types of edge detectors implemented in this thesis are:

- Laplace,
- Laplacian of Gaussian (LoG),
- Difference of Boxes (DoB),
- Sobel,
- Robert's,
- Kirsch,
- Prewitt, and
- Robinson.

Corner detectors are used to identify corners in an image. Corners are the junctions of edges. Analogous to edges false corners may be detected in regions having shadows or textures. The standard corner detectors discussed in this thesis are:

- Kitchen-Rosenfeld,
- Beaudet,
- Plessey,
- Noble, and
- Harris-Stephens.

Apart from these, an attempt was made to realize the concepts of non-maxima suppression for corners detectors. In addition, corner detectors based on derivatives of corner response were also implemented.

Thinning filters are used to thin edges and lines in an image. This is because we may only be interested in the presence of lines, not in their thickness. Thinning filters try to reduce the width of edges or lines as much as possible, ideally to one pixel. The filters implemented in this thesis are:

- Stefanelli-Rosenfeld,
- Lü-Wang, and
- Hall-Guo.

The standard Stefanelli-Rosenfeld algorithm [5] was modified to speed up calculations as explained in [6]. Similarly, the Hall-Guo algorithm [7, 8] was adapted to obtain much better and faster results.

Finally, Optic flow was also implemented. Optic flow is a vector field that shows the direction and magnitude of intensity changes of a pixel in an image. In this thesis, correlation-based optic flow was implemented. It was utilized to find similarities between two successive images of a sequence. The concepts of sum of the squared differences (SSD) [9-11] and sum of the absolute differences (SAD) [10] were applied in obtaining optic flow.

All these filters were implemented as plug-ins in ImprovQT [3]. A number of different combinations of these filters were experimented on test images. This was done in order to achieve better results and speed up computation. The images considered for these tests were real and synthetic gray-scale. These algorithms and techniques were compared, and tested on various images.

# 2. Literature review

A number of sources regarding image processing techniques and algorithms were reviewed and studied. The filters implemented in this thesis were motivated by this review.

Deriche and Giraudon [12] proposed a new scale-space based approach that combines Laplacian and Beaudet's measure for corner and vertex detection.
According to the authors, there are two groups of corner detectors:

- The first group extracts the edges as a chain code and then searches for points having maximum curvature. Thus, it involves an extra calculation for detecting edges.
- The second group is based on approaches that operate directly on a gray-level image. They either find "interest points" or work on the measurements of gradients and curvatures of the surface.

Their work analyses several corner detectors based on the second group. This work gave a good starting point for implementing the Beaudet [13], Kitchen-Rosenfeld [14], Plessey [15], Noble [16], and Harris-Stephens [17] corner detectors. This work highlights similarities and differences between Kitchen-Rosenfeld and Zuniga-Haralick, and Plessey, Noble and Harris-Stephens corner detectors. As shown in the paper, Zuniga-Haralick is based on the facet model while Kitchen-Rosenfeld is an operator that directly computes whether a pixel is a corner. Zuniga-Haralick differs from Kitchen-Rosenfeld with an expression $\sqrt{I_x^2 + I_y^2}$ in the denominator (Equation 39). Plessey (Equation 43) was proposed by Harris; and Noble gave a theoretical formulation for this detector using differential geometry (Equation 45). Harris and Stephens made a slight modification to the original Plessey (Equation 47). The authors state that none of these corner detectors has good corner localization. Their approach attempts to correct this.

“Interest points” are better explained by Sonka *et al* [18] as an investigation among a much smaller number of points of an image. These points are expected to have some typical local property. For example, if square objects are present in an image then corners are very good interest points. The web-site mentions corner operators Moravec (based on interest points), Zuniga-Haralick (based on the facet model as explained in [12]), and Kitchen-Rosenfeld. Deriche and Giraudon [12] give a different representation of the Zuniga-Haralick corner operator. Kovesi helped in further understanding Harris corner detector [19, 20]. The papers by Cooper *et al* [21, 22] was also a source of understanding of Kitchen-Rosenfeld corner detector and few other concepts for corners.

A number of edge detectors and smoothing filters were listed by Sonka *et al* [18]. These were Laplace, Laplacian of Gaussian (LoG) Sobel, Robert’s, Prewitt, Kirsch, and Robinson. Their description of the Laplacian of Gaussian (LoG) edge detector and Gaussian mean filter was useful in their implementation. Bräunl [6] gives a good explanation and skeleton code for the Laplace and Sobel edge detectors along with their edge directions. Bernd [23] gave a good start towards understanding the Difference of Boxes (DoB) edge detector.

The discussions on smoothing filters like Mean, Gaussian, and Median in Sonka *et al* [18], Gonzalez *et al* [4], and Jain *et al* [24] assisted in the implementation of these filters.

In fields like fingerprint and pattern matching, the shape of the lines or objects is more important than its thickness. When the thickness changes uniformly throughout the image, these applications can produce results by neglecting the thickness, as suggested by Alt [25]. In general, the input for these applications does not exhibit a uniform change in thickness.

To perform matching without thickness affecting the process, thinning was proposed by Stefanelli and Rosenfeld [5]. They stressed on the *presence* and *shape* of line-based objects, rather than their *thickness*. One method to achieve thinning is to find the “medial line” of the objects and to delete pixels not on the medial line. They proposed two algorithms for identifying the medial-line. In the first method, at each iteration, all contour points (points lying on the outline of the object) except the final points (points lying on the medial line) of the object are deleted. This method has a disadvantage that it can yield non-connected or even no medial line for connected figures. To overcome this, the second method has four sub-iterations; at each sub-iteration, only a part of the contour is removed.

Lü-Wang [26] proposed an improved version of the algorithm introduced by Zhang and Suen [27]. They pointed out the disadvantages in [27] and overcame these by preserving necessary and essential lines which should not be deleted. The process of thinning was divided into two sub-iterations with conditions acting on each of the sub-iteration. The conditions for each sub-iteration were different to ensure that in the first sub-iteration only the southeast pixels are deleted. The second sub-iteration concentrated on deleting pixels on the northwest boundary. They pointed out difference in the approach from [27], by swapping the order of the sub-iterations. This method overcomes the problem of preserving important structures, which were part or whole of a pattern. This was accomplished by manipulating the conditions proposed by Zhang-Suen [27].

Guo and Hall proposed two parallel thinning algorithms [8]. They proposed a two sub-iteration process for deleting unwanted pixels. Alternately, these sub-iterations concentrated on deleting northeast and southwest pixels, and applying a thinning operator to one of two subfields. Their work did not concentrate on recovering the original image unlike a few other approaches that focussed on image recovery as proposed by Arcelli *et al* [28]. This is because they stated that image recovery after thinning is not always necessary. They pointed out that the fully parallel thinning algorithms, which are restricted to 3 x 3 operators, have difficulty in preserving the connectivity. Hence, few authors [5, 8, 27] partially serialized their algorithms by dividing an iteration into several distinct sub-iterations or by partitioning the image into distinct subfields [29]. Guo and Hall pointed out that by defining distinct subfields these algorithms applied thinning operators to different parts of an image. In addition, they pointed out that on using 3 x 3 local operators, algorithms defining two sub-iterations achieved optimal results. They pointed out that the thinning filter proposed by Lü and Wang [26], and Zhang and Suen [27] outperforms the one proposed by Stefanelli and Rosenfeld [5]. The original algorithm [27] and its improved version [26] was modified by Guo and Hall to overcome the problems of maintaining connectivity. The modified algorithm had a variable that was useful in endpoint detection while the second algorithm proposed by them had a scheme of dividing the image in a checkerboard fashion as described by Holt *et al* [30]. The medial curve of thinness produced by the two algorithms was comparable but the second algorithm was faster. This thesis concentrates on studying the second algorithm.

In his PhD thesis, Camus [10] presented optical flow algorithms for real-time robots. He pointed out limitations of robotic systems that affect optical flow calculations. These are like restricted sensing capacity, limited computational power, and restricted mobility. This has resulted in limited implementation of optic flow for practical applications. The success of an optical flow algorithm is measured by how well it fulfils the three basic requirements of robotic vision; robustness, speed, and accuracy. According to Camus, accuracy can be optimised once the first two requirements are met.

In his thesis, he has discussed the limitations and implementation of gradient-based, velocity-tuned filter based, and correlation-based techniques for computing optical flow. The algorithms based on correlation-based optical flow are shown to be robust in practice but are practically infeasible due to the high computational costs. He tried tackling the problems of correlation-based approach by devising a space-time trade off to this algorithm. His modification reduces the running time of the algorithm from quadratic-time to linear-time. His work gave a good start for understanding correlation-based optic flow.

Correlating successive frames helps to analyse the motion of an object. However due to the aperture problem it may not be possible to obtain the correct optical flow vector for all the corresponding image frames. This problem cannot be solved completely. However, by correlating the pixel's neighbourhood in successive frames and then finding the corresponding match of a pixel helps to minimise the aperture problem. To find the corresponding match of a pixel a maximum possible displacement is presumed. Camus [10] assumed this value to be seven for his experiments while Wei-Härle [31] assumed the value to be five. The match strength was calculated as the sum of the match values between each pixel in the displaced patch in the first image and the corresponding pixel in the actual patch in the second image. Camus implemented two techniques to compute match strength. These are SAD, which takes the absolute differences between the two pixels' intensity values and SSD, which takes the squared differences of their respective intensity values. The lowest value of the match strength was considered the best match. Moravec used a variance of normalized cross-correlation (NCC) to calculate match measure for stereo matching. Camus stated that NCC and SAD were insensitive to contrast. However, he observed that NCC did not produce good results. In addition to these techniques, Singh [11] listed direct cross correlation and mean normalized cross correlation.

Temizer [32] calculated optic flow by first applying a Gaussian smoothing filter to blur (smooth) the image. A Laplacian filter was then applied to obtain the edges from the images. After applying these filters, the displacements are calculated by correlation as explained by Nishihara [33]. Temizer evaluated the best match by comparing the number of matching pixels in two patches with some (percentage) threshold. If the number is greater than the threshold, then the two patches are considered to match.

A practical implementation of correlation-based optical flow was used for obstacle avoidance as outlined by Grünewald *et al* [34] and for object detection as discussed by Batavia *et al* [35].

To implement edge and corner detectors, understanding the concept of applying derivatives was necessary. Sonka *et al* [18], Gonzalez *et al* [4] and Jain *et al* [24] were helpful in clearing these concepts. Kreyszig [36] and Mathworld [37] helped in clarifying general mathematical concepts. Mathworld [37] and Gonzalez *et al* [4] helped in the understanding of convolution and correlation.

There was a constant source of inspiration from [38], [39], and [40].

## 3. Smoothing filters

*Smoothing* filters are used to blur an image and reduce noise. There are different types of such filters that can be applied for different kinds of problems. The filters discussed here are the *Mean*, *Gaussian*, and *Median* filters. The Mean and Gaussian are linear filters whereas Median is a non-linear filter. In *linear* filters, the output pixel value is calculated using the weighted sum of the input pixels. A *non-linear* filter does not calculate the weighted sum of pixels in the neighbourhood. It assigns a value to the output pixel, which is directly based on the values of the pixels in the neighbourhood.

### 3.1. Mean filter

The Mean filter calculates the mean (arithmetic) value in the neighbourhood of a pixel under consideration and assigns it to the pixel. If the dimensions of the template used are odd (e.g. 3 x 3), then the pixel under consideration is the centre pixel. If not, then a decision has to be made about selecting which pixel to consider. When selecting this pixel for the first neighbourhood, consistency has to be followed in maintaining the relative position of the pixel in successive neighbourhoods of the image.

Equation 9 shows the mean function that is applied to every pixel of the image. Here, `f (i,j)` is the output image at pixel `(i,j)` and `g (i,j)` is the input image. `k` is the index in the template and `n` is the number of the elements of the neighbourhood. $\frac{1}{n}$ is the normalization factor that makes sure that the output image values are within the range.

$$f(i,j) = \frac{1}{n}\sum_{k=1}^{n} g_k(i,j)$$

Equation 9: Mean function [24].

An example of the mean 3 x 3 filter is shown in Equation 10.

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

$$3 \times 3$$

Equation 10: Mean filter 3 x 3.

In general, a square template looks like Equation 11.

$$\frac{1}{K \times K}\begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline 1 & 1 & 1 & \cdots & 1 \\ \hline \end{array}$$

$$K \times K$$

Equation 11: Mean filter K x K.

A rectangular template looks like Equation 12.

$$\frac{1}{M \times N} \quad \begin{array}{c|c|c|c|c|c|} & 0 & 1 & 2 & ... & N \\ \hline 0 & 1 & 1 & 1 & \vdots & 1 \\ \hline 1 & 1 & 1 & 1 & \vdots & 1 \\ \hline : & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline M & 1 & 1 & 1 & \vdots & 1 \\ \hline \end{array}_{M \times N}$$

Equation 12: Mean filter M x N.

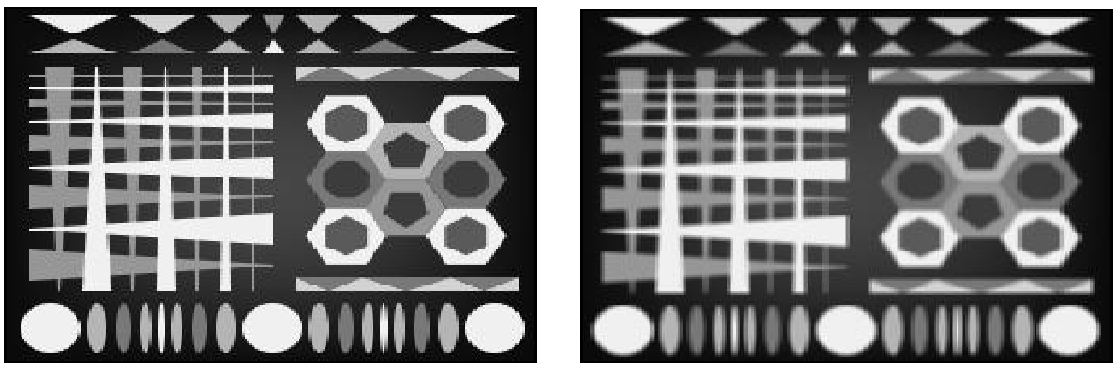
In Mean filtering, the larger the template size, the more the blurring in the result image and more time is consumed in calculations. This can be explained by assuming the filter’s template size to be 1 x 1. Upon filtering, the value of the centre pixel is replaced by the mean value, which is its own value. Hence, in the end, we get the same image without any smoothing and a very fast process. On inc reasing the size of the template, say to 3 x 3, the mean of the centre pixel is no longer its own value. It is now calculated from the expanded neighbourhood, that is, the value of the other pixels in the neighbourhood starts affecting the mean calculation. In addition, the process takes more time. For this reason, the larger the template size, the more the blurring and the better the smoothing. Generally, Mean filters based on the 3 x 3 template have shown the best performance in speed and blurring effect.

In certain templates, the centre pixel is given more stress, i.e. its value is considered higher than the neighbourhood pixels. Such a template may look like Equation 13.

$$\frac{1}{10}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 2 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

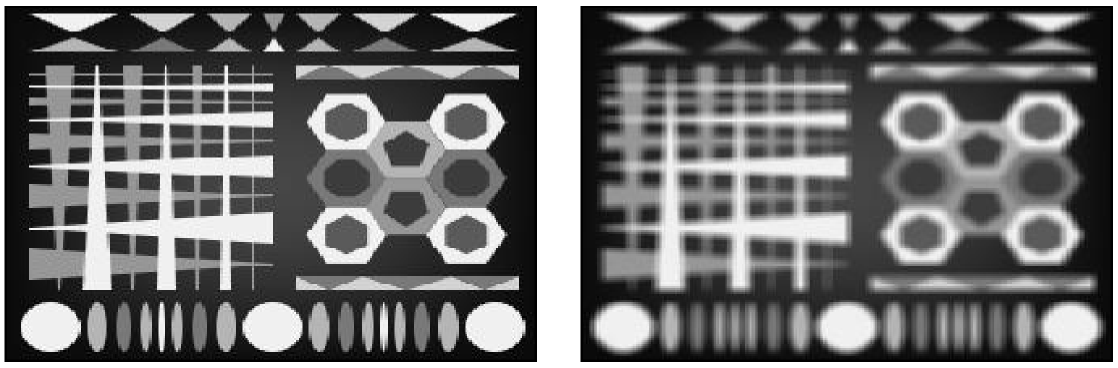
Equation 13: Mean filter with variable weights [24].

This helps bring about a better approximation of noise in the neighbourhood. This may often be required at further image processing stages.



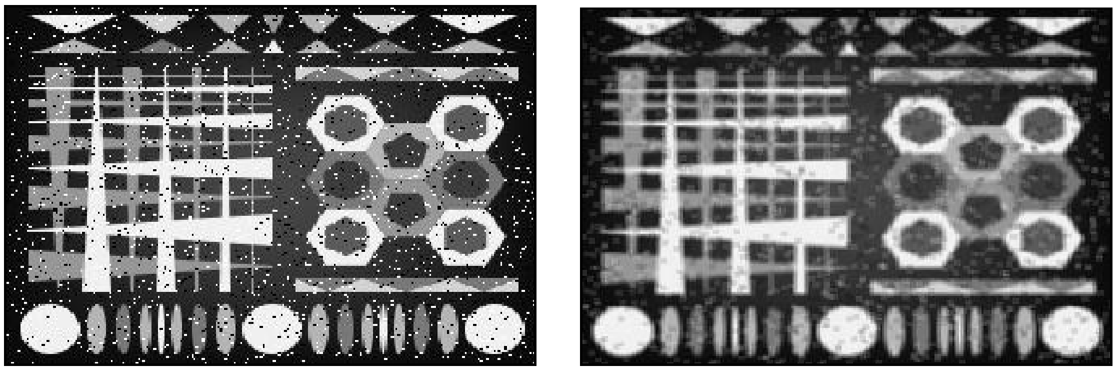
(a) (b)

Figure 3: Mean 3 x 3 filter (Equation 10): (a) Original image, (b) Smoothed image.



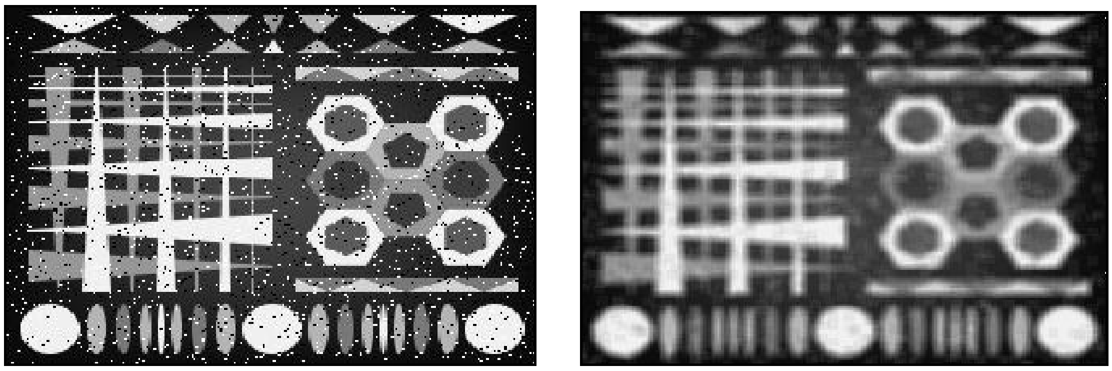
(a) (b)

Figure 4: Mean 5 x 5 filter (Equation 11, K = 5): (a) Original image, (b) Smoothed image.



(a) (b)

Figure 5: The effect of Salt and Pepper noise on an image: (a) Original image corrupted with noise, (b) Smoothed image with Mean 3 x 3 filter (Equation 10).



(a) (b)

Figure 6: The effect of Salt and Pepper noise on an image: (a) Original image corrupted with noise, (b) Smoothed image with Mean 5 x 5 filter (Equation 11, K = 5).

## 3.2. Gaussian filter

The Gaussian filter is similar to the Mean filter with the difference of the weights assigned to every pixel in the kernel. The standard one-dimensional Gaussian filter is defined as Equation 14.

$$G(x) = \frac{1}{\sqrt{2p}\, s} e^{\frac{-x^2}{2s^2}}$$

Equation 14: Gaussian function - one-dimensional [18, 41].

The *Gaussian function* is used to form the kernel, which is convolved with every pixel in the image. '$s$ ' is the standard deviation, a variable factor which indicates the width of the *Gaussian curve* over a set of pixels, generally '3 $s$ '. The window size (size of the kernel) is calculated by Equation 15.

$$window\ size = 1 + 2 * ceil(3 * s)$$

Equation 15: Calculation of "window size" in Gaussian filtering [42].

The Gaussian curve response for one-dimension with mean 0 and with different sets of '$s$ ' is shown in Figure 7, Figure 8, and Figure 9.

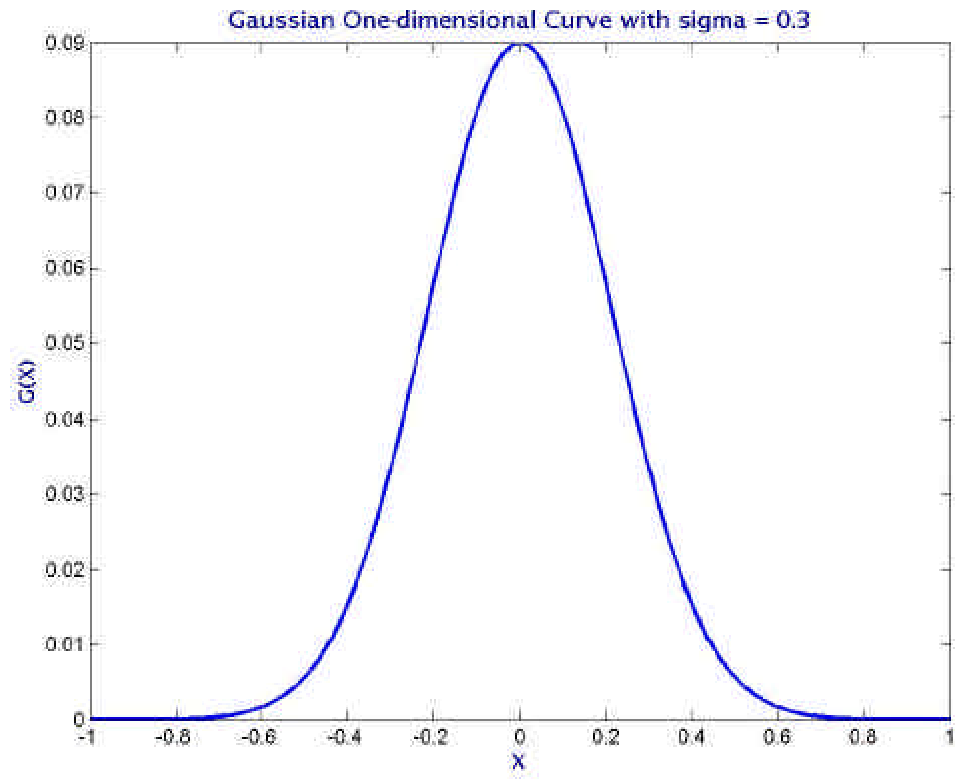


Figure 7: Gaussian One-dimensional Curve with mean 0 and standard deviation ( $s$ ) = 0.3 (Equation 14).

In the above Figure 7, the curve is steep. This is due to the value of standard deviation '$s$ '.

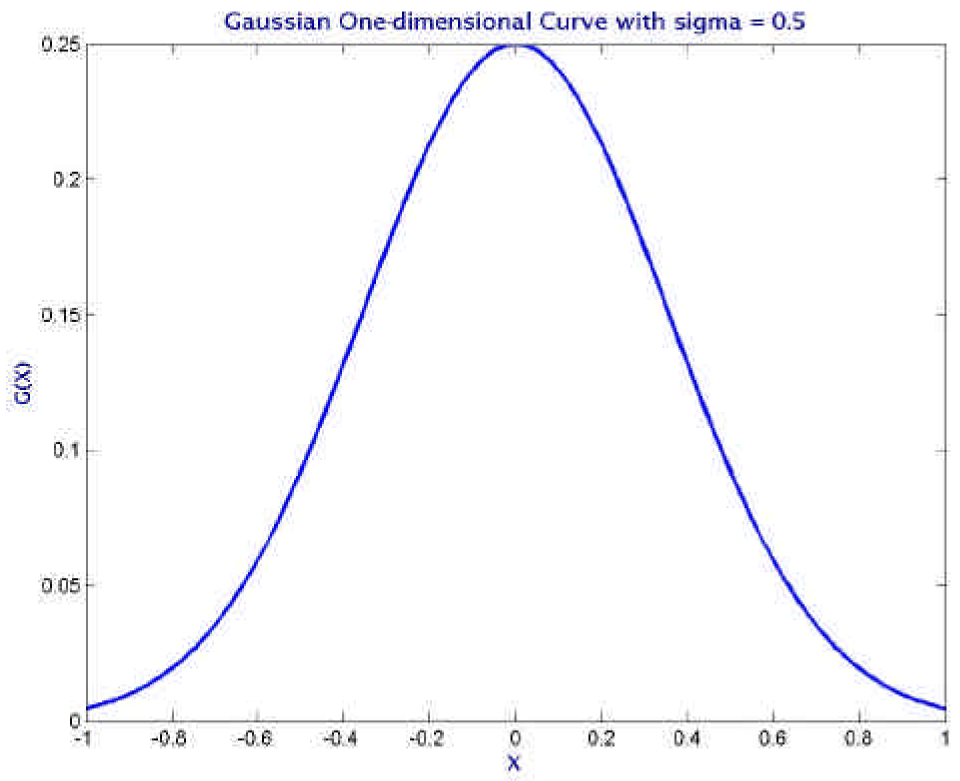


Figure 8: Gaussian One-dimensional Curve with mean 0 and standard deviation ( s ) = 0.5 (Equation 14).

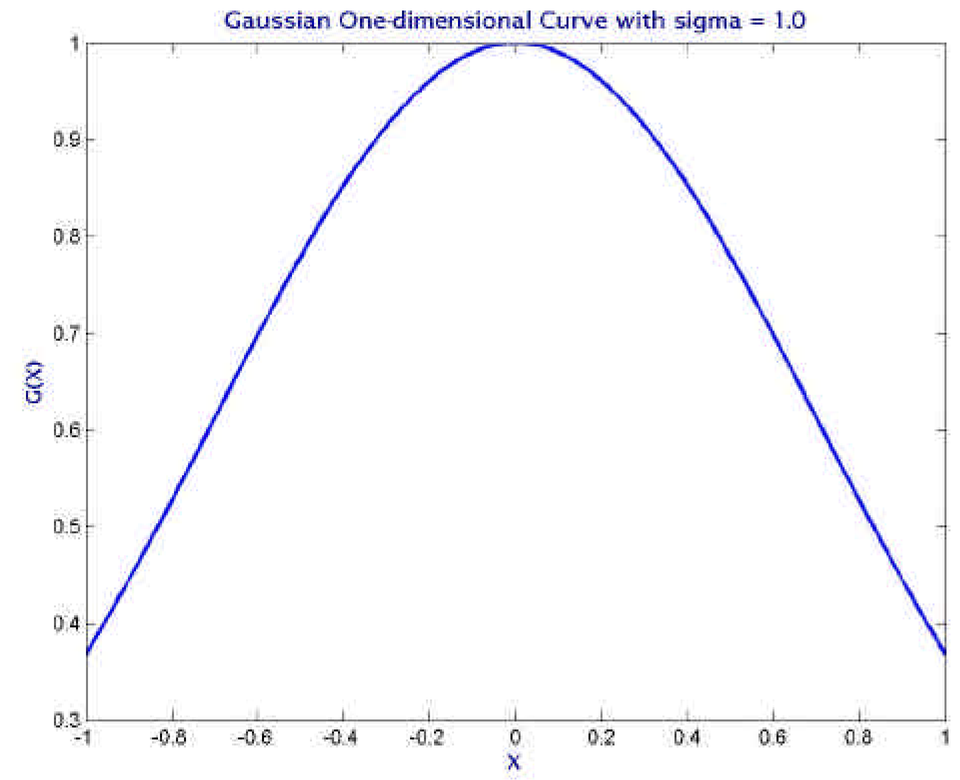


Figure 9: Gaussian One-dimensional Curve with mean 0 and standard deviation ( s ) = 1.0 (Equation 14).

As seen from Figure 7, Figure 8, and Figure 9, the higher the value of sigma ‘$s$’, the shallower the gradient of the curve. For this reason, the Gaussian curve in Figure 9 ($s$ = 1.0) is smoother than the curve in Figure 8 ($s$ = 0.5) which is smoother than the Figure 7 ($s$ = 0.3).

The one-dimensional Gaussian filters along the x and the y directions acting together on the image results in one two-dimensional Gaussian filter. The relation is shown in Equation 16.

$$G(x, y) = G(x)\, G(y)$$

Equation 16: Two one-dimensional Gaussian filter resulting in one two-dimensional Gaussian filter.

Using the same value of ‘$s$’, applying the two-dimensional Gaussian filter has the same effect as applying the one-dimensional Gaussian filter twice, once each in the x and y directions. A two-dimensional Gaussian function can be written as Equation 17.

$$G(x, y) = \frac{1}{2ps^2}\, e^{\frac{-(x^2 + y^2)}{2s^2}}$$

Equation 17: Gaussian function - two-dimensional [18, 41].

The one-dimensional Gaussian function along the x and the y directions can be applied in either way, that is, x after y or y after x, to result in two-dimensional Gaussian function. Both are shown in Figure 10.

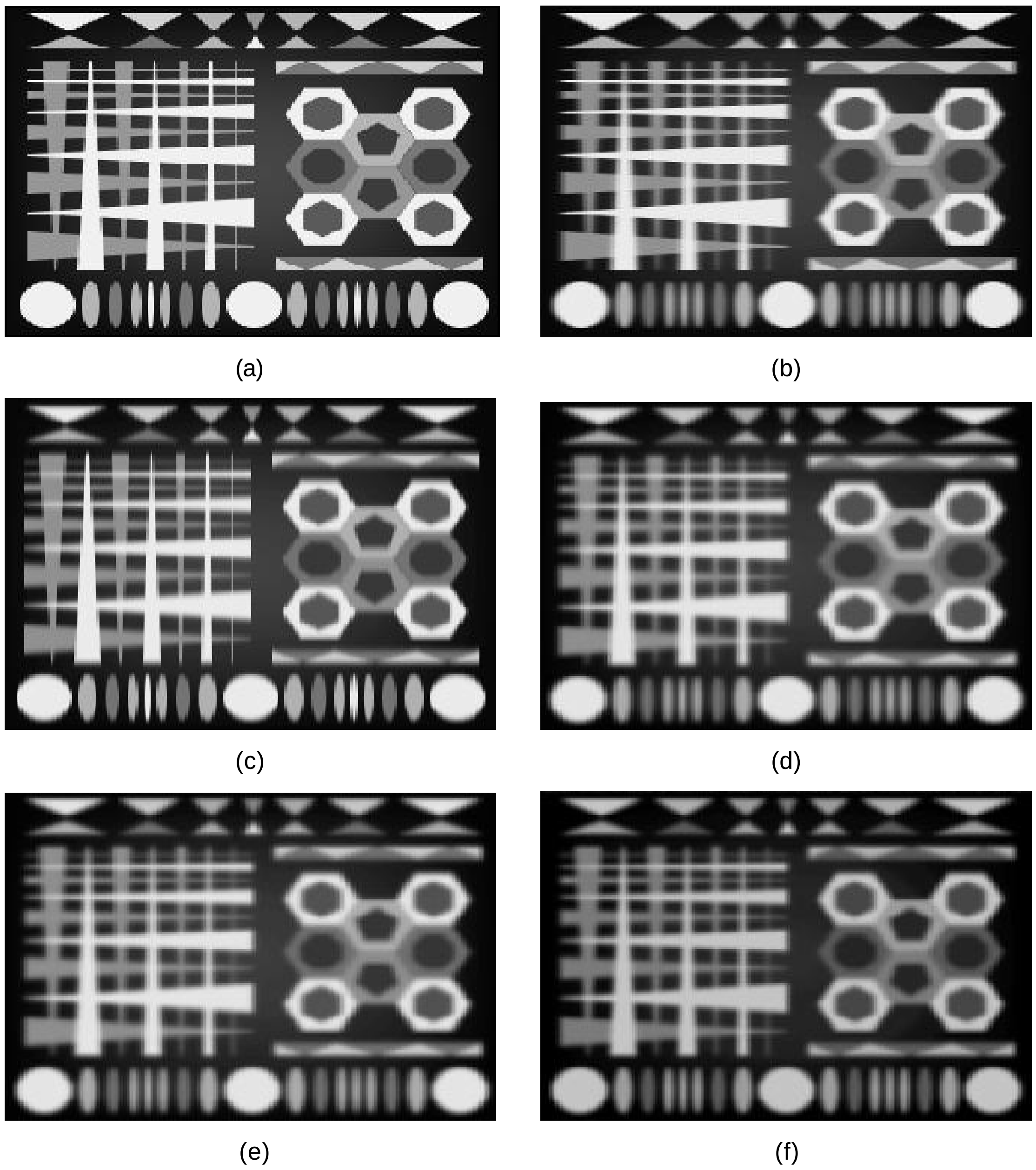


Figure 10: The result of the Gaussian function applied to the image: (a) Original image, (b) Gaussian x on original image (Equation 14), (c) Gaussian y on original image (Equation 14), (d) Gaussian x after y on original image, (e) Gaussian y after x on original image, (f) Gaussian two-dimensional function on original image (Equation 17).

The value of standard deviation ' $s$ ' used in all these results is 1.5 (Figure 7).

In Figure 10, (d), (e), and (f) exhibit the same result on applying the filter to (a). Figure 10 (b) and (c) are the results of applying Gaussian function along the x and the y directions respectively. Figure 10 (d) and (e) are the results of applying Gaussian function x after y and y after x respectively. Figure 10 (f) is the result of applying two-dimensional

Gaussian function on the original image (a). In all these Gaussian functions, the value of $s$ was taken 1.5.

Similar to the one-dimensional Gaussian function, the two-dimensional function is used to form a kernel, which is then convolved with every pixel in the image. The window size is calculated as given by Equation 15. Unlike the one-dimensional Gaussian, the two-dimensional convolution kernel is a square template whose size is given by `windowsize * windowsize`.

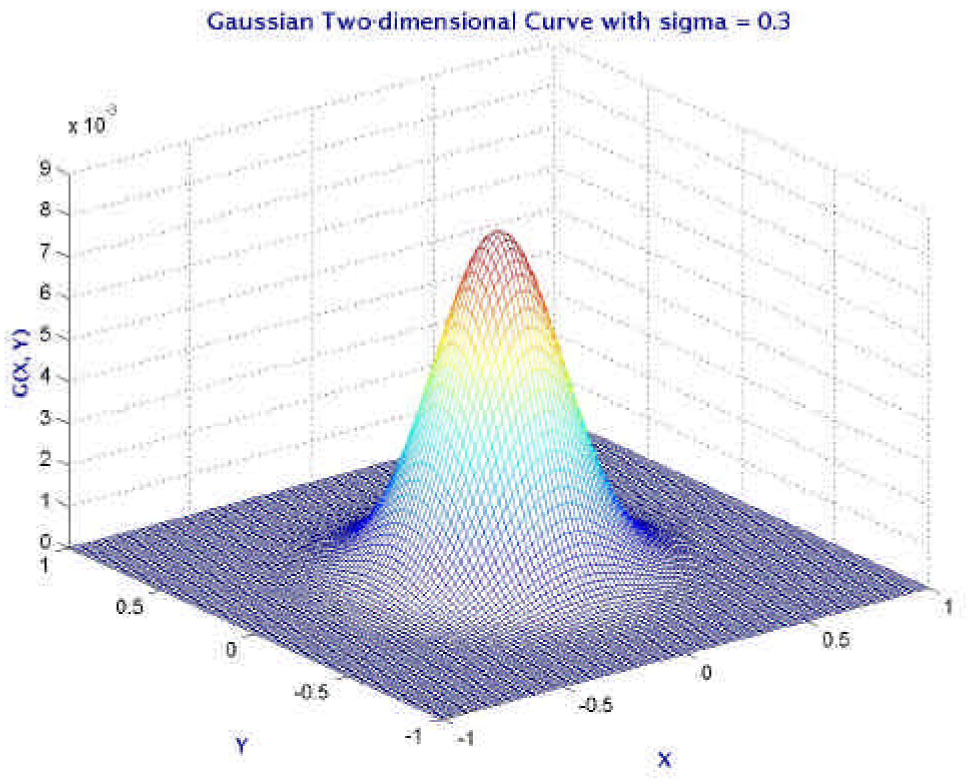


Figure 11: Gaussian Two-dimensional Curve with mean 0 and $s$ = 0.3 (Equation 17).

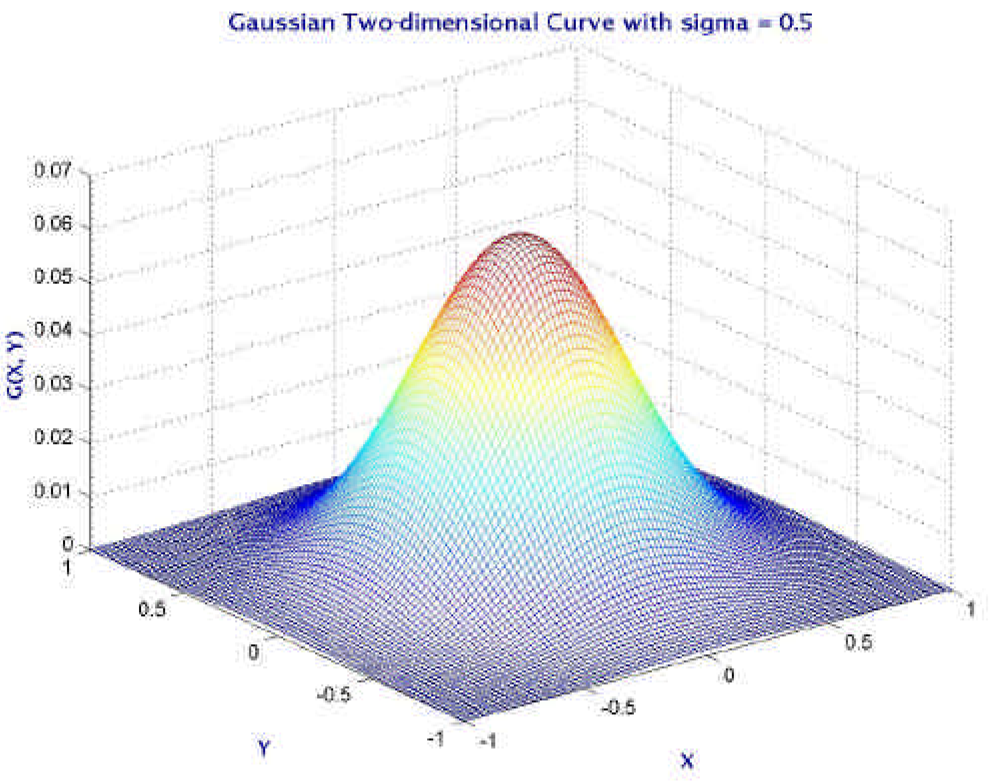


Figure 12: Gaussian Two-dimensional Curve with mean 0 and $s$ = 0.5 (Equation 17).

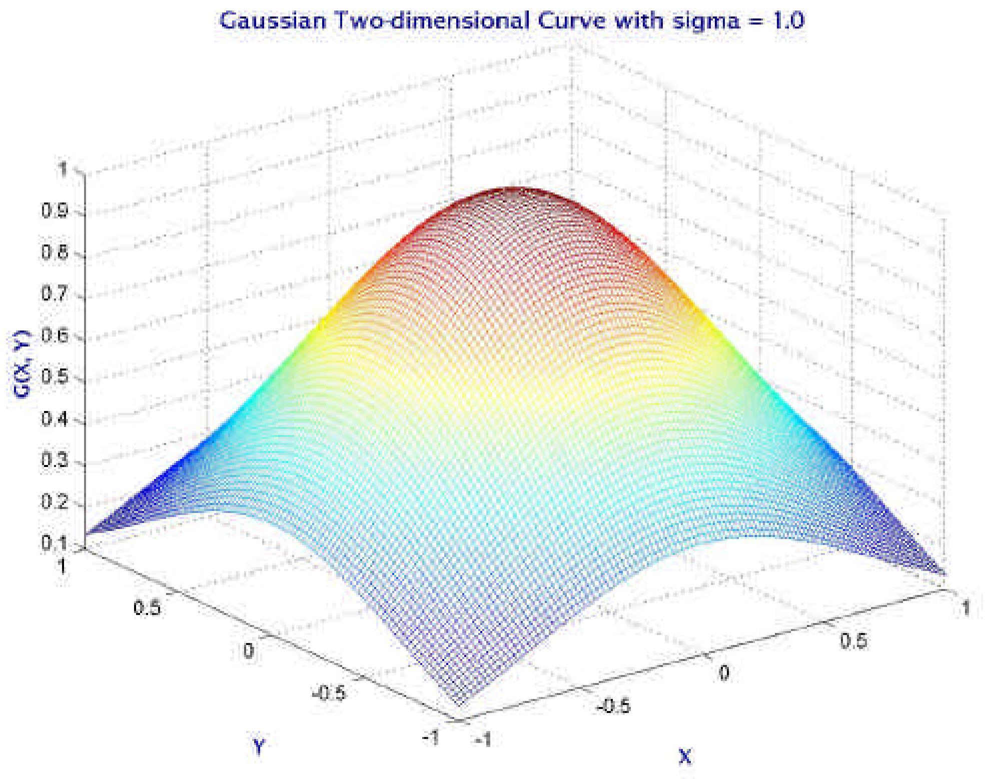


Figure 13: Gaussian Two-dimensional Curve with mean 0 and $s$ = 1.0 (Equation 17).

The value of '$s$ ' affects the slope of the curve in Figure 11, 16, and 17, just as in the case of Gaussian one-dimensional filter. The higher the value of '$s$ ', the smoother the resulting image.

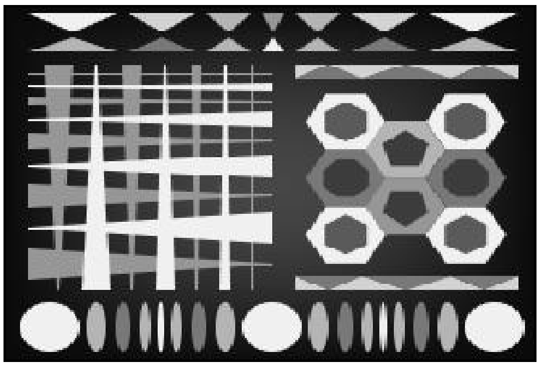
As outlined by Fisher *et al* [41], a two-dimensional Gaussian function is a discrete approximation with the value $s$ = 1.4 as shown in Equation 18.

$$\frac{1}{155}\begin{array}{|c|c|c|c|c|}\hline 2 & 4 & 5 & 4 & 2 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 5 & 12 & 15 & 12 & 5 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 2 & 4 & 5 & 4 & 2 \\ \hline \end{array}_{\;5 \times 5}$$

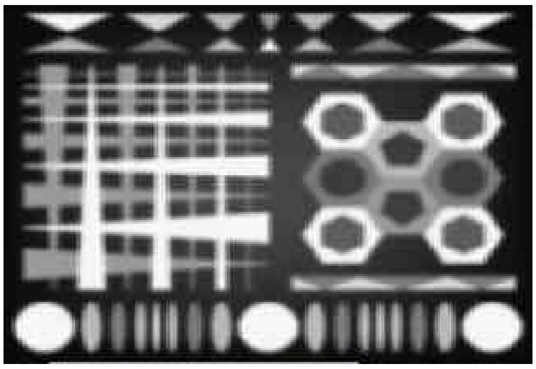
Equation 18: Discrete approximation to Gaussian function with $s$ = 1.4 [41].

The result of the discrete approximation of Gaussian function with $s$ = 1.4 along with $s$ = 1.5 is shown in Figure 14. Figure 14 (a) shows the original image. Figure 14 (b) and (c) are the two-dimensional Gaussian function approximated at $s$ = 1.4 and $s$ = 1.5 respectively.

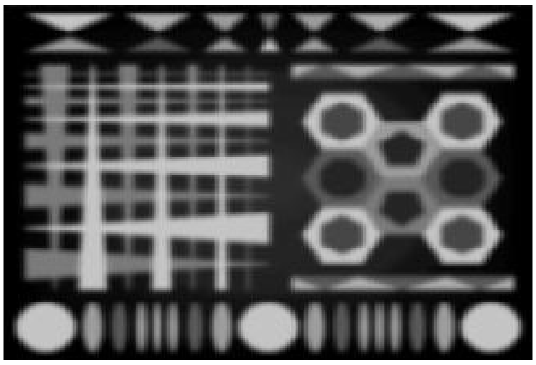
It can be clearly seen that the image in Figure 14 (b) is less blurred than Figure 14 (c). It was observed that it took longer to produce the result in Figure 14 (c) compared to Figure 14 (b). This is due to the kernel size; where latter uses a 5 x 5 kernel size to smooth the image, the former takes a 11 x 11 kernel size to smooth the image.



(a)



(b)



(c)

Figure 14: Two-dimensional Gaussian functions on an image: (a) Original image, (b) Gaussian function at $s$ = 1.4 (5 x 5 template from Equation 17), (c) Gaussian function with $s$ = 1.5 (11 x 11 template using Equation 15 and Equation 18).

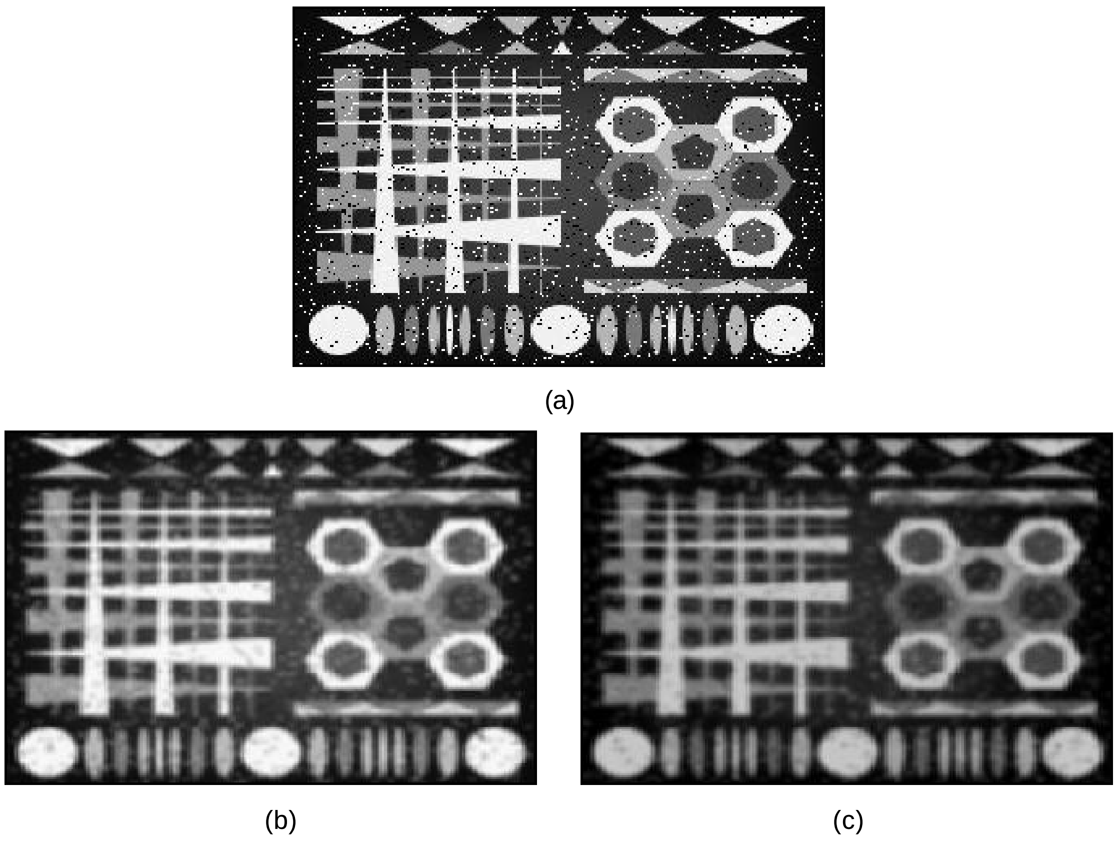


Figure 15: The effect on Salt and Pepper noise on an image: (a) Original image corrupted with noise, (b) Smoothed image with Gaussian function at $s$ = 1.4 (5 x 5 template from Equation 17), (c) Smoothed image with Gaussian function with $s$ = 1.5 (11 x 11 template using Equation 15 and Equation 18).

## 3.3. Median filter

Not only does the Median filter reduce the noise in the image, it also reduces the contrast of the edges. Compared to the Mean and Gaussian filters, the Median filter has less blurring effect at edges. The Median filter calculates the median of the neighbourhood for the pixel under consideration and assigns this value to the same position in the output image. Generally, the Median filter uses templates of size 3 x 3. The process for each pixel under consideration is as follows:

- Consider the neighbourhood values.
- Sort them in ascending or descending order.
- Estimate the median using the following criteria:

  - o If there are an odd number of pixels in the template, the median is the centre of the sorted values.
  - o If there is an even number of pixels in the template, the median is the average of the two centre pixels of the sorted values.
- Assign this median value to the pixel in the same location in the output image.

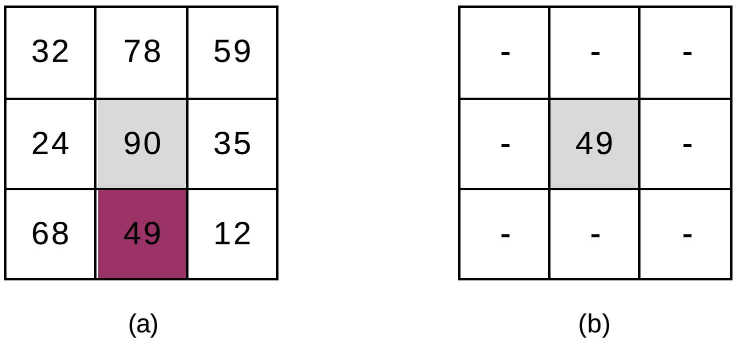


Figure 16: Median filter: (a) The pixels values in the input image, (b) The result of applying a median filter to the pixel under consideration in the output image. This result value (49) replaces the old value (90).

Suppose we take the centre pixel (value 90) in Figure 16 as the pixel under consideration.

Median (at the centre pixel) = median (32, 78, 59, 24, 90, 35, 68, 49, 12)

First, the values are sorted in ascending order.

Sort (32, 78, 59, 24, 90, 35, 68, 49, 12) = 12, 24, 32, 35, 49, 59, 68, 78, 90.

On sorting, we get value 49 as the median value for the neighbourhood. The initial value of the pixel under consideration, 90, is replaced with the median value, 49.

The Median filter is not optimal in speed, as it has to sort the neighbourhood values for every pixel before the median can be decided.

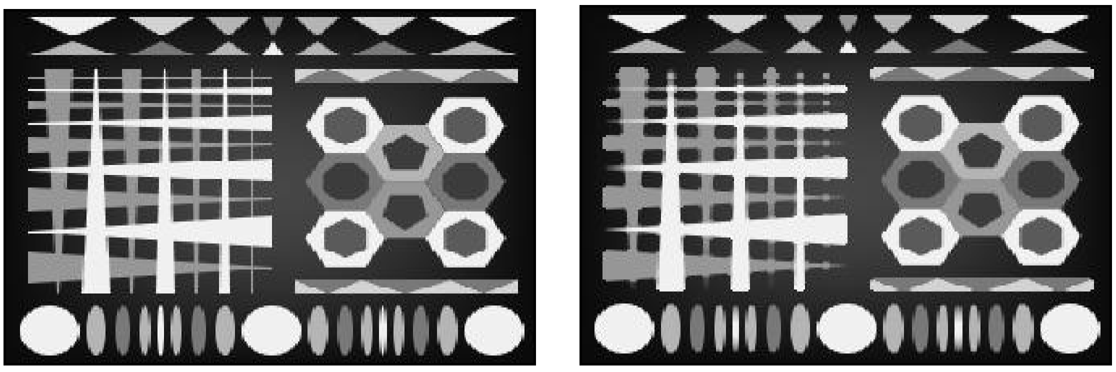

(a) (b)

Figure 17: Median filter: (a) Original image, (b) Image after applying Median filter.

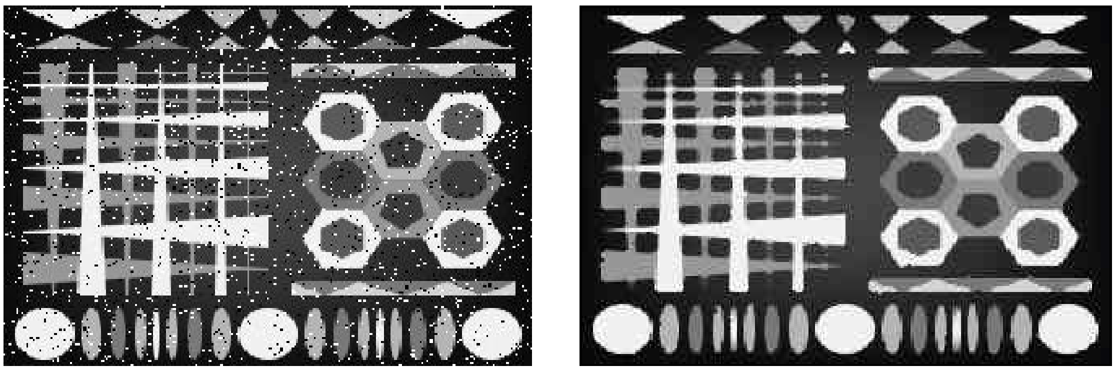

(a) (b)

Figure 18: Median filter on salt and pepper noise: (a) Original image, (b) Image after applying Median filter.

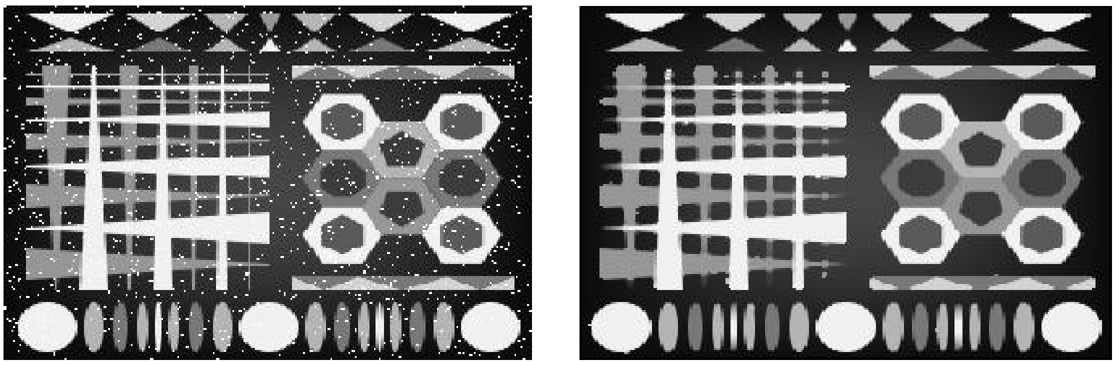

(a) (b)

Figure 19: Median filter on impulse noise: (a) Original image, (b) Image after applying Median filter.

As seen in Figure 17, Median filtering preserves the edges but reduces the sharpness formed by a corner. At the junctions of edges, Median filter shows a curved shape and that corner cannot be clearly detected by a corner detector. This is the reason why a median filter is not preferred in corner detection.

## 3.4. Comparison of smoothing filters

The Mean and Gaussian filters blurred the image to some extent to remove the inconsistency in the fluctuations of neighbouring pixel values. However, they were not able to remove the salt and pepper and impulse noises. Compared to the Mean filter, Gaussian filter had a better blurring effect due to the different weights assigned to different positions of pixels. While Median filter did not blur the image as much, it was able to suppress salt and pepper, and impulse noises. In the Median filter, it was observed that the intersection of edges were no longer as sharp as in the original image. Comparatively, it preserved the *quality* of edges. This is because, the median value is not calculated using the weighted sum of pixels like the Mean and Gaussian filters but is taken as one of neighbourhood pixel values. Table 1 compares the speed of different smoothing filters applied on the test image (Figure 3 (a)).

The following are some of the key parameters that were selected for measurement of performance of smoothing filters.

- Original speed: This is the maximum speed, measured in fps (frames per second), possible with no smoothing filters applied to the image. This value remains same for every smoothing filter as the same image (Figure 3 (a)) was used for testing all filters. For the image used in these tests, the original speed was at a maximum of 222 fps.

- Final speed: This is the maximum speed with the particular smoothing filter applied on the image.

- Parameters: For an optimal solution, a smoothing filter may depend on parameter(s), which need to be tuned at run-time.

- Type: The smoothing filters were classified into either of the two types, linear and non-linear.

| | Final Speed | Parameters | Type |
|---|---|---|---|
| Mean 3 x 3 | 53.0 | - | Linear |
| Mean 5 x 5 | 46.0 | - | Linear |
| Gaussian Two-Dimensional | 2.8 | $s$ = 1.5[1] | Linear |
| Gaussian 5 x 5 | 26.9 | $s$ = 1.4[2] | Linear |
| Median | 27.6 | - | Non-Linear |

Table 1: Comparison of different smoothing filters.

1 - A kernel of size 11 x 11 was created using Equation 15 with '$s$' = 1.5.

2 - Discrete Approximation with '$s$' =1.4 for defining kernel of size 5 x 5 (Equation 18).

# 4. Edge detectors

*Edge detection*, the ability to determine the edge of an object [43], is a primary step in many image enhancement procedures. In an image, an *edge* is an abrupt change in gray level intensity values of successive pixels. Hence, when there is a high difference between two neighbouring pixels, a possible edge is detected. An edge detector is essentially a high-pass filter. The intensity of the pixels at the borders of a shadow also transit from a low to a high value. Due to this, any edge detection technique detects this outline of shadows as edges. This results in detection of false edges. Similarly, when there is a little change in the intensity between two objects, some edge detectors may fail in detecting this small difference as an edge of the object.

Edges help in identifying the outline of an object. The primary goal of edge detectors is to output the edges required for further image-processing stages like detecting the object, its shape, size, and orientation. Edge detection is extensively used in the area of surgical and medical machines.

Most of the edge detection techniques are based on applying simple convolution masks to the entire image in order to compute the first-order and/or second-order derivative, thus resulting in an edge. A derivative is nothing but a calculation of differences in pixel values.

Edge detection can be divided into two types.

- First-order based edge detection (*Gradient*) - the first order derivative at a pixel is used to decide the presence of an edge. The first order derivative is searched for the maximum or the minimum value and the pixel containing this value is considered an edge. An example of this is the Sobel edge detector.

- Second-order based edge detection (*Laplacian*) - the second order derivatives are used to decide the presence of an edge. The pixel that has its second order derivative as zero is considered an edge, that is, this method searches for zero-crossings. An example of this is the Laplace edge detector.

In any of the above cases, a pre-defined template is created, generally of size 3 x 3. These templates are convolved with every neighbourhood in an image. The convolved value is assigned to the pixel under consideration.

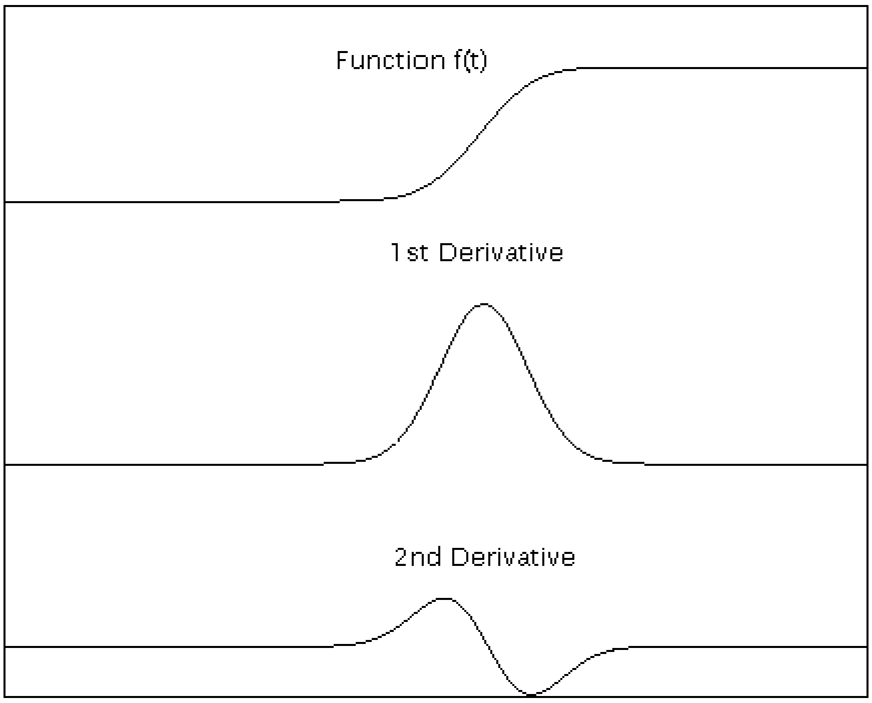


Figure 20: The general behaviour of an edge and its derivative: The relationship changes comparatively with the slope on the function `f(t)`, which is a cross-section of an edge in one particular direction [41].

Consider the image function `f(t)`. The corresponding first-order and second order derivatives are given as `f'(t)` and `f''(t)`. Whenever the first-order derivative (gradient) is local maximum or minimum corresponding to an edge pixel, the second-order derivative reaches zero. The width of the first and second order derivative is inversely proportional to the slope of the image function. This general behaviour of an edge and its derivatives is shown in Figure 20. Pixels having gradient values above a certain threshold are considered as edges.

## 4.1. Laplace edge detector

The standard *Laplace* edge detector calculates the partial second order derivatives along x and y direction for every pixel. The sum of these derivatives is assigned to the respective pixel, as shown in Equation 19.

$$L\{f(x,y)\} = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Equation 19: Laplace equation [4].

The edge direction can be formulated as in Equation 20.

$$q(x,y) = \tan^{-1}\left(\frac{\frac{\partial^2 f}{\partial y^2}}{\frac{\partial^2 f}{\partial x^2}}\right)$$

Equation 20: The edge direction for the Laplace edge detector.

The template form of this equation is shown in Equation 21 (a). This template only considers four immediate neighbours of the pixel (top, bottom, left, and right). Equation 21 (b) shows the template in which eight neighbours of the pixel are considered, that is, the four neighbours of Equation 21 (a) and the diagonal pixels. This is useful when the diagonal elements and the isolated pixels in the neighbourhood are considered for calculations. Equation 21 (c) is used when the diagonal elements need to be stressed more than the adjacent pixels. Similarly, in Equation 21 (d), the adjacent elements are stressed more than the diagonal elements. However, the templates (c) and (d) of Equation 21 are seldom used.

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

(a)

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -8 | 1 |
| 1 | 1 | 1 |

(b)

| | | |
|---|---|---|
| 2 | -1 | 2 |
| -1 | -4 | -1 |
| 2 | -1 | 2 |

(c)

| | | |
|---|---|---|
| -1 | 2 | -1 |
| 2 | -4 | 2 |
| -1 | 2 | -1 |

(d)

Equation 21: Laplace Edge Detection Template [4].

The two-dimensional Laplace template in Equation 21 (a) can be decomposed into two separate one-dimensional templates as shown in Equation 22. These two one-dimensional components can be applied separately to the image in either direction instead of a single two-dimensional template.

$$L_x = \begin{array}{|c|c|c|} \hline 1 & -2 & 1 \\ \hline \end{array}$$

(a)

$$L_y = \begin{array}{|c|} \hline 1 \\ \hline -2 \\ \hline 1 \\ \hline \end{array}$$

(b)

Equation 22: The Laplace filter: (a) $L_x$ - x component, (b) $L_y$ - y component.

Equation 22 (a) and (b) shows the x and the y component of the Laplacian operator resulting in detection of vertical and horizontal edges respectively.

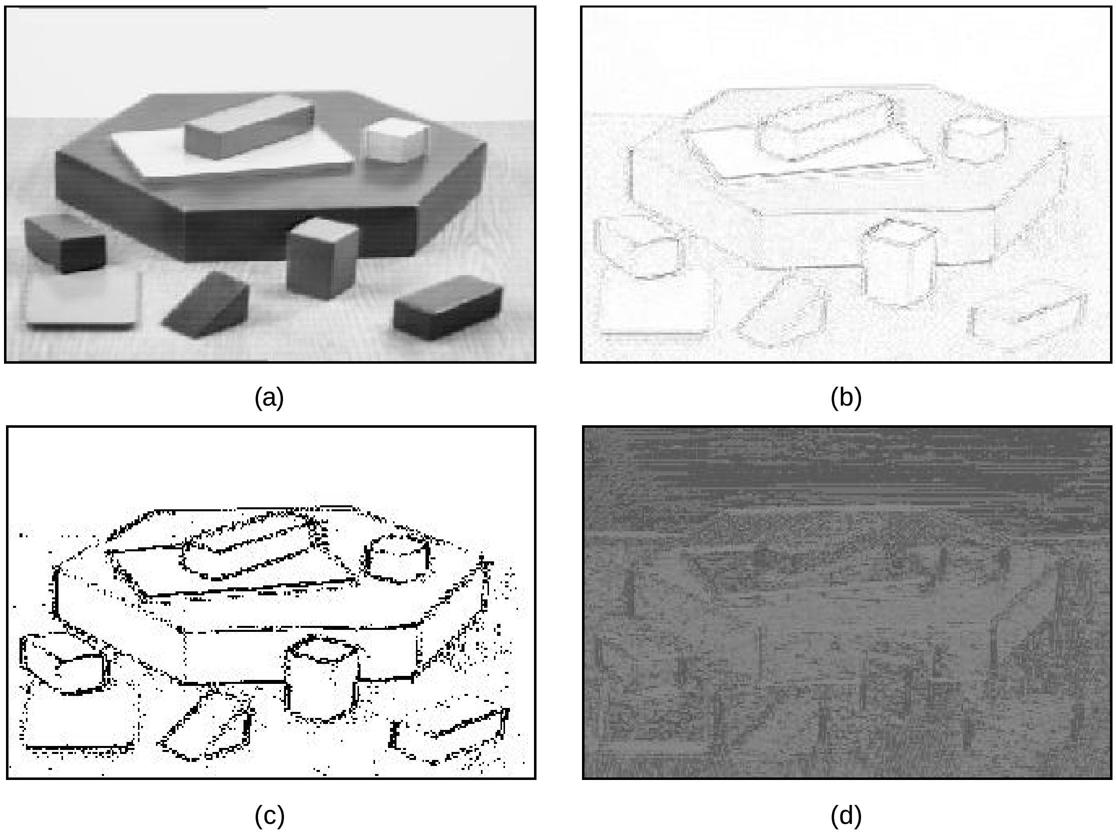


Figure 21: Laplace edge detector (Equation 21 (a)): (a) Original image, (b) Edge strength (Equation 21 (a)), (c) Binary edge strength (Threshold value = 26), (d) Edge direction (Equation 20).

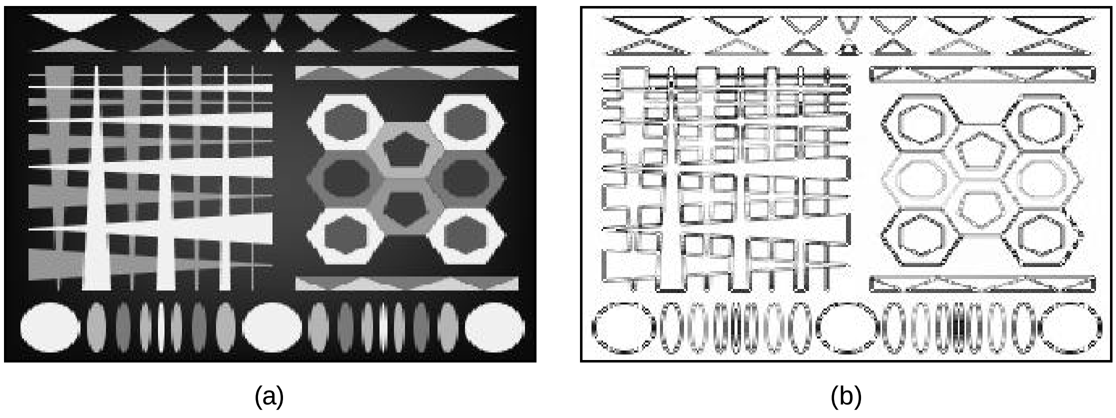


Figure 22: Laplace edge detector (Equation 21 (a)): (a) Original image, (b) Edge strength (Equation 21 (a)).

## 4.2. Laplacian of Gaussian (LoG)

The *Laplacian of Gaussian* (LoG) is the Laplace filter applied on an image that has been smoothed through the Gaussian filter. This two-step process can be divided into two one-step process to speed up calculations. First, the Laplace filter is applied on the Gaussian filter and the resulting kernel is applied to the image. This is shown in Equation 23.

$$L\,(G * I) = (L * G)\,I$$

Equation 23: Laplacian of Gaussian on Image is the same Laplace on the Gaussian of the Image.

The Laplace filter from Equation 19 is applied to the two-dimensional Gaussian filter from Equation 17 to result in Equation 24.

$$LoG\,(x, y) = \frac{1}{p s^{4}}\left(1 - \frac{x^{2} + y^{2}}{2 s^{2}}\right) e^{-\frac{x^{2} + y^{2}}{2 s^{2}}}$$

Equation 24: The Laplacian of Gaussian (LoG) function [41].

The kernel formed on selecting the standard deviation ($s$ ), shows the response as shown in Figure 23.

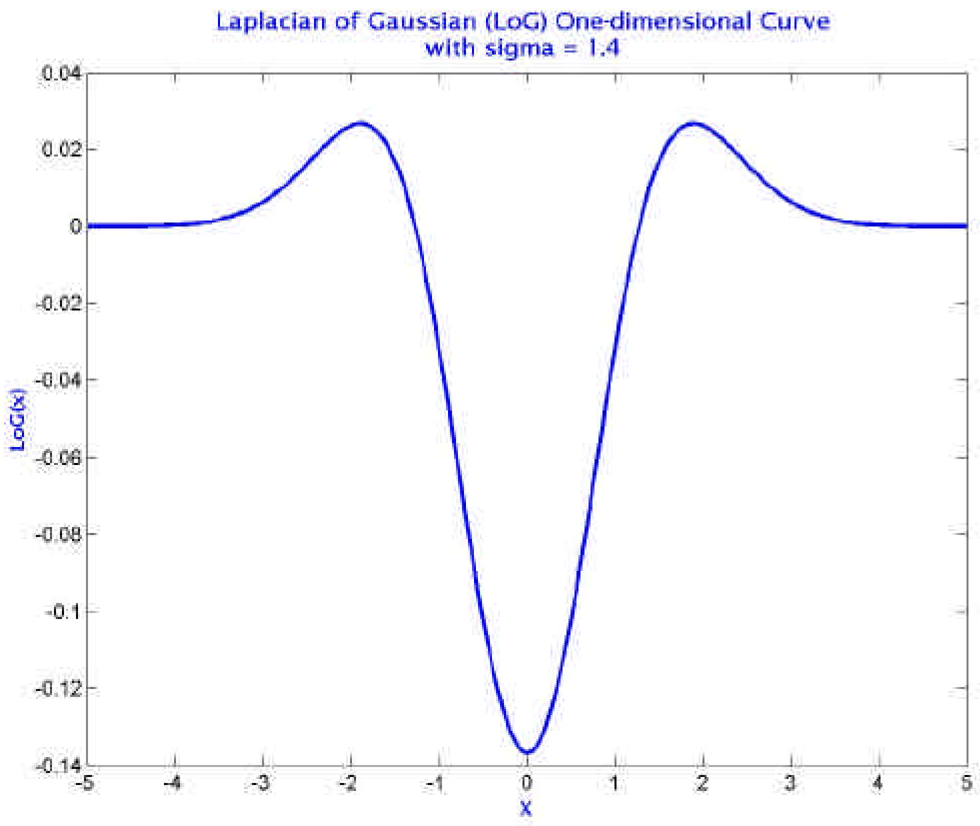


Figure 23: The response of the kernel formed for Laplacian of Gaussian (LoG) with a standard deviation ' s ' = 1.4.

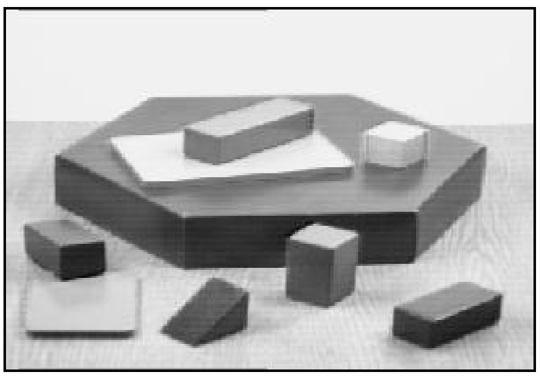
As shown in Equation 25, Fisher *et al* [41] stated a 9 x 9 kernel created using the standard deviation, ' *s* ' = 1.4 to achieve the edge detection.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 2 | 2 | 2 | 3 | 0 | 0 |
| 0 | 2 | 3 | 5 | 5 | 5 | 3 | 2 | 0 |
| 3 | 3 | 5 | 3 | 0 | 3 | 5 | 3 | 3 |
| 2 | 5 | 3 | -12 | -23 | -12 | 3 | 5 | 2 |
| 2 | 5 | 0 | -23 | -40 | -23 | 0 | 5 | 2 |
| 2 | 5 | 3 | -12 | -23 | -12 | 3 | 5 | 2 |
| 3 | 3 | 5 | 3 | 0 | 3 | 5 | 3 | 3 |
| 0 | 2 | 3 | 5 | 5 | 5 | 3 | 2 | 0 |
| 0 | 0 | 3 | 2 | 2 | 2 | 3 | 0 | 0 |

9 x 9

Equation 25: The Laplacian of Gaussian (LoG) pre-defined kernel calculated using standard deviation ( $s$ ) = 1.4 [41].

Generally, a smoothing filter needs to be applied before edge detection, in order to achieve better results. In the case of LoG, the smoothing and edge detection work at the same time. This is due to having applied the Laplace filter on the Gaussian filter.


(a)

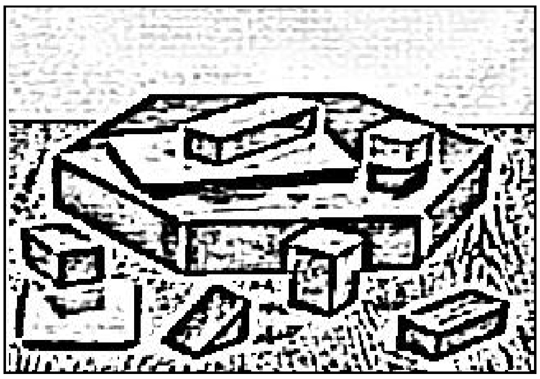
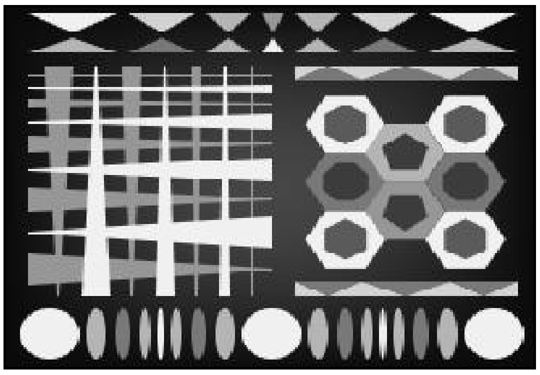

(b)

Figure 24: Laplacian of Gaussian 9x9 (Equation 25): (a) Original image, (b) Edge strength (Equation 25).


(a)

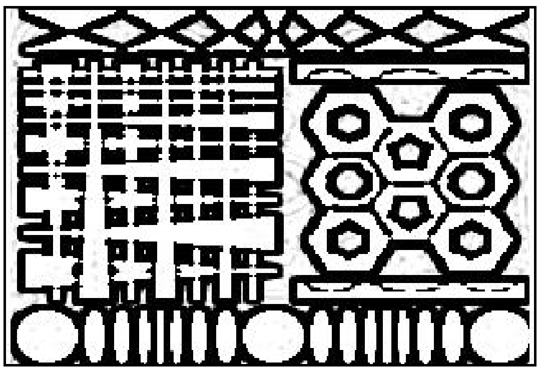
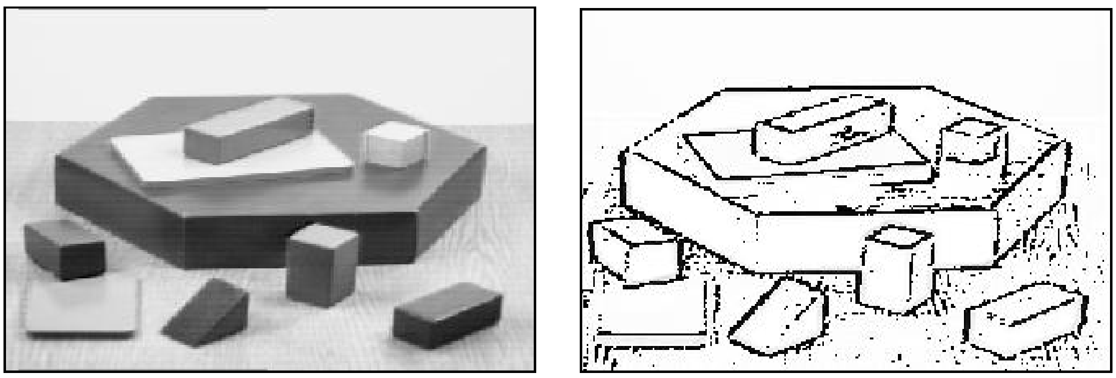

(b)

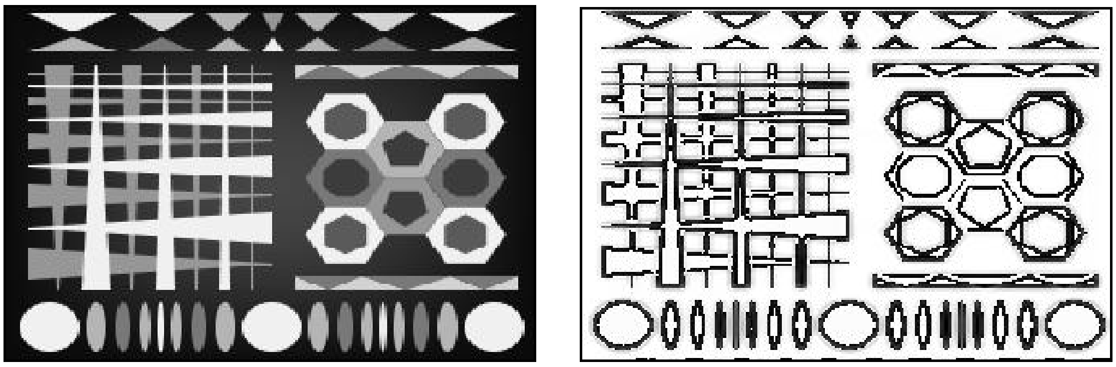
Figure 25: Laplacian of Gaussian 9x9 (Equation 25): (a) Original image, (b) Edge strength (Equation 25).

Apart from using a standard 9 x 9 kernel, the standard deviation '*s* ' = 1.5, was used for edge detection. The size of the kernel was found using Equation 15 as 11 x 11. The results can be seen in Figure 26 and Figure 27.



(a) (b)

Figure 26: Laplacian of Gaussian – two dimensional with $s$ = 1.5 (Equation 24): (a) Original image, (b) Edge strength (Equation 24).



(a) (b)

Figure 27: Laplacian of Gaussian – two dimensional with $s$ = 1.8 (Equation 24): (a) Original image, (b) Edge strength (Equation 24).

LoG with pre-defined kernel (Equation 25) was faster in producing the results but LoG with standard deviation ('$s$ ') = 1.5 (Equation 24) showed better results.

## 4.3. Difference of Boxes (DoB)

*Difference of Boxes* (*DoB*) is similar to LoG with a slight modification. Both use the concept of differentiation after smoothing. LoG performs Laplace filtering after Gaussian filtering whereas DoB takes the difference of two Mean filters of different dimensions. Instead of applying two Mean filters and taking difference, it internally computes the differences of two Mean filters and then applies the result of this as the kernel to the input image. Hence, it calculates two separate smoothing filters and then on computing the differences, edges are projected in the output image.

The larger the kernel sizes the thicker are the edges and the higher is the noise suppression. The simplest implementation of this detector is differencing the Mean 3 x 3 filter with Mean 1 x 1 filter (the pixel by itself). The result of this is thinner edges and less noise suppression. This is seen in Figure 28.

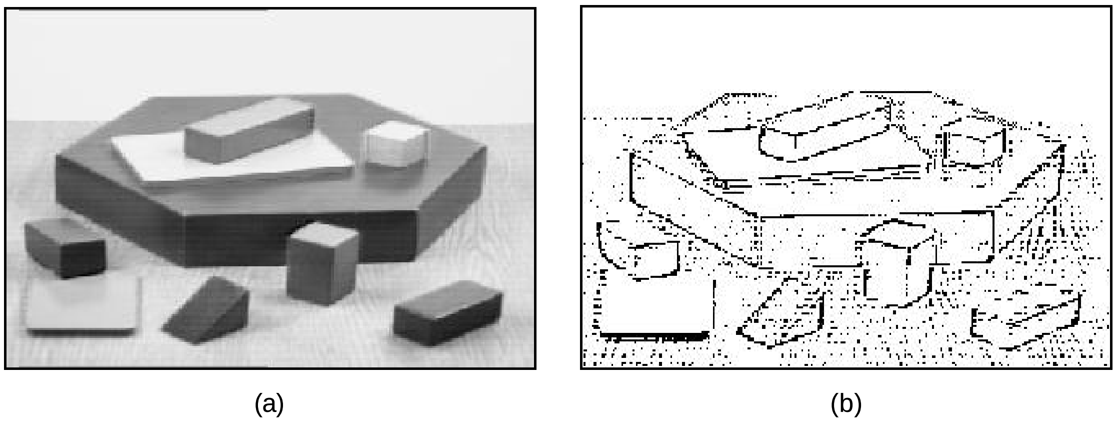


Figure 28: DoB calculated with Mean 3 x 3 and Mean 1 x 1: (a) Original image, (b) Edge strength.

In Figure 28 (b), the edges are not clear and more noise is apparent. This can be corrected by taking Mean 5 x 5 with Mean 1 x 1 (the pixel by itself). This is shown in Figure 29.

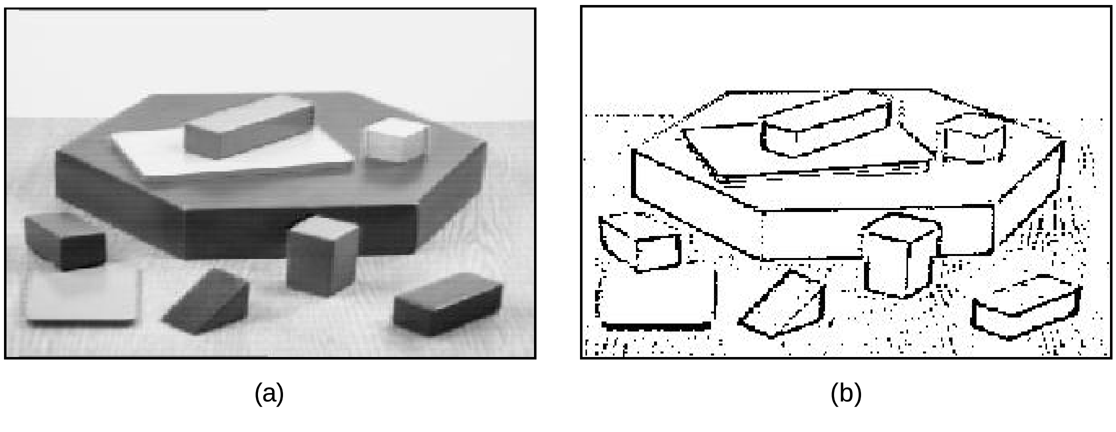
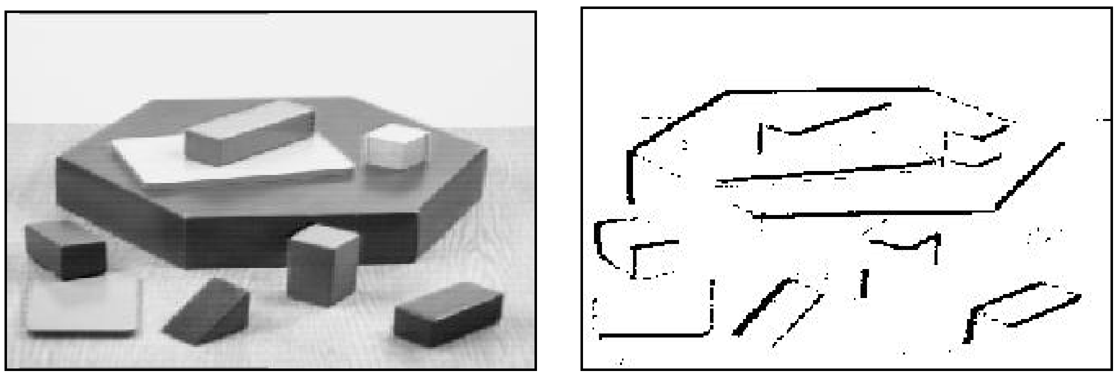


Figure 29: DoB calculated with Mean 5 x 5 and Mean 1 x 1: (a) Original image, (b) Edge strength.

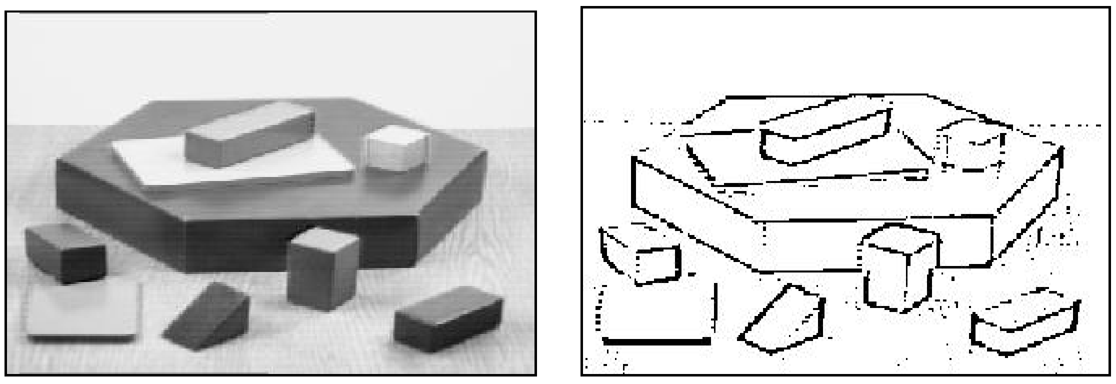
Using a Mean filter of even dimension leads to edges not being detected in all directions. This can be seen in Figure 30.



(a) (b)

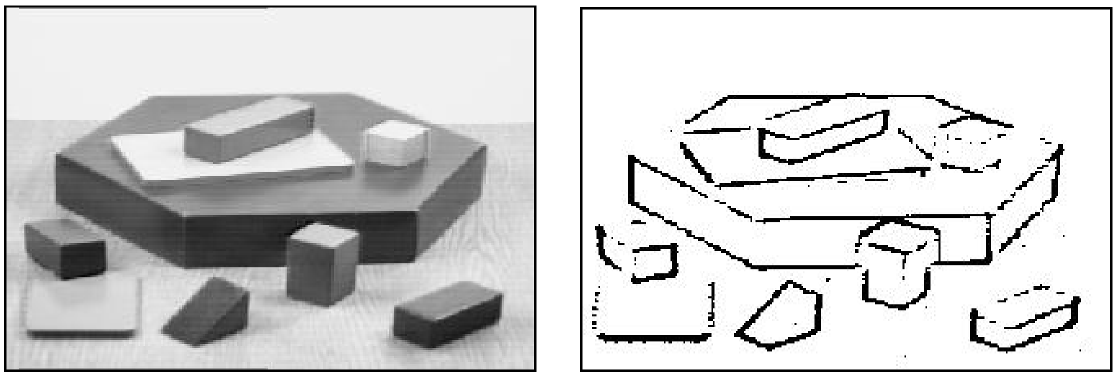
Figure 30: DoB calculated with Mean 3 x 3 and Mean 2 x 2: (a) Original image, (b) Edge strength.

In Figure 30 (b), many edges are missed because of using a Mean filter of even dimensions (2 x 2). Therefore, Mean filters with odd dimensions should be preferred. It was observed that using Mean filters of higher odd dimensions (5 x 5 or 7 x 7) result in better noise suppression. The results are shown in Figure 31 and Figure 32.



(a) (b)

Figure 31: DoB calculated with Mean 5 x 5 and Mean 3 x 3: (a) Original image, (b) Edge strength.



(a) (b)

Figure 32: DoB calculated with Mean 7 x 7 and Mean 3 x 3: (a) Original image, (b) Edge strength.

It was observed that as we increase the size of the Mean filters, edges in the output of the detectors grew thicker. DoB with Mean 5 x 5 and Mean 3 x 3 filters gave better results than any other, with respect to quality and speed.

## 4.4. Sobel edge detector

The *Sobel* edge detector calculates the gradient along the x and y direction separately.

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

(a)

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

(b)

| | | |
|---|---|---|
| 0 | -1 | -2 |
| 1 | 0 | -1 |
| 2 | 1 | 0 |

(c)

| | | |
|---|---|---|
| 2 | 1 | 0 |
| 1 | 0 | -1 |
| 0 | -1 | -2 |

(d)

Equation 26: Sobel Edge Detection Templates [18, 44].

The Sobel edge detector uses different weights for the position of pixels in the masks. This is because it computes the derivatives in one direction and at the same time, smoothes the image in another direction (averaging). These weights change with the larger sized filters and alteration in the image dimensions.

Generally, in Sobel edge detection, only the first two templates in Equation 26 (a) and (b) are considered. Equation 26 (a) shows the template required detecting the vertical edges or the edges made parallel to the y-axis, termed as $S_x$. Similarly, Equation 26 (b) shows the template required to detect horizontal edges or edges that are almost parallel to the x-axis, termed as $S_y$. The remaining two templates, Equation 26 (c) and (d), are seldom used to detect diagonal edges in respective directions, Equation 26 (a) and (b).

The two-dimensional Sobel x-component, $S_x$, in Equation 26 (a) can be decomposed into two one-dimensional components, $S_{x1}$ and $S_{x2}$, as shown in Equation 27. $S_{x1}$ computes the averaging in the direction of the edge and perpendicular to the edge strength while $S_{x2}$ computes the first-order differentiation along the direction of edge strength.

$$S_{x1} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \qquad\qquad S_{x2} = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline \end{array}$$

(a) (b)

Equation 27: Sobel – x component – $S_x$ is separated into: (a) $S_{x1}$ and (b) $S_{x2}$.

Similarly, the Sobel y-component, $S_y$, can be composed into $S_{y1}$ and $S_{y2}$ as shown in Equation 28.

$$S_{y1} = \begin{array}{|c|}\hline -1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array} \qquad S_{y2} = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array}$$

(a) (b)

Equation 28: Sobel – y component – $S_y$ is separated into (a) $S_{y1}$ and (b) $S_{y2}$.

Finally, the gradient value is computed by taking the square root of the sum of the squares of the gradients in either direction, as shown in Equation 29.

$$\begin{aligned} S_x &\rightarrow Sobel_x\,(x, y) \\ S_y &\rightarrow Sobel_y\,(x, y) \\ Sobel\,(x,y) &= \sqrt{S_x^{\,2} + S_y^{\,2}} \\ q\,(x, y) &= tan^{-1}\left(\frac{S_y}{S_x}\right) \end{aligned}$$

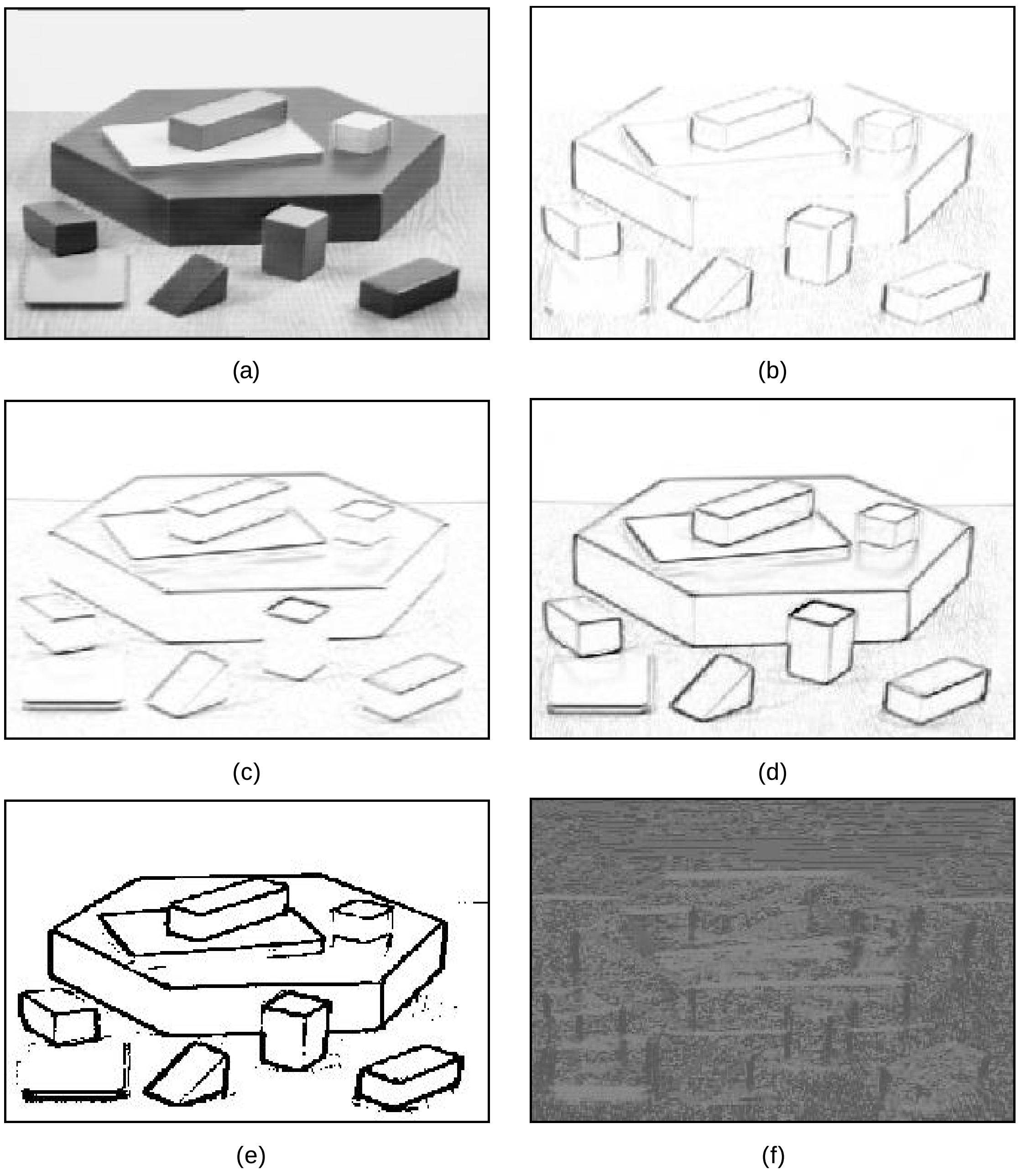
Equation 29: The Sobel edge calculation [4].

The edge magnitude is also calculated as the absolute values of $S_x$ and $S_y$. This can be written as Equation 30.

$$Sobel\,(x, y) = |S_x| + |S_y|$$

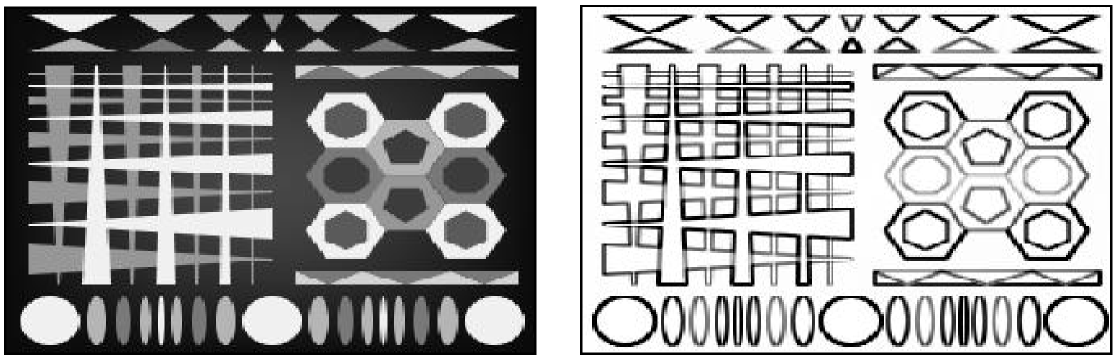
Equation 30: An alternative to calculate Sobel edge magnitude [4].

Due to the reduction in complexity, the calculation of edge magnitude in Equation 30 is faster in calculation of the values than that in Equation 29. For better results, the scale factor of $\frac{1}{3}$ was used for calculations. The results of the edge strength's in x and y directions along with the final edge results with edge direction are shown in Figure 33 and Figure 34.



(a) (b) (c) (d) (e) (f)

Figure 33: Sobel edge detector: (a) Original image, (b) Sobel x-component (Equation 26 (b)), (c) Sobel y-component (Equation 26 (a)), (d) Sobel in x and y direction - final edge strength (Equation 29) with scale factor = $\frac{1}{3}$, (e) Binary edge strength (Threshold value = 38), (f) Edge direction (Equation 29).



(a) (b)

Figure 34: Sobel edge detector (Equation 29) with scale factor = $\frac{1}{3}$: (a) Original image, (b) Edge strength (Equation 29).

## 4.5. Robert's edge detector

*Robert's* edge detection technique is the most basic of all the techniques discussed. It uses two 2 x 2 masks to find the orthogonal derivatives. Extension to the higher image dimensions is not possible in this edge detection technique. In addition, the gradient is not shifted by half-a-pixel in both directions. Due to this, Robert's edge detector is more sensitive to noise compared to other edge detectors. This is reflected in its output, which has a higher amount of noise that that of other filters.

(a)

| | |
|---|---|
| 1 | 0 |
| 0 | -1 |

(b)

| | |
|---|---|
| 0 | 1 |
| -1 | 0 |

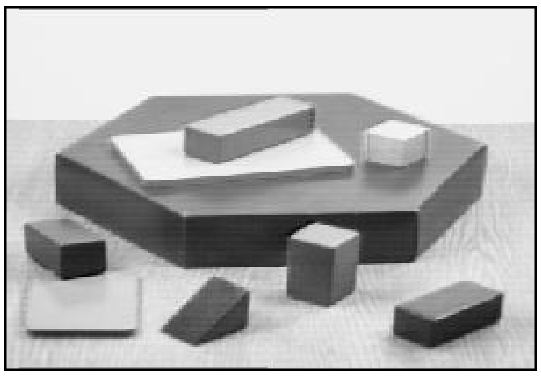
Equation 31: Robert's Edge Detection Templates [18, 44].

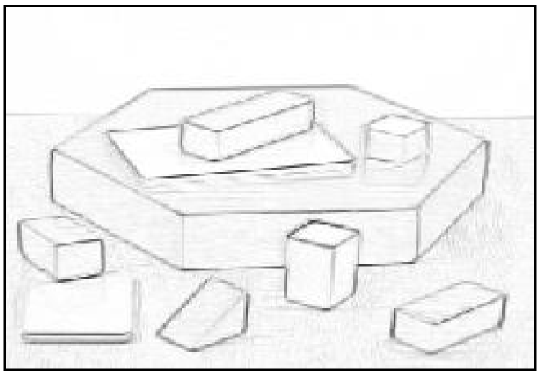
Equation 31 (a) and (b) computes the orthogonal derivatives, that is, the detector calculates derivatives along the diagonals, termed as $R_1$ and $R_2$ respectively. $R_1$ calculates derivatives along the major diagonal. This detects edges parallel to the minor diagonal. Similarly, $R_2$ calculates derivates along the minor diagonal, to identify edges parallel to the major diagonal.

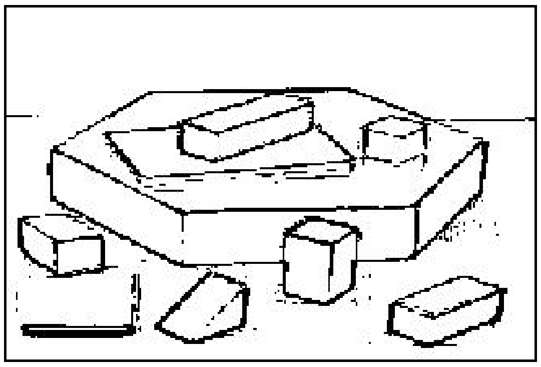
$$R_1 \rightarrow Roberts_1(x, y)$$
$$R_2 \rightarrow Roberts_2(x, y)$$
$$Roberts(x, y) = |R_1| + |R_2|$$
$$q(x, y) = tan^{-1}\left(\frac{R_2}{R_1}\right) + \frac{p}{4}$$

Equation 32: The Robert’s edge calculation [45].

To speed up calculations, the edge magnitude is calculated as the absolute values of the orthogonal derivatives. An angle of $\frac{p}{4}$ was added to the result of the edge direction [45]. This is because; the Roberts edge detector calculates the intensity fluctuations along diagonals, that is, it calculates orthogonal derivatives.


(a)


(b)


(c)

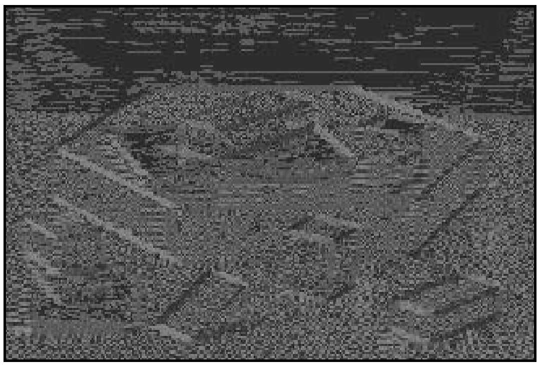
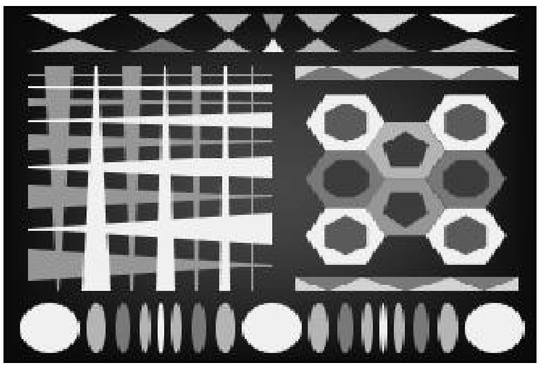

(d)

Figure 35: Robert's edge detector (Equation 32): (a) Original image, (b) Edge strength (Equation 32), (c) Binary edge strength (Threshold value = 31), (d) Edge direction (Equation 32).


(a)

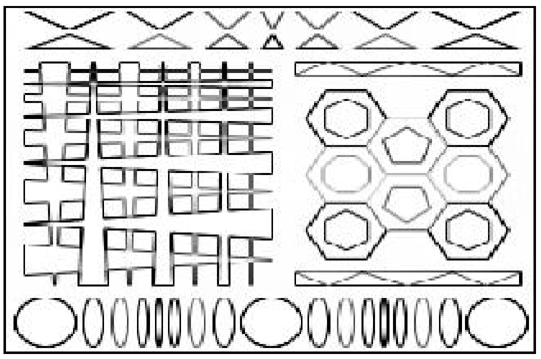

(b)

Figure 36: Robert's edge detector (Equation 32): (a) Original image, (b) Edge strength (Equation 32).

## 4.6. Kirsch edge detector

The *Kirsch* edge detector works in a similar fashion to the Sobel edge detector. The difference is that the weights are assigned to the different positions in the templates. In the Kirsch edge detector, weights are assigned to all pixel positions except the centre pixel, as shown in Equation 33.

| | | |
|---|---|---|
| 3 | 3 | 3 |
| 3 | 0 | 3 |
| -5 | -5 | -5 |

(a)

| | | |
|---|---|---|
| 3 | 3 | 3 |
| -5 | 0 | 3 |
| -5 | -5 | 3 |

(b)

| | | |
|---|---|---|
| -5 | 3 | 3 |
| -5 | 0 | 3 |
| -5 | 3 | 3 |

(c)

| | | |
|---|---|---|
| -5 | -5 | 3 |
| -5 | 0 | 3 |
| 3 | 3 | 3 |

(d)

Equation 33: Kirsch Edge Detection Templates [18, 44].

Again, the gradient is calculated along x and y directions, and along the diagonals. Equation 33 (a) shows the calculations of the gradient along the y-direction; (c) along the x-direction; and (b) and (d) along the diagonals. Similar to Equation 30, the edge magnitude of the Kirsch edge detector is calculated by adding the absolute values of the gradients in the x and y directions, as shown in Equation 34.

$$K_x \rightarrow Kirsch_x\ (x, y)$$
$$K_y \rightarrow Kirsch_y\ (x, y)$$
$$Kirsch\ (x, y) = |K_x| + |K_y|$$
$$q\ (x,y) = tan^{-1}\left(\frac{K_y}{K_x}\right)$$

Equation 34: The Kirsch edge calculation.

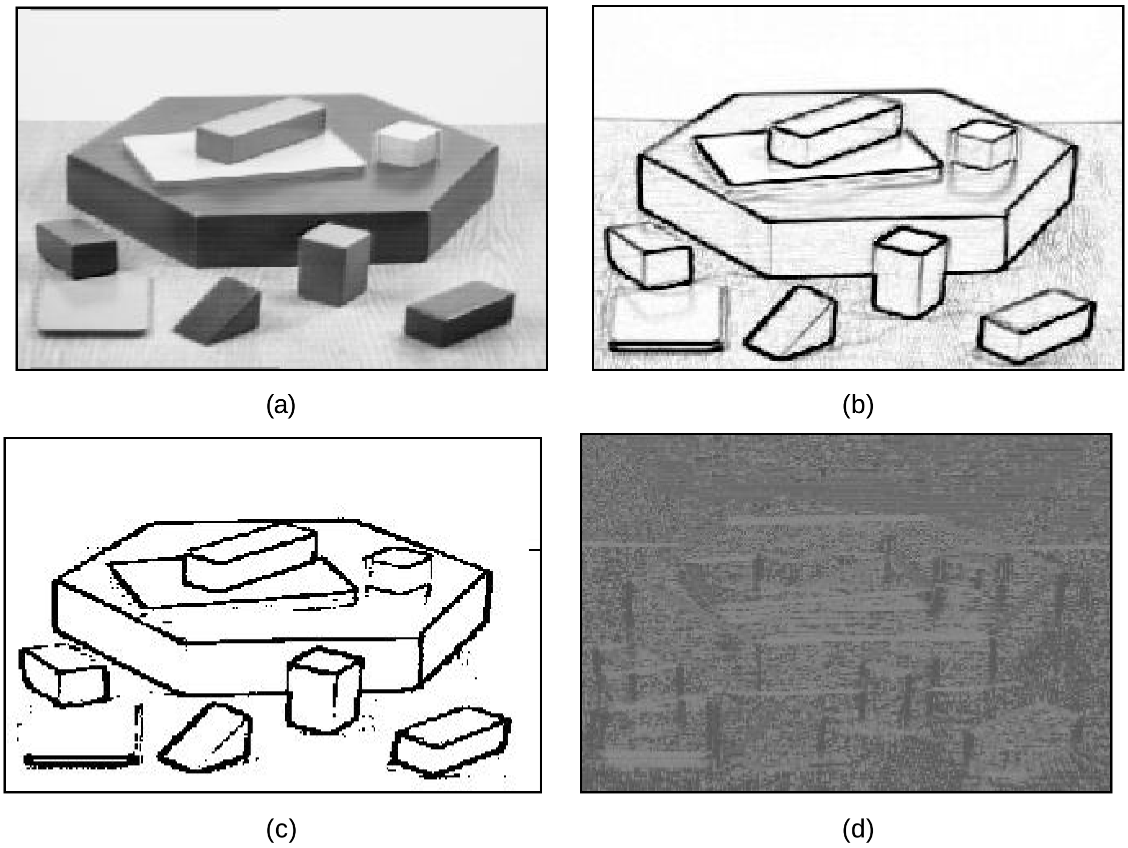


Figure 37: Kirsch edge detector (Equation 34) with scale factor = $\frac{1}{4}$: (a) Original image, (b) Edge strength (Equation 34), (c) Binary edge strength (Threshold value = 92), (d) Edge direction (Equation 34).

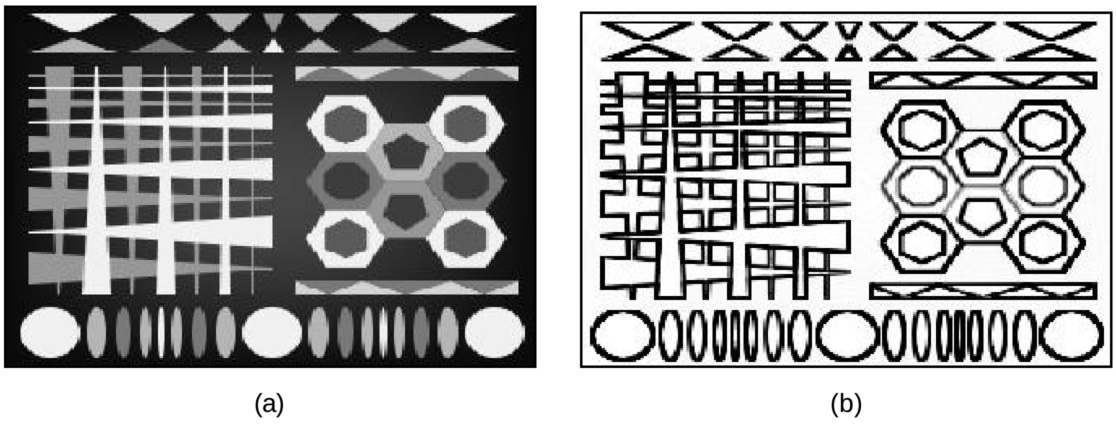


Figure 38: Kirsch edge detector (Equation 34) with scale factor = $\frac{1}{4}$: (a) Original image, (b) Edge strength (Equation 34).

## 4.7. Prewitt edge detector

The *Prewitt* edge detector behaves in the same way as the Sobel or Kirsch edge detectors. As discussed earlier, the change lies in the weights of the masks assigned to the respective pixels of the neighbourhood. The masks used in Prewitt edge detection are based on the Mean filter. All the values on one side of the pixel are considered positive and the other side equal and negative. The masks used for Prewitt edge detector are shown in Equation 35.

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

(a)

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| -1 | -1 | -1 |

(b)

| | | |
|---|---|---|
| 0 | -1 | -1 |
| 1 | 0 | -1 |
| 1 | 1 | 0 |

(c)

| | | |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | -1 |
| 0 | -1 | -1 |

(d)

Equation 35: Prewitt Edge Detection Templates [18, 44].

Equation 35 (a) shows the template used to detect the vertical edges, termed as $P_x$. Equation 35 (b) shows the template used to detect the horizontal edges, termed as $P_y$. Equation 35 (c) and (d) are used to compute the edges parallel to the major and minor diagonals, respectively: as with the Sobel and Kirsch edge detectors, these templates are rarely used.

$$\begin{aligned}
&P_x \rightarrow Prewitt_x\,(x, y)\\
&P_y \rightarrow Prewitt_y\,(x, y)\\
&Prewitt\,(x, y) = |P_x| + |P_y|\\
&q\,(x, y) = \tan^{-1}\left(\frac{P_y}{P_x}\right)
\end{aligned}$$

Equation 36: The Prewitt edge calculation.

The edge strength is the sum of the absolute values of the gradient calculated along the x and y directions. Like Sobel, Prewitt edge detector templates are separable. The two-dimensional templates shown in Equation 35 (a) and (b) can be separated into two one-dimensional components. The component in the direction of the edge work as an averaging filter and the other component compute the first-order differentiation in the direction of the edge response.

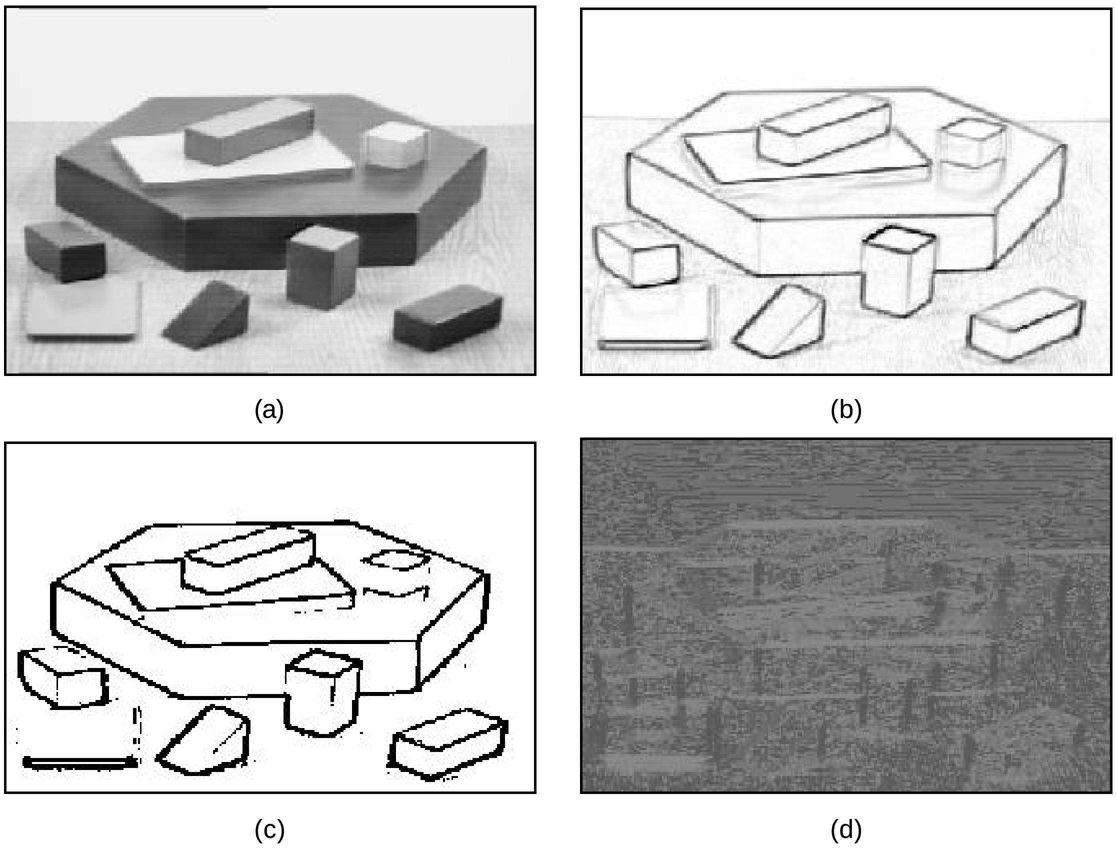


Figure 39: Prewitt edge detector (Equation 36) with scale factor = $\frac{1}{2}$: (a) Original image, (b) Edge strength (Equation 36), (c) Binary edge strength (Threshold value = 51), (d) Edge direction (Equation 36).

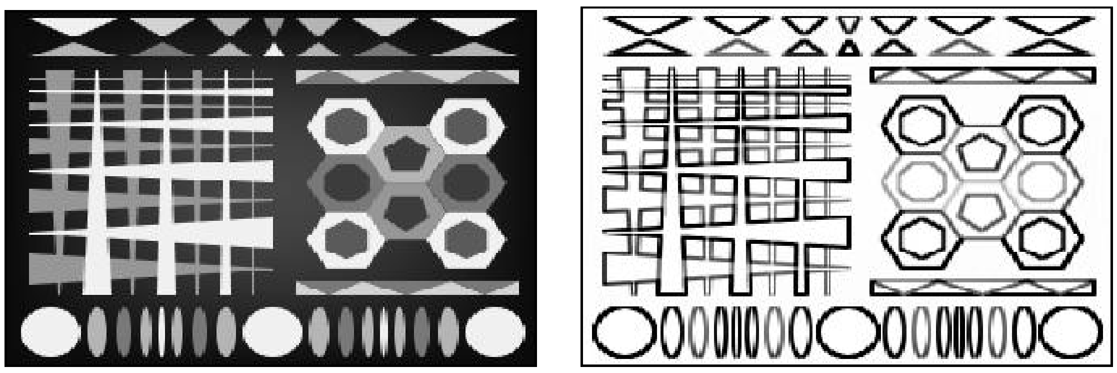


Figure 40: Prewitt edge detector (Equation 36) with scale factor = $\frac{1}{2}$: (a) Original image, (b) Edge strength (Equation 36).

## 4.8. Robinson edge detector

The *Robinson* detector performs in a different way from the other edge detectors. Apart from Laplace (which calculates the second-order derivative), Robinson is the only detector that stresses on the centre pixel for calculating the first order derivatives.

| | | |
|---|---|---|
| -1 | 1 | 1 |
| -1 | -2 | 1 |
| -1 | 1 | 1 |

(a)

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -2 | 1 |
| -1 | -1 | -1 |

(b)

| | | |
|---|---|---|
| 1 | 1 | 1 |
| -1 | -2 | 1 |
| -1 | -1 | 1 |

(c)

| | | |
|---|---|---|
| -1 | -1 | 1 |
| -1 | -2 | 1 |
| 1 | 1 | 1 |

(d)

Equation 37: Robinson Edge Detection Templates [18, 44].

Equation 37 (a) is used to detect the vertical edges, termed as $Ro_x$. Similarly, Equation 37 (b) shows the template for determining the horizontal edges in an image, termed as $Ro_y$; Equation 37 (c) and Equation 37 (d) are the templates used to determine the diagonal edges.

$$Ro_x \rightarrow Robinson_x\,(x, y)$$
$$Ro_y \rightarrow Robinson_y\,(x, y)$$
$$Robinson\,(x, y) = \left|Ro_x\right| + \left|Ro_y\right|$$
$$q\,(x, y) = tan^{-1}\left(\frac{Ro_y}{Ro_x}\right)$$

Equation 38: The Robinson edge calculation.

The edge strength is the sum of the absolute values of the gradient calculated along x and y directions.

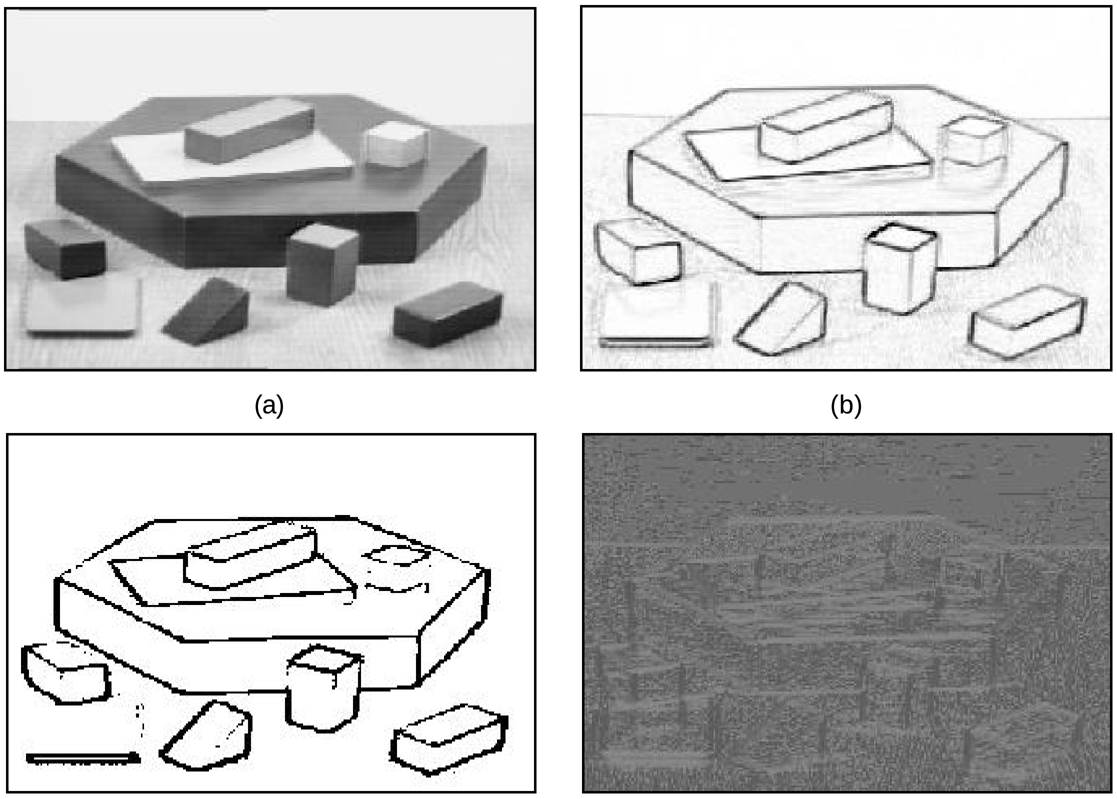
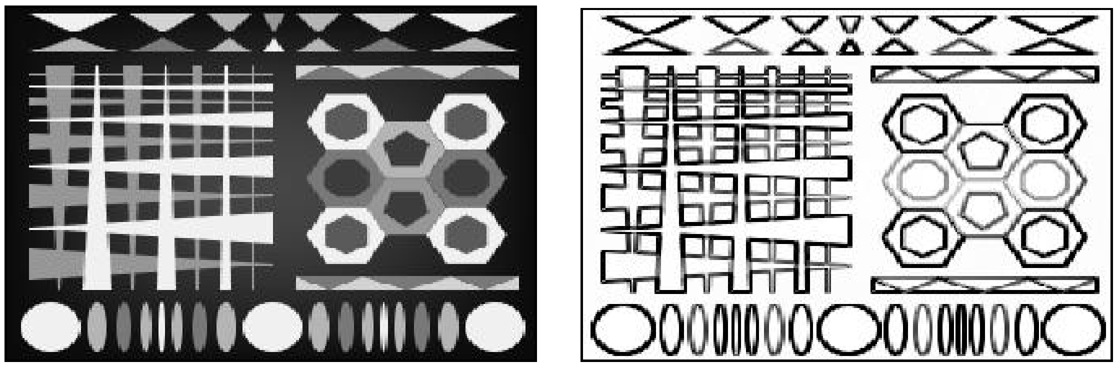


Figure 41: Robinson edge detector (Equation 38) with scale factor = $\frac{1}{2}$: (a) Original image, (b) Edge strength (Equation 38), (c) Binary edge strength (Threshold value = 64), (d) Edge direction (Equation 38).



(a) (b)

Figure 42: Robinson edge detector (Equation 38) with scale factor = $\frac{1}{2}$: (a) Original image, (b) Edge strength (Equation 38).

## 4.9. Comparison of edge detectors

Based on quality, Sobel edge detector showed the best results. Kirsch edge detector showed good results but the edges detected were very thick. Laplace and Robert's show broken edges at some junctions. LoG was the most expensive in computations. On taking an odd and even sized mean filters, DoB failed to show edges at certain locations. Hence, DoB should be used with odd-sized mean filters. Though Prewitt worked faster than Robinson did, their results were of average quality compared to the Sobel and Kirsch detectors. Table 2 shows the results of applying the different edge detectors, discussed earlier in the chapter.

The following are some of the key parameters that were selected for quantitative measurement of performance of the edge detectors.

- Original speed: This is the maximum speed, measured in fps (frames per second), possible with no edge detectors applied to the test image in Figure 21 (a). This value remains same for every edge detectors as the same image was used for testing all detectors. For the image used in these tests, the original speed was at a maximum of 130 fps.

- Final speed: This is the maximum speed with the particular edge detector applied on the image.

- Threshold: All the pixels having a value greater than the threshold were projected as edges. This was selected at run-time for the best results. The result of applying threshold is shown as binary edge strengths in respective edge detectors. The result of overlaying these edges on the original image is shown in Figure 86.

- Scale factor: A scale factor is multiplied with the final edge strength calculated. This is the scale factor observed to be the best when tested at run-time.

| | Final Speed | Threshold | Scale Factor |
|---|---|---|---|
| Laplace | 45.6 | 26 | 1 |
| LoG (9x9)[3] | 22.4 | 255 | 1 |
| LoG[4] | 4.5 | 230 | 1 |
| DoB (Mean 5 x 5 - Mean 3 x 3) | 18.7 | 5 | 1 |
| DoB (Mean 7 x 7 - Mean 3 x 3) | 12.2 | 8 | 1 |
| Sobel | 40.1 | 38 | $\frac{1}{3}$ |
| Robert's | 47.0 | 31 | 1 |
| Kirsch | 33.4 | 92 | $\frac{1}{4}$ |
| Prewitt | 39.1 | 51 | $\frac{1}{2}$ |
| Robinson | 38.4 | 64 | $\frac{1}{2}$ |

Table 2: Comparison of different edge detectors.

[3] - Discrete Approximation with '$s$' =1.4 for defining kernel of size 9 x 9 (Equation 25).

[4] - A kernel of size 11 x 11 was created using '$\sigma$' = 1.5 in Equation 15.

# 5. Corner detectors

## 5.1. Introduction and classification

In many applications, finding pairs of corresponding points in two images, that is, finding the geometric transformation of each of the two points between the two images is useful. Finding the changes in position and orientation is part of this calculation. Finding correspondence of all the points may be tedious, time-consuming, and highly complex. Hence, an effort is made to concentrate on certain points in the image, like corners. For example, when a square is considered, its transformation can be calculated if the corners are known: it is not necessary to know the entire set of points. The input for corner detectors is a gray level image and the output is an image in which all the corner pixels are marked. The difference between corners and edges is that at corners there is a large change in intensity when that pixel is compared with the neighbouring pixels in any direction. In addition, corners are subject to less variance than edges when any kind of transformations are made to the image, for example, rotation. There are two basic types of corner detectors: *template-based* and *geometry-based* corner detectors, as outlined by Shen *et al* [46].

### 5.1.1. Template-based corner detector

In template-based corner detection, a set of templates is created and then every pixel and the neighbouring pixels in the template are compared to produce certain results. The templates created can be of any size lesser than the size of the image. The templates are usually of size 3 x 3. When these templates are compared with the pixels of the images, it results in information that helps in judging whether a pixel is a corner. The templates shown in Figure 43 are the Laplace edge detector. Two examples, both of 3 x 3 templates, one with 4-

neighbourhood, and the second, with 8-neighbourhood, are shown in Figure 43.

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

(a)

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -8 | 1 |
| 1 | 1 | 1 |

(b)

Figure 43: The Laplace Operator with a 3 x 3 template with (a) 4-neighbourhood and (b) 8-neighbourhood.

In Figure 43, the centre of the template is convolved with the pixel of the image under consideration and then the neighbouring pixels are convolved with the values shown in the template. The result of this convolution is the edge value of that pixel under consideration. In this way all the pixels that are part of an edge will be shown in the output image. This system incurs high computational costs. The number of computations for the template shown in Figure 43 (a) and Figure 43 (b) will be 5*m*n and 9*m*n respectively, where m*n are the number of the pixels of the image.

### 5.1.2. Geometry-based corner detector

Geometry-based corner detectors measure the differential geometric features that lead to corners. This method involves many steps, such as extracting the boundary of the object as a chain code and then looking for significant change in the values of the chain code. This change of values means the direction of the next pixel is different from the previous pixel. This means that the pixel that indicates a change in direction is a corner. This is illustrated by Figure 44 and Figure 45.

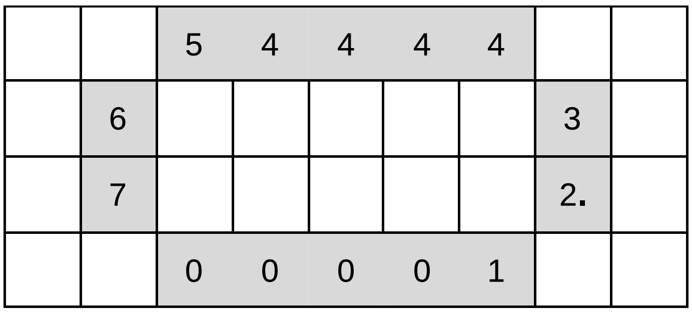


Figure 44: An object in an image. The boundary is marked gray. The figures shown in the pixels show the relative direction of the next pixel from the pixel under consideration.

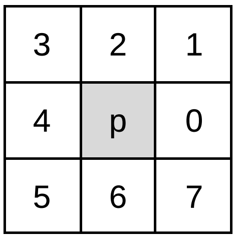


Figure 45: Each pixel is numbered according to the relative direction of that particular pixel with pixel under consideration, p.

The run length code starts from the rightmost bottom pixel of the chain, which is marked by a dot (.) after the value at that pixel. Using the convention shown in Figure 45, the code for the object boundary marked in Figure 44 can be written as 2, 3, 4, 4, 4, 4, 5, 6, 7, 0, 0, 0, 0, and 1. The first occurrence of the 2, 3, 4, 5, 6, 7, 0 and 1 are considered to be corners as the direction of the path changes. This is a time-consuming method as it is applied to all objects in the image. In addition, an error in segmentation while creating the chain will result in wrong pixels being identified as corners. If we consider the object shown in Figure 44, the computational costs will be 2 `* (i + j)` where `(i + j)` is the total number of pixels lying on the boundary of the object. The computational cost for this will depend on the size, shape, and length of the object.

## 5.2. Kitchen-Rosenfeld corner detector

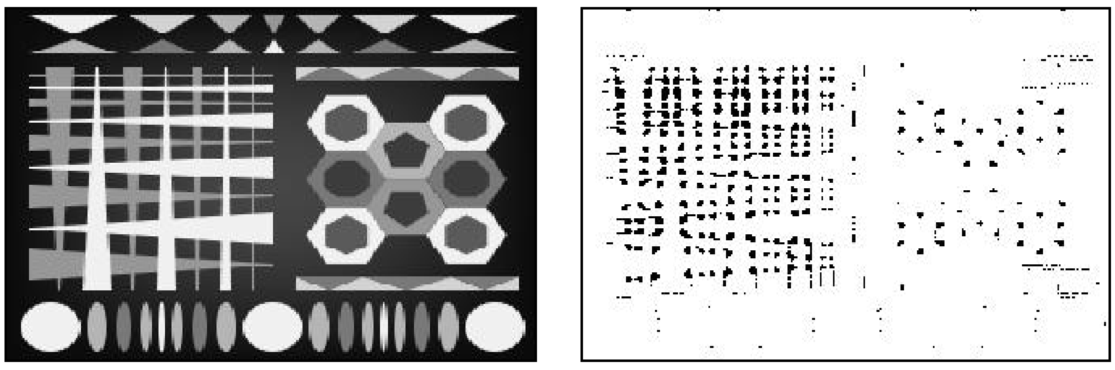
Kitchen and Rosenfeld proposed a corner detector [14] that works by taking second order derivatives along the direction of an edge.

$$KR(i,j) = \frac{I_x^2 I_{yy} - 2 I_x I_y I_{xy} + I_y^2 I_{xx}}{I_x^2 + I_y^2}$$

Equation 39: Kitchen-Rosenfeld corner detector.

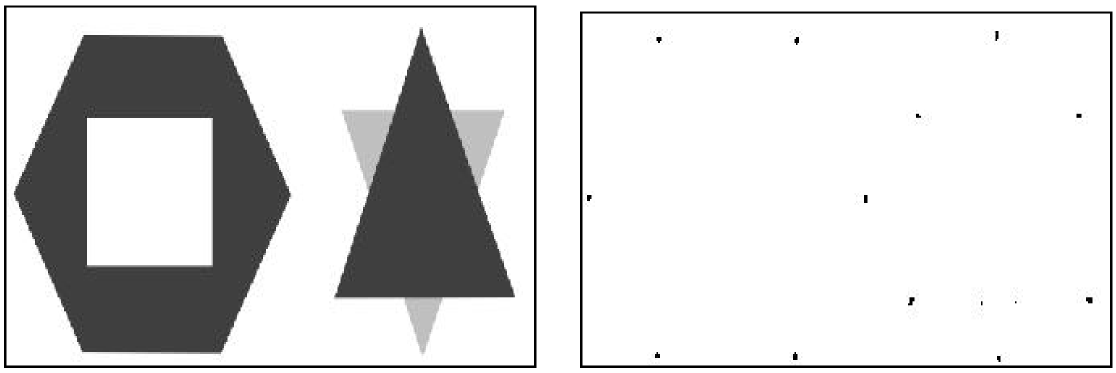
The Kitchen-Rosenfeld corner detector can be written as Equation 39. Here, $I_x$ and $I_y$, and $I_{xx}$ and $I_{yy}$ are the first-order and second-order derivatives along x and y directions, respectively. $I_{xy}$ is the partial order derivative along the x and y directions. The first derivative along the x direction is calculated and then the derivative of this result is computed along the y direction.

On gray-scale images, Kitchen-Rosenfeld is accurate and not very sensitive to noise. Applying the corner detector directly to binary images did not give good results. However, if the filter was applied after edge detection, the results were accurate. In order to view the corners in the output image, a threshold is applied to the result of the detector.



(a) (b)

Figure 46: Kitchen Rosenfeld corner detector (Equation 39): (a) Original image, (b) Corners.



(a) (b)

Figure 47: Kitchen-Rosenfeld corner detector (Equation 39): (a) Original image, (b) Corners.

The initial test image in Figure 46 (a) and Figure 47 (a) was smoothed using the Mean 3 x 3 filter. Kitchen-Rosenfeld was applied to this result. Finally, this result was overlayed on the initial test image. The result of overlaying in Figure 47 (b) was tested with threshold value 18 (Figure 87).

## 5.3. Beaudet corner detector

From Hessian matrix form [12], the derivatives are written as Equation 40 (a). Equation 40 (b) shows an example of these values.

$$\begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} \qquad \begin{bmatrix} 20 & 15 \\ 15 & 20 \end{bmatrix}$$

(a) (b)

Equation 40: The second-order derivatives along x and y directions and partial derivative along x and y direction: (a) Hessian matrix form, (b) An example of the Derivatives' value.

The *Beaudet* corner detector [13] is different from the others, in that it just calculates the determinant (`'det'`). The higher the determinant (the value `'det'` obtained from Equation 41), the higher the chance of a pixel being a corner. Hence, from the Hessian matrix form, ‘det’ can be calculated as shown in Equation 41.

$$det\,(i,j) = I_{xx}\, I_{yy} - I_{xy}{}^{2}$$

Equation 41: Beaudet corner detector’s equation for computing the ability of the pixel to be a corner.

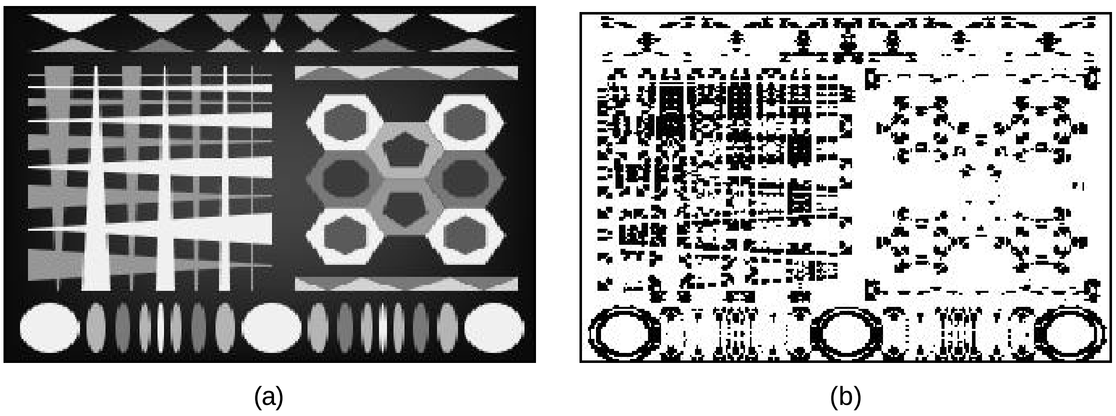

(a) (b)

Figure 48: Beaudet corner detector (Equation 41): (a) Original image, (b) Corners.

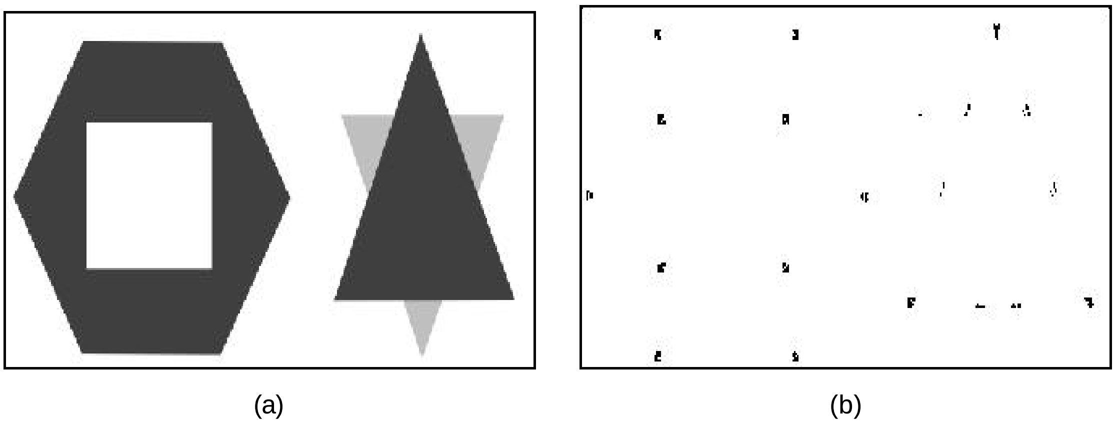

(a) (b)

Figure 49: Beaudet corner detector. (Equation 41): (a) Original image, (b) Corners.

The Beaudet corner detector was applied to Figure 48 (a), and Figure 49 (a). Prior to this, both images were smoothed using the Mean 5 x 5 filter. The result in Figure 49 (b) was overlayed at the threshold value 255 (Figure 87).

## 5.4. Plessey corner detector

Based on Equation 40, the determinant (‘detm’) is calculated as Equation 43. The value of ‘tracem’ is calculated by adding the major diagonal elements. Whether a pixel is a corner can be decided by the value of ‘plessey’ that is assigned to every pixel. Equation 42 shows a matrix based on first differentials of images was designed, which was further used to calculate whether a pixel is a corner.

$$C = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Equation 42: Matrix for calculating first differentials [12].

Based on this matrix, calculations are made in calculating ‘plessey’ as shown by Equation 43.

$$\begin{aligned} detm &= I_x^2 I_y^2 - (I_x I_y)^2 \\ tracem &= I_x^2 + I_y^2 \\ plessey(i,j) &= detm / tracem \end{aligned}$$

Equation 43: The calculation of det and trace, and the ability of the pixels to be corner in Plessey corner detector.

The value of ‘plessey’ is more at the junction of the edges, where the possibility of a pixel being a corner is high.

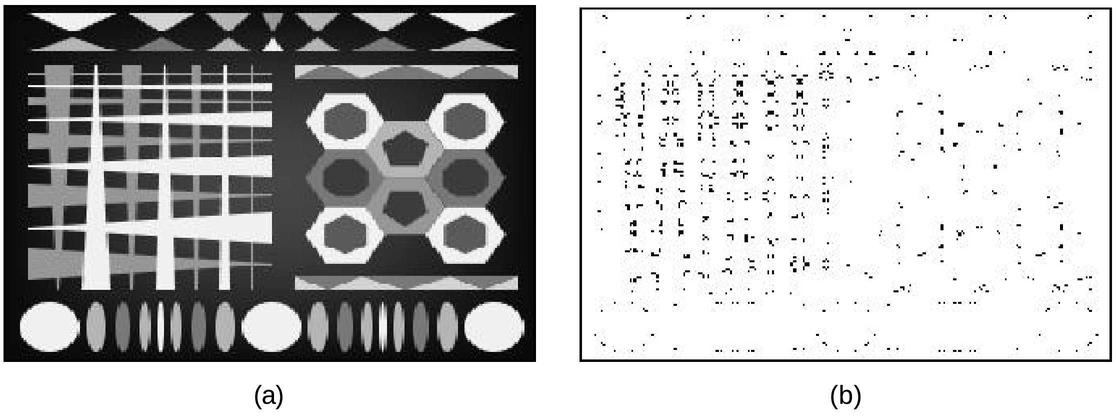


Figure 50: Plessey corner detector (Equation 43): (a) Original image, (b) Corners.

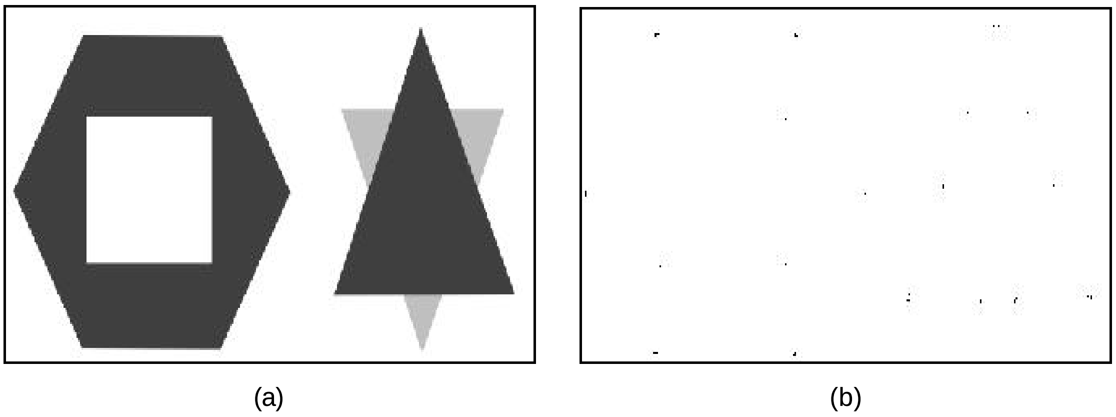


Figure 51: Plessey corner detector (Equation 43): (a) Original image, (b) Corners.

The Mean 3 x 3 filter was applied to Figure 50 (a) and Figure 51 (a) followed by the Plessey corner detector. The result in Figure 51 (b) is overlayed on the initial image at the threshold value of 43 (Figure 87).

## 5.5. Noble corner detector

The *Noble* corner detector [16] works in a similar manner to the Plessey corner detector. This detector was based on the theoretical formulation for the corner detection problem using differential geometry [12]. The matrix shown in Equation 42 was modified to another form as seen in Equation 44.

$$C = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} + s^2 \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}^2$$

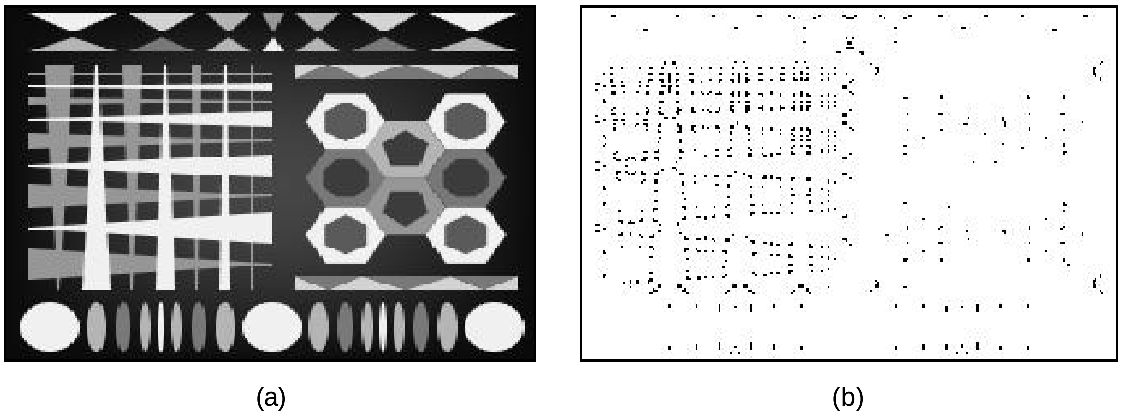
Equation 44: Modification in representing the matrix shown in Equation 42 for further calculation for Noble corner detector [12].

Using this matrix, '`detm`' and '`tracem`' are calculated as shown in Equation 45.

$$\begin{aligned} detm &= (I_x^2 I_y^2 - (I_x I_y)^2) \\ &-s^2((I_{xx}^2 + I_{xy}^2)(I_{yy}^2 + I_{xy}^2) - (I_{xx} I_{xy} + I_{yy} I_{xy})^2) \\ tracem &= I_x^2 + I_y^2 + s^2(I_{xx}^2 + I_{yy}^2 + 2 I_{xy}^2) \\ noble(i,j) &= detm / tracem \end{aligned}$$

Equation 45: The calculation of det and trace, and the ability of the pixels to be corner in Noble corner detector.

In this detector, the value of sigma ('$s$') is manually assigned. This value was varied at run-time and was tested on every pixel to check whether the pixel is a corner.



(a) (b)

Figure 52: Noble corner detector: (a) Original Image, (b) Corners.

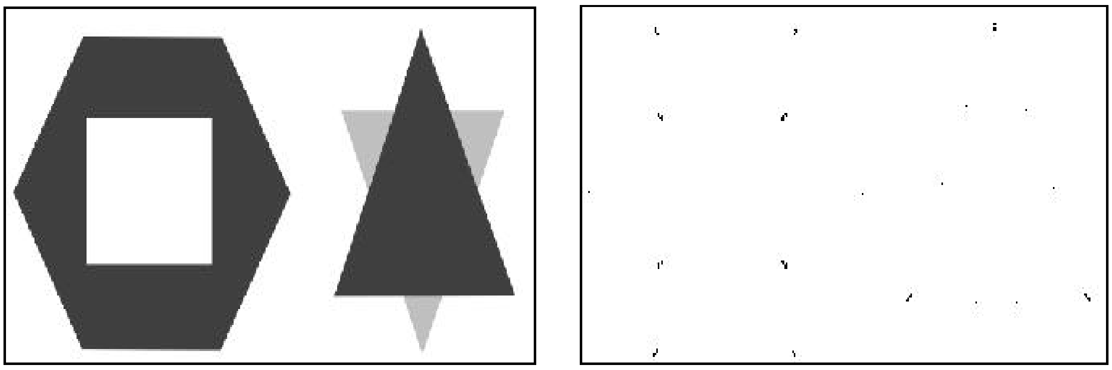


Figure 53: Noble corner detector: (a) Original Image, (b) Corners.

The Mean 3 x 3 filter was applied to Figure 52 (a) and Figure 53 (a) followed by the Noble corner detector. The result in Figure 53 (b) is overlayed on the initial image at the threshold value of 255 (Figure 87).

## 5.6. Harris-Stephens corner detector

Plessey corner detector was slightly modified to result in the *Harris-Stephens* corner detector [17]. It is also based on the concept of first order differentials. The determinant is calculated and termed as '`detm`' and the sum of the major diagonals of this matrix is termed as '`tracem`'.

$$detm = I_x^2 I_y^2 - (I_x I_y)^2$$
$$tracem = I_x^2 + I_y^2$$

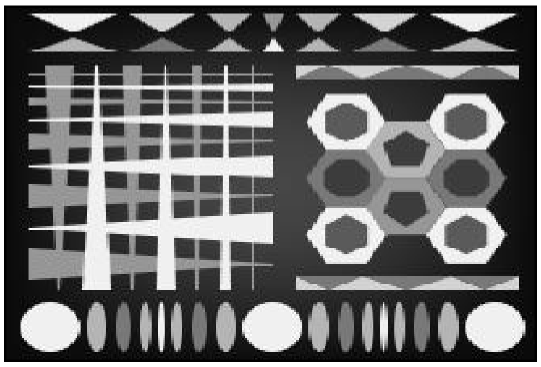
Equation 46: Det and Trace derived from the basics of matrices and using the above Equation 40.

The *Harris-Stephens* corner detector is calculated using '`detm`' and '`tracem`' as in Equation 46. This is calculated as in Equation 47.

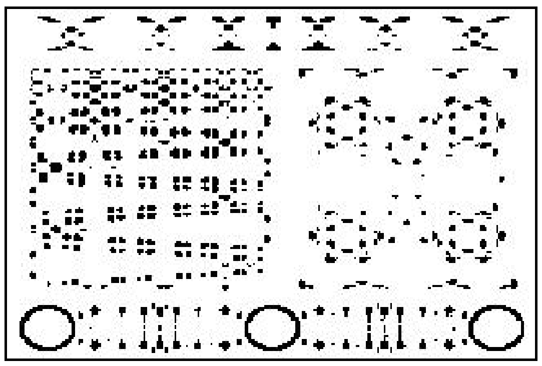
$$harris(i, j) = detm - k * (tracem)^2$$

Equation 47: Harris-Stephens corner detector's equation for computing the ability of the pixel to be a corner [12].

The term `'harris'` is the calculated value shown in Equation 47. At corners, this value is higher than the neighbourhood pixels.

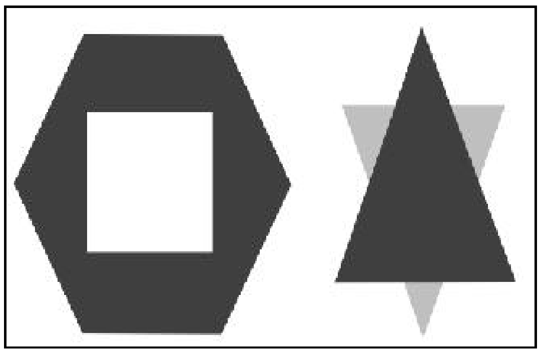


(a)

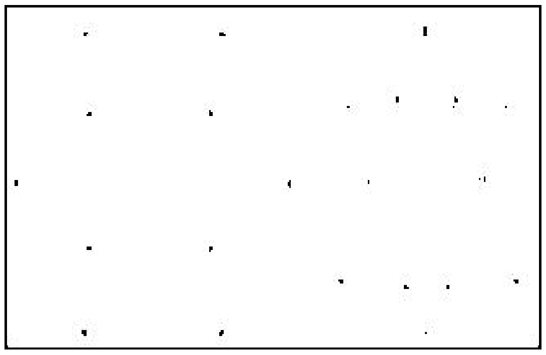


(b)

Figure 54: Harris-Stephens corner detector (Equation 47): (a) Original image, (b) Corners.



(a)



(b)

Figure 55: Harris-Stephens corner detector (Equation 47): (a) Original image, (b) Corners.

The Harris-Stephens corner detector was applied to the test image shown in Figure 54 (a) and Figure 55 (a). The corner detector was applied to the result of Mean 5 x 5 filter applied on the initial image. The Mean 5 x 5 filter was necessary because this detector required higher blurring. The result shown Figure 54 (b) and Figure 55 (b) was overlayed on the initial image with the threshold value of 217 (Figure 87).

## 5.7. A corner detector based on the concept of zero-crossing

In a *zero-crossing* edge detector, the second-order derivative of an edge response always passes through the origin as shown in Figure 20. That means the value to the left of this zero-crossing second-order derivative has an opposite polarity from the value to the right. This can be illustrated by Figure 20. Apart from being of opposite polarities, ideally, these values are equal in magnitude. However, these values are generally not the same, due to the change in real-time calculations of the edge-response and its derivatives. The first-order derivative has a curve similar to a Gaussian distribution curve. Hence, corresponding to the position where the second-order derivative turns zero, there is a maximum value or maxima in the first-order derivative.

Similarly, an ideal corner has the response similar to a Gaussian distribution curve Figure 56. This corner response was used to check whether a pixel is a corner. An attempt was made to compute this possibility by using the values of the corner responses' first and second-order derivatives.

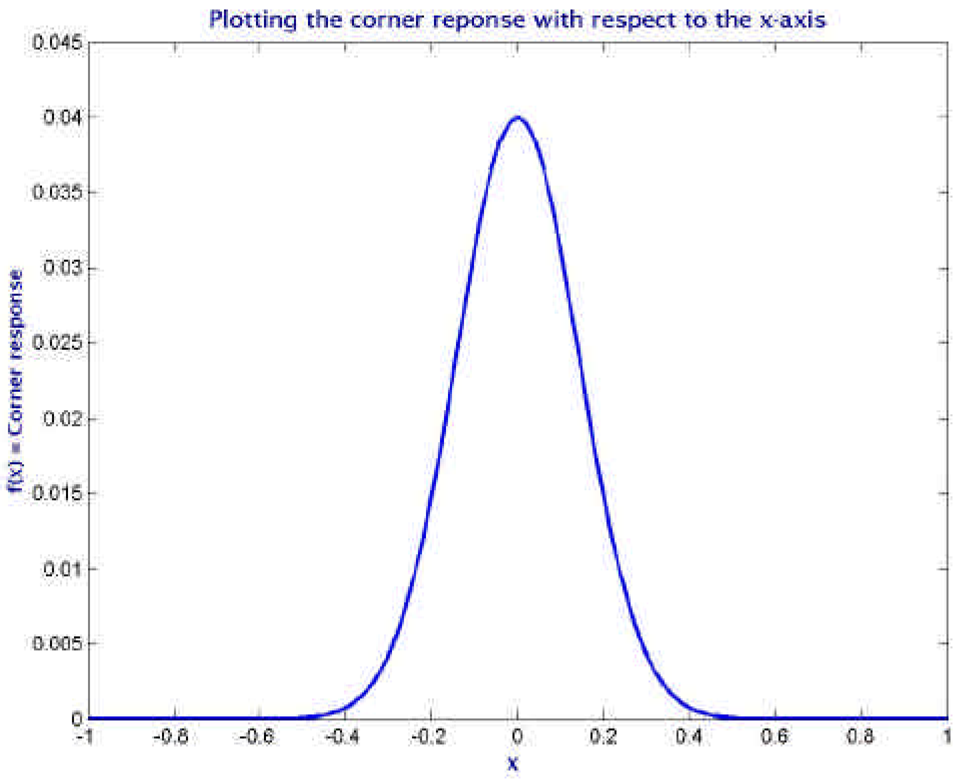


Figure 56: A Corner response.

Hence, when a derivative of this corner response is computed, the highest value (or the peak) of the initial response corresponds to the zero-value in the first-order derivative curve. This is shown in Figure 57.

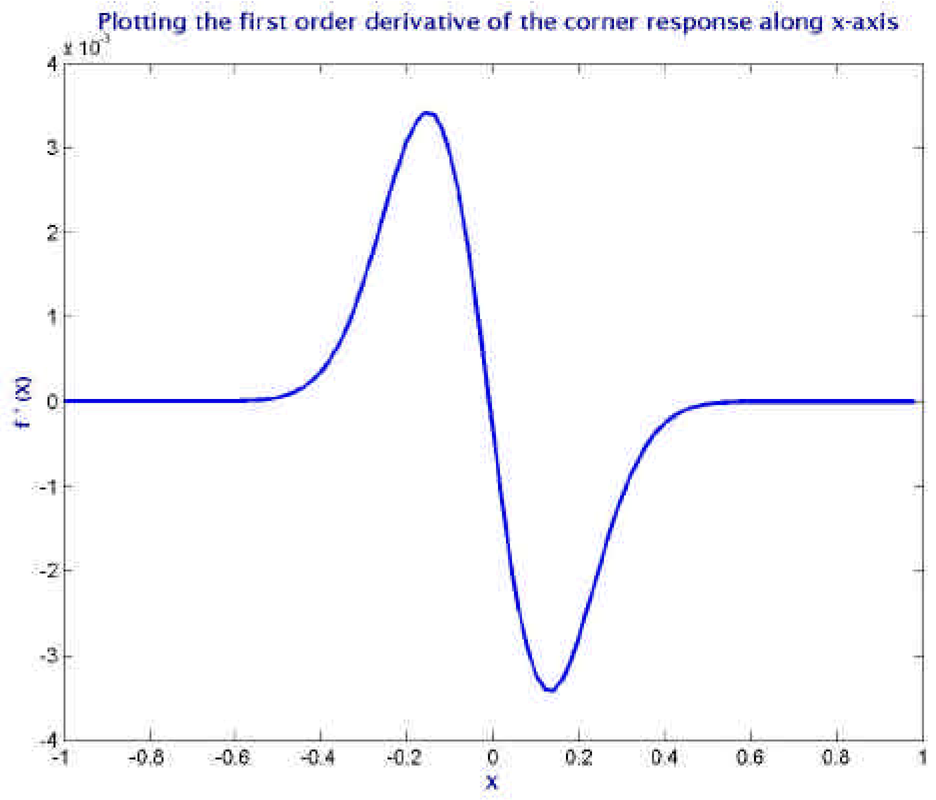


Figure 57: The first-order derivative of the initial corner response shown in Figure 56.

In addition, the second-order derivative of this initial function (corner response) has its minima or minimum value at the position where the first-order derivative passes through zero. Let us name this concept *Zero-1*. This can be better understood from Figure 58.

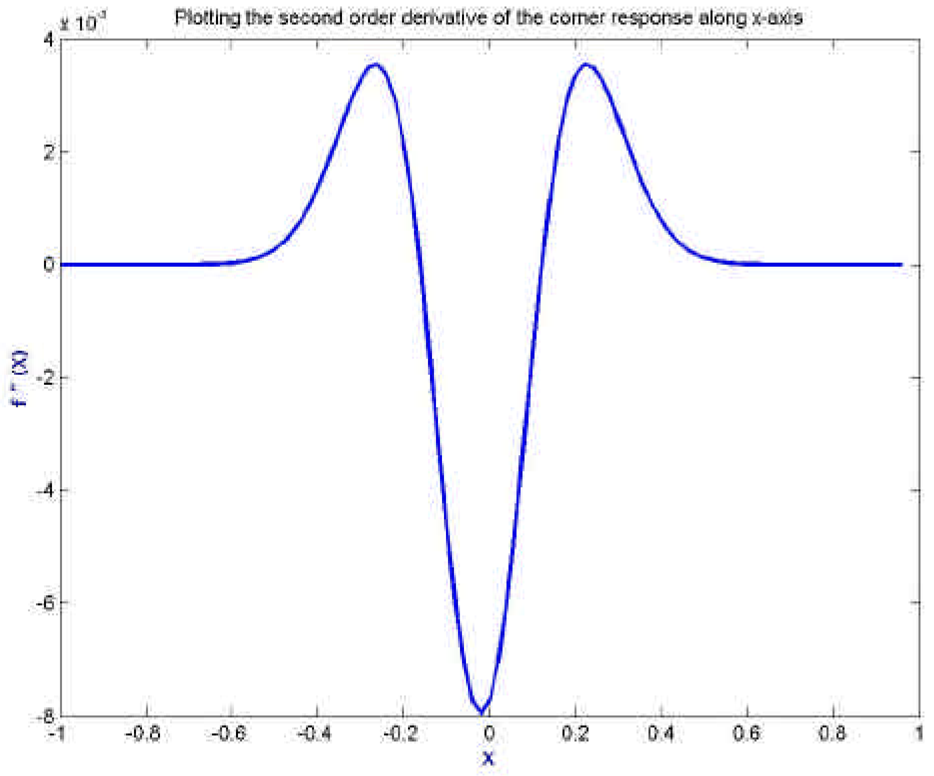


Figure 58: The second order derivative response of the initial corner response from Figure 56.

Adding to the *Zero-1* concept, both the second-order derivatives along x and y were checked to be equal. This leads to another corner detection technique. Let us name this concept *Zero-2*.

Two corner detectors based on the concepts *Zero-1* and *Zero-2* were tried and tested on several image sequences.

The second-order derivatives of the corner response along the x and y directions are not always both equal and minimum in the same position. This is because real-time images may have random pixel values (intensity). Hence, the *Zero-1* concept outperforms the *Zero-2* concept due to the general behaviour of a corner, where at least one of the second-order derivatives of the corner is at minima in that region.

*Zero-2* may be preferred over *Zero-1* in applications that are constrained by memory, as *Zero-2* requires less memory for storing its results.

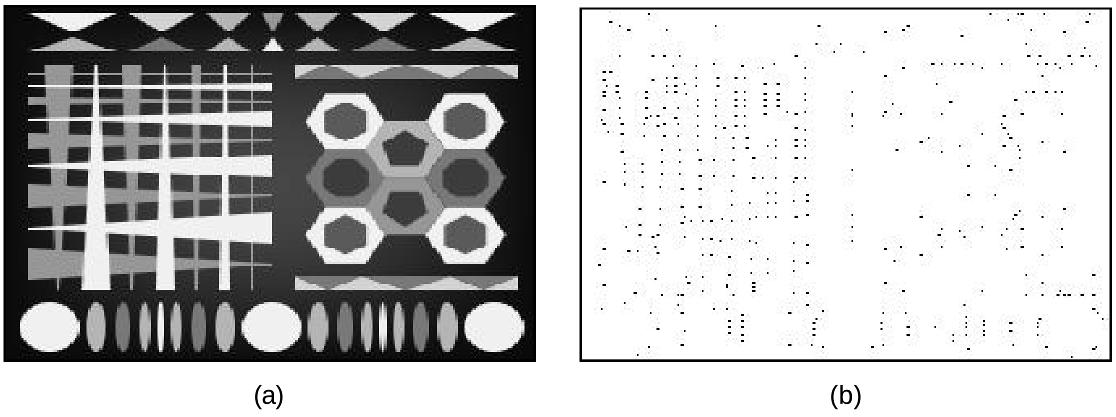

(a) (b)

Figure 59: A corner detector after Zero-1 concept: (a) Original image, (b) Corners.

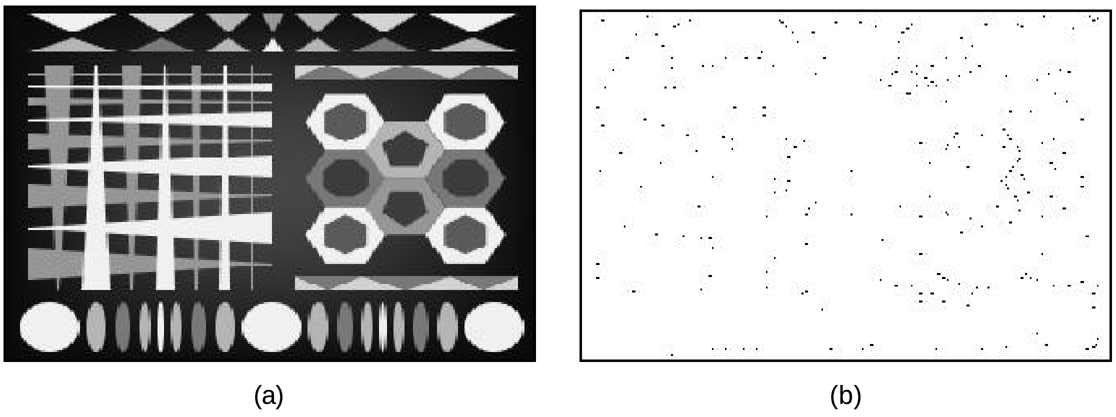

(a) (b)

Figure 60: A corner detector after Zero-2 concept: (a) Original image, (b) Corners.

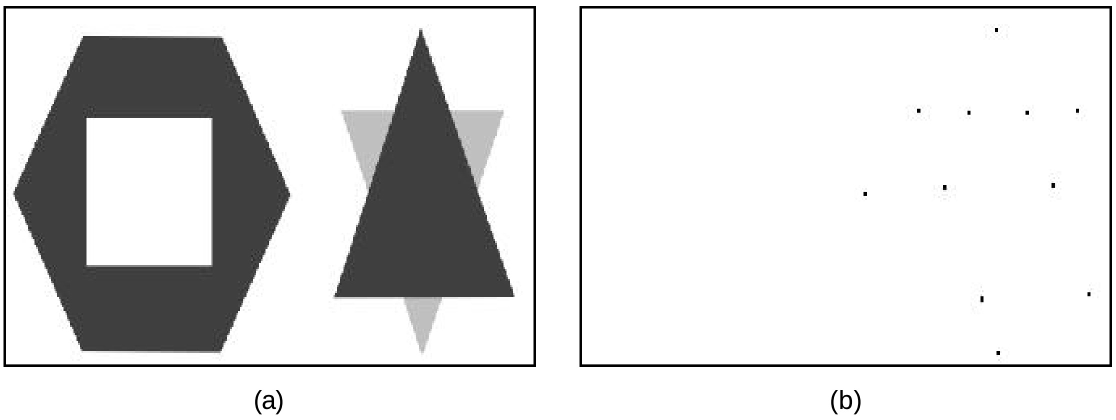

(a) (b)

Figure 61: A corner detector after Zero-1 concept: (a) Original image, (b) Corners.

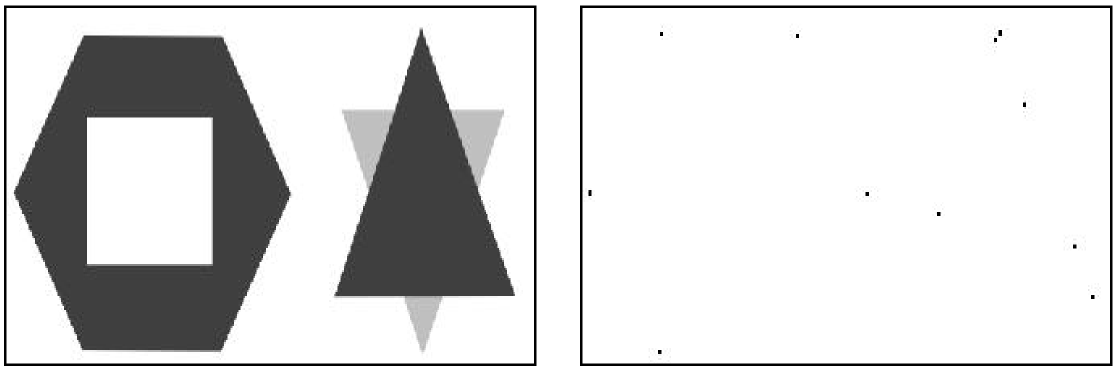

(a) (b)

Figure 62: A corner detector after Zero-2 concept: (a) Original image, (b) Corners.

The Kitchen-Rosenfeld corner detector was used to get the initial corner response for Figure 59 (a), Figure 60 (a), Figure 61 (a), and Figure 62 (a). The *Zero-1* and *Zero-2* concepts were applied to get the result shown in Figure 59 (b) and Figure 61 (b), and Figure 60 (b) and Figure 62 (b) respectively. For the *Zero-1* concept, the Mean 3 x 3 filter was applied prior to the corner detection. The results in Figure 61 (b) and Figure 62 (b) were obtained after applying a threshold of 13 and 71 respectively (Figure 87).

## 5.8. A corner detector based on the concept of non-maxima suppression

*Non-maxima suppression* revolves around the concept of deleting pixels that are not the maximum in a certain region (neighbourhood or direction). Whenever a neighbourhood is considered, all the pixels are compared with one another. If the centre pixel (the one that is under consideration) is greater than or equal to all the pixels in the neighbourhood, then that pixel is retained. If this condition fails, then that pixel is deleted. In the end, only the pixels that are greater than or equal to the values of the pixels in the neighbourhood are retained.

If non-maxima suppression, as explained above, is applied after a corner detector, sharp pixels (pulses) are shown in the output. This output is considered ideal for a corner detector. Let us name this concept *Maxima-1*.

In addition, non-maxima suppression was manipulated with a different concept where dilation was applied to the output of a corner detector. When applying gray-scale dilation to every neighbourhood in the image, the centre pixel of each neighbourhood is assigned the maximum value of the neighbourhood. Gray-scale dilation ensures that the value of a true corner remains unchanged. After dilation, its outcome is compared with the initial values of the corner response (Figure 63).

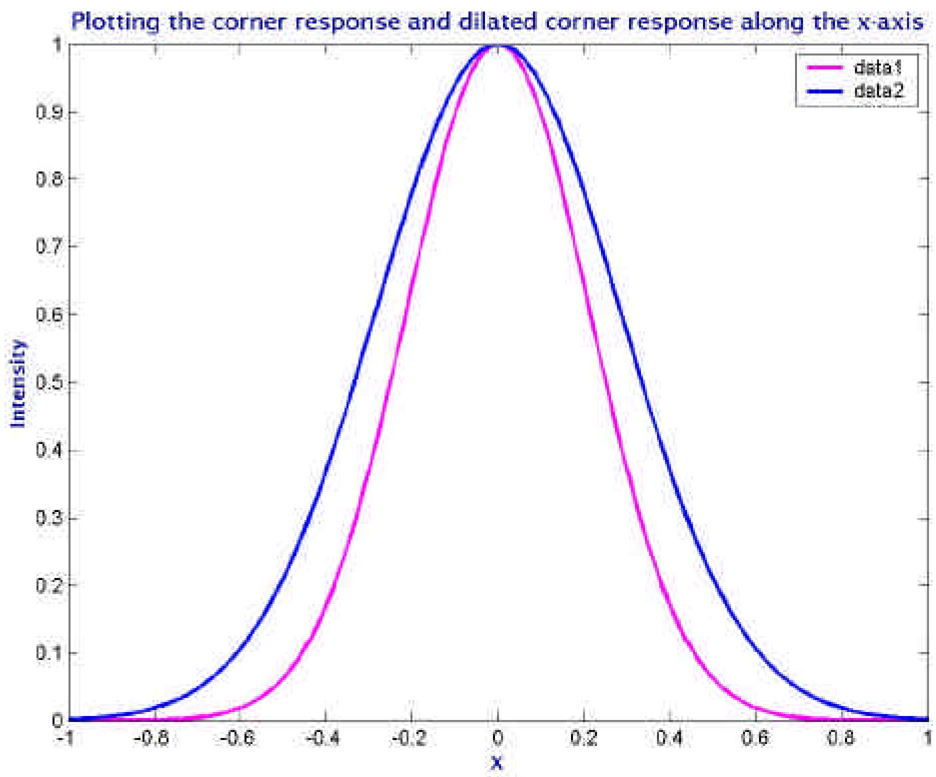


Figure 63: The corner response and the dilated form of the corner response. Here, data1 corresponds to the initial corner and data2 corresponds to the dilated form of the initial corner response.

If the corner value in the initial response is not the greatest in that neighbourhood, then it is not a true corner. On comparison, all the pixels having the same values in both the corner response and the output of the dilation function and at the same position are identified as corners and are projected in the output image. This can be shown in Figure 64. Let us name this concept *Maxima-2*.

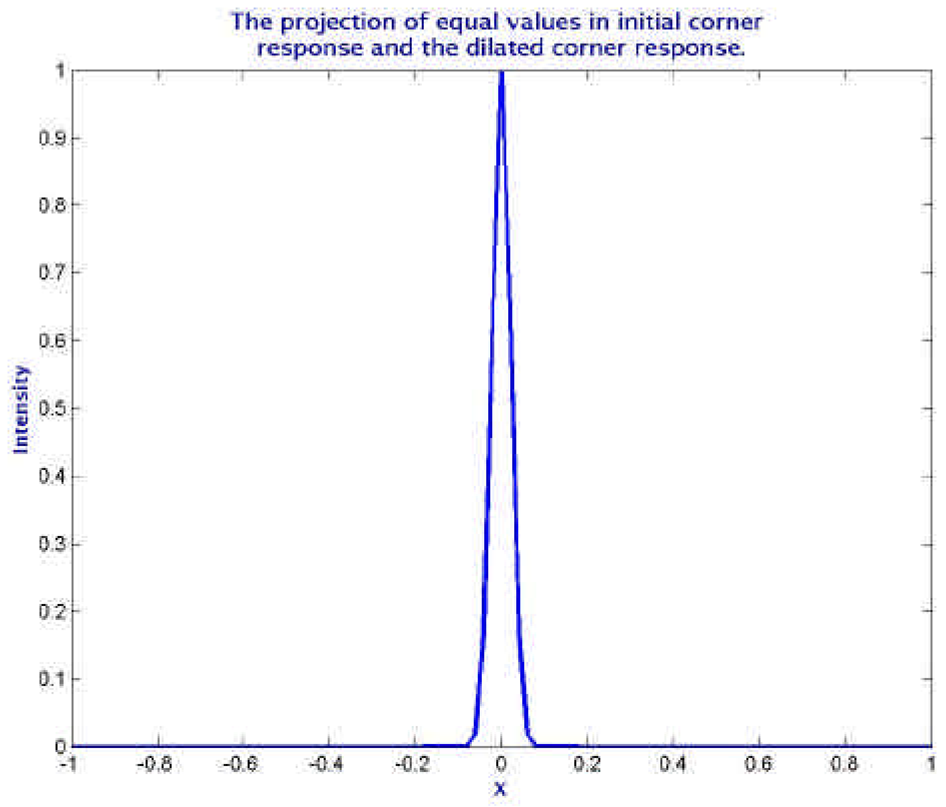


Figure 64: The result of the projected values that are equal in corner response and dilated form of the corner response from Figure 63.

These concepts might not work accurately on real-time images due to factors like noise, resolution of the image, and complexity of the image objects.

*Maxima-1* compares the intensity values of the pixels in the neighbourhood, resulting in projection of one value for every neighbourhood. It was observed that when a corner detector is applied after an edge detector *Maxima-1* shows parts of the edges as corners.

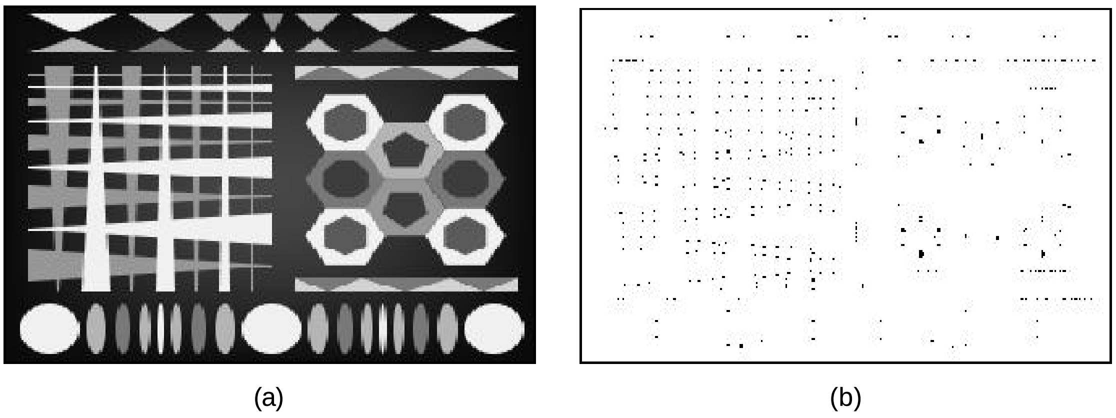


Figure 65: A corner detector after Maxima-1 concept: (a) Original image, (b) Corners.

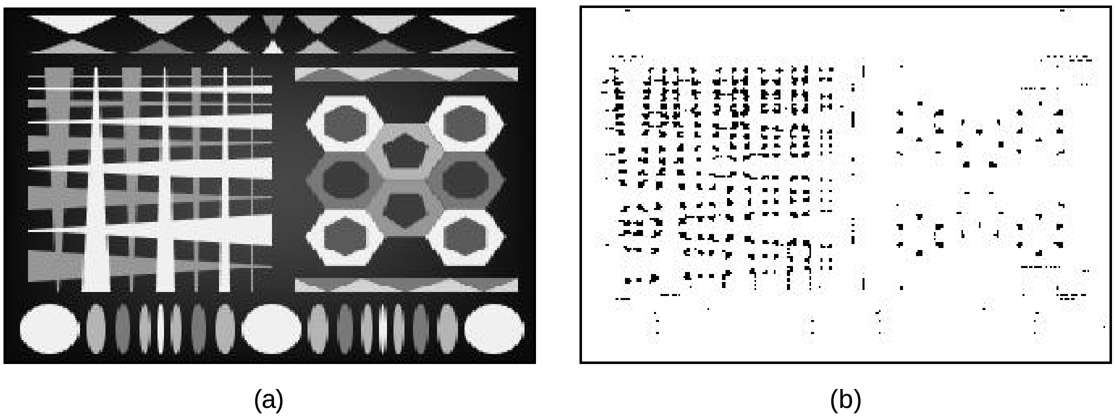


Figure 66: A corner detector after Maxima-2 concept: (a) Original image, (b) Corners.

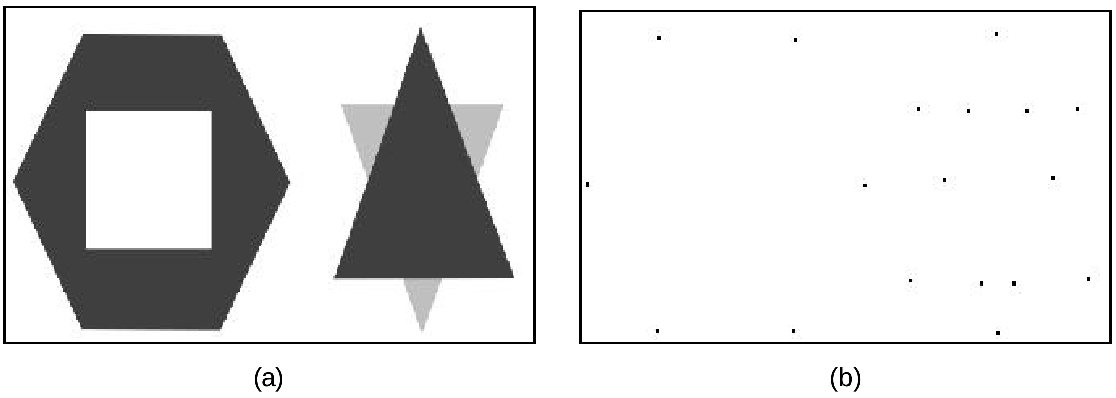


Figure 67: A corner detector after Maxima-1 concept: (a) Original image, (b) Corners.

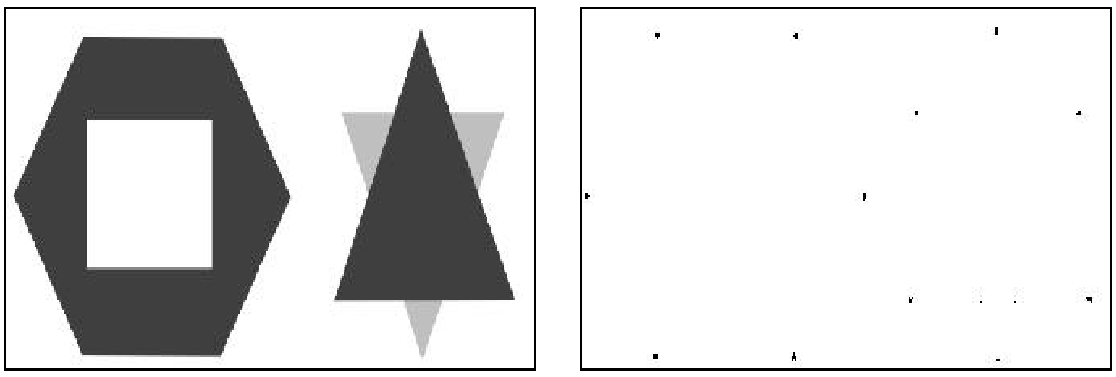

(a) (b)

Figure 68: A corner detector after Maxima-2 concept: (a) Original image, (b) Corners.

The Kitchen-Rosenfeld corner detector was used to get the initial corner response for images in Figure 65 (a), Figure 66 (a), Figure 67 (a), and Figure 68 (a). The *Maxima-1* and *Maxima-2* concepts were applied to get the results shown in Figure 65 (b) and Figure 67 (b), and Figure 66 (b) and Figure 68 (b) respectively. Prior to applying all these, Mean 3 x 3 filter was applied. The final results shown in Figure 67 (b), and Figure 68 (b) were obtained after applying a threshold of 13 and 18 respectively (Figure 87).

## 5.9. A corner detector based on pre-defined templates

Unlike the detectors discussed previously, this detector identifies corners in a binary image. These detectors are used to detect black corners in white backgrounds and vice-versa.

If the image is gray-scale, it is first converted into a binary image by setting a threshold value. This converts all the values higher than the threshold parameter to white and anything lower to black. The corner detection is performed using eight corner detector templates (Figure 69). These eight corner operators are formed taking the first corner operator (a) and then rotating it seven times by $45^0$, resulting in eight operators. Each of this eight is applied to each pixel. If any pixel satisfies any of the templates shown in Figure 69, then it is considered a corner. Let us name this corner detector *Binary-1*.

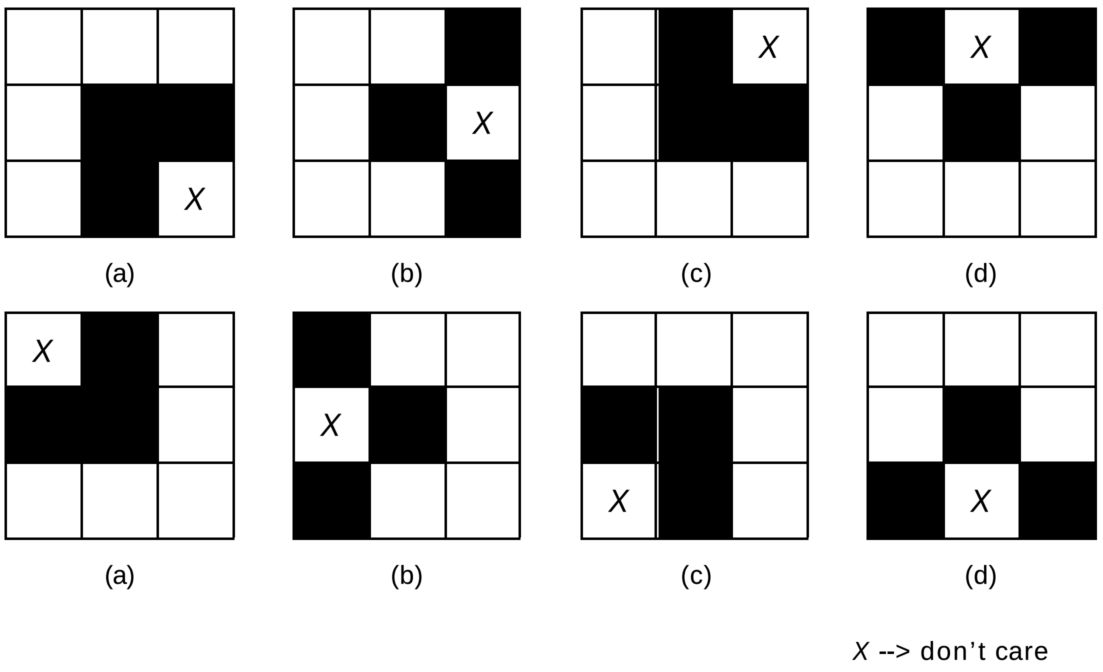


Figure 69: Templates for Simple binary corner detector (Binary-1) [47].

To try out this concept, experiments were performed on various images, gray-scale and binary with varying threshold values. It should be noted that, on converting a gray-scale image to binary, some pertinent data might be lost or distorted. Due to this, there is a high possibility of wrong points being identified as corners or true corners being missed altogether.

It was observed that with increase in the threshold value, the number of wrong corners detected also increased.

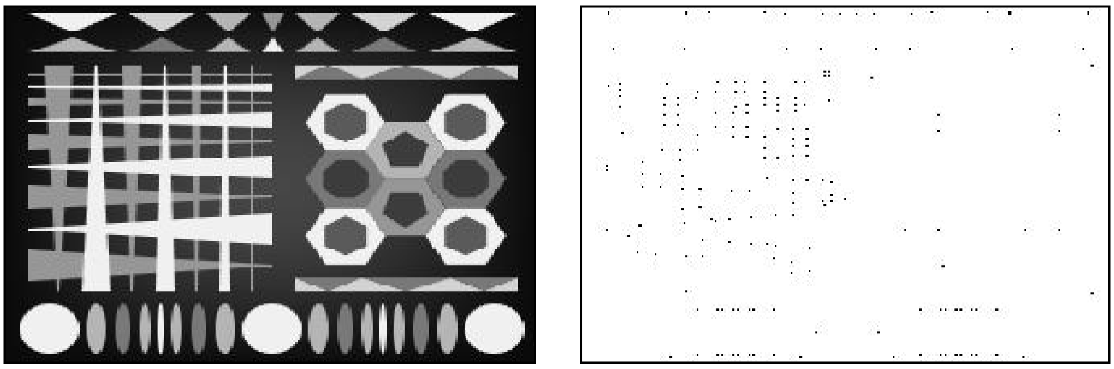

(a) (b)

Figure 70: Corner detector based on predefined templates (Figure 69, threshold value = 128): (a) Original Image, (b) Corners after converting the original image to binary image (threshold value = 128).

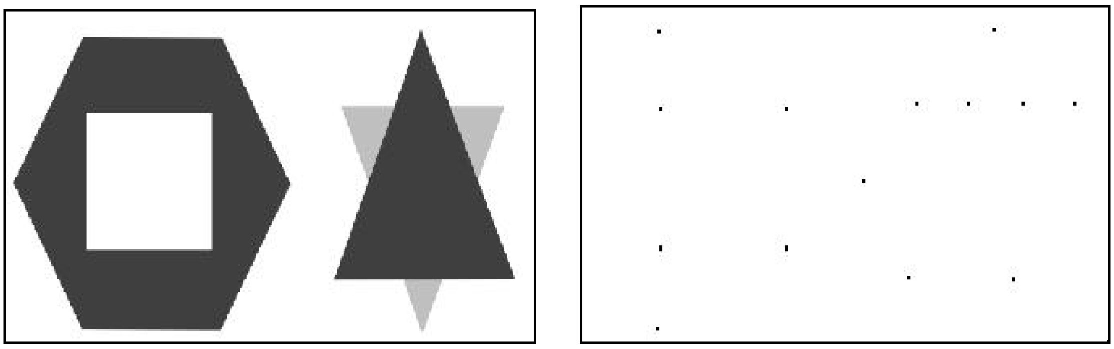

(a) (b)

Figure 71: Corner detector based on predefined templates (Figure 69, threshold value = 255): (a) Original Image, (b) Corners after converting the original image to binary image (threshold value = 255).

## 5.10. Comparison of corner detectors

Harris-Stephens corner detector showed the best results considering corner localization. Plessey corner detector showed better results than Kitchen-Rosenfeld. The results of Nobel corner detector were comparable to that of Plessey. In the Beaudet corner detector, apart from detecting a true corner pixel, even some surrounding pixels are detected as corners. This is because, the detector assigns the same intensity values to these pixels as that of the true corner pixel. This leads to a cluster of neighbouring pixels being labelled as corners. It was observed that where edges of distinct objects intersect, this detector identified the junction as multiple corners. Apart from Noble, it was observed that the corners obtained by Beaudet corner detector had high intensity values. This is because both detectors find corners using only differences and not dividing by any number or expression. The detectors derived from the zero-crossing concept (*Zero-1* and *Zero-2*) failed to show good results. However, the detectors derived from the concept of non-maxima suppression (*Maxima-1* and *Maxima-2*) gave good results. The results obtained from *Maxima-1* were of better quality than that of *Maxima-2*. Table 3 shows the results of applying the different corner detectors, discussed earlier in the chapter.

The following are some of the key parameters that were selected for quantitative measurement of performance of the corner detectors.

- Original speed: This is the maximum speed, measured in fps (frames per second), possible with no corner detectors applied to the image. This value remains same for every corner detectors as the same image was used for testing all detectors. For the image used in these tests, the original speed was at a maximum of 128 fps.

- Initial correct corners: This is the number of corners that are visible in the image. The total number of corners in the original image is 22.

- Final speed: This is the maximum speed with the particular corner detector applied on the image.

- Correct corners detected: This is the number of corners that are correctly identified by the detector.

- Multiple corners detected: This is the number of corners that are identified more than once by the detector.

- Extra corners detected (error positive): This is the number of pixels incorrectly identified as corners.

- Missed corners (error negative): This is the number of corners, which the detector failed to identify.

- Threshold: All the pixels having a value greater than the threshold were projected as corners. The results of corners overlayed on the test image Figure 47 (a) are shown in Figure 87. For *Binary-1* detector, threshold was applied to convert the gray-scale image into binary image.

- Parameters: For an optimal solution, a corner detector may depend on parameter(s), which need to be tuned at run-time.

- Other filters used: In a real image, there might be a significant change in the intensity values of the neighbouring pixels, which might cause them to be a misidentified as a corner. The reasons for this change can be surface texture, lighting effect, etc. Hence, before applying any corner detector, an averaging filter may be applied to the image.

|  | Kitchen Rosenfeld | Beaudet | Plessey | Noble | Harris Stephens | Zero-1 | Zero-2 | Maxima-1 | Maxima-2 | Binary-1 |
|---|---|---|---|---|---|---|---|---|---|---|
| Final Speed | 29.4 | 30.6 | 28.2 | 21.1 | 29.3 | 14.8 | 14.5 | 19.1 | 25.2 | 20.6 |
| Correct corners detected (22) | 14 (64%) | 20 (91%) | 18 (82%) | 19 (86%) | 22 (100%) | 11 (50%) | 8 (36%) | 18 (82%) | 14 (64%) | 14 (64%) |
| Multiple corners detected | 0 | 8 | 3 | 8 | 2 | 0 | 1 | 0 | 0 | 0 |
| Error positive | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| Error negative | 8 | 2 | 4 | 3 | 0 | 11 | 14 | 4 | 8 | 8 |
| Total Error (22) | 8 (36%) | 10 (45%) | 7 (32%) | 11 (50%) | 2 (9%) | 11 (50%) | 17 (77%) | 4 (18%) | 8 (36%) | 8 (36%) |
| Threshold | 18 | 255 | 43 | 255 | 217 | 13 | 71 | 13 | 18 | 255 |
| Parameters | - | - | - | 0.35[6] | 0.04[5] | - | - | - | - | - |
| Other Filters used | Mean 3 x 3 | Mean 5 x 5 | Mean 3 x 3 | Mean 3 x 3 | Mean 5 x 5 | Mean 3 x 3 | - | Mean 3 x 3 | Mean 3 x 3 | Mean 3 x 3 |

Table 3: Comparison of different corner detectors.

[5] - The value of k, from Equation 47, was tested at run-time.

[6] - The value of $\sigma$ , from Equation 45, was tested at run-time.

# 6. Thinning filters

In many image-processing tasks, processing on all points will not be required. In case of images that are mostly line-based, that is, text or line drawings, thinning the image may prove to be beneficial to further processing stages. This is because in these stages, the presence of lines is important and not their thickness. Hence, thinning plays a key role at the image pre-processing stage. "The method of removing redundant data from the lines and retaining the lines of required width is called *thinning* or *skeletonizing*" *[6]*. This chapter describes three methods that remove all unnecessary data but retain the important data required for processing.

Thinning methods are morphological operations applied to binary images, which in turn produce binary images. The original image may contain thick edges or lines that may make subsequent processing time-consuming. Thinning filters try to reduce the width of these edges or lines as much as possible, ideally to one pixel. Detecting and matching human fingerprints is one application that may benefit from thinning. This is because, in matching fingerprints, detecting the presence and shape of certain lines is more essential than their thickness. Another application where the same concept can be used is pattern matching or pattern recognition. Here, detecting certain lines that make the shape of a particular object or pattern is more crucial than the width of the lines lying on the outline. Thinning of extended objects is necessary in image analysis, where again the shape and pattern may be more important than the width of the object. In most cases, when thinning is applied the thickness of the original object is lost and cannot be traced back.

Ideally, a thinning procedure should follow some basic principles. These are refined from *[8]* as:

- The connectivity of the curve or the object should be preserved.

- Objects should remain unchanged in their shape, that is, a curve or an intersection of curves should remain in the same shape and orientation.
- After thinning, the skeleton of the object should lie approximately along the centre of the object.
- The skeleton of these objects should be as thin as possible.
- The thinning procedure should be as fast as possible, hence, should comprise as few iterations as possible.

## 6.1. Methods implemented

The thinning methods that were implemented and tested are *Stefanelli-Rosenfeld [5]*, *Lü-Wang [26]*, and *Hall-Guo [8]*. All these thinning methods were implemented as described in *[6]*.

Thinning can be achieved by directly comparing the pixels of a neighbourhood with their magnitudes *[6]*. An ideal thinning filter deletes a pixel if it considers that pixel not to be a part of the skeleton. A thinning filter operates on pixels of an image and its neighbourhood. A 3 x 3 neighbourhood can be explained as:

| | | |
|---|---|---|
| P1 | P2 | P3 |
| P8 | P | P4 |
| P7 | P6 | P5 |

Figure 72: The relation of the neighbouring pixels with the pixel under consideration.

Here, pixel P, marked in gray in Figure 72, is the pixel under consideration. In each of these filters, the value of pixel PN (1 <= N <= 8) is considered to be 1 if it is black in colour and 0 if it is white. The decision to delete a pixel is made by analysing different logical conditions in its 3 x 3 neighbourhood. These conditions are white to black patterns (A(P)), number of black pixels (B(P)), and connectivity number (C(P)).

A(P) stands for the number of '01' patterns in the neighbourhood. It calculates the number of white to black patterns in the neighbourhood of pixel P in a clockwise direction. This can be better understood with the help of Equation 48.

$$\begin{aligned} A(P) = {} & (\,!P1 \ \&\& \ P2) + (\,!P2 \ \&\& \ P3) \\ & + (\,!P3 \ \&\& \ P4) + (\,!P4 \ \&\& \ P5) \\ & + (\,!P5 \ \&\& \ P6) + (\,!P6 \ \&\& \ P7) \\ & + (\,!P7 \ \&\& \ P8) + (\,!P8 \ \&\& \ P1) \end{aligned}$$

Equation 48: The Computation of A(P).

B(P) stands for the number of black pixels in the neighbourhood. It can be calculated from Equation 49.

$$B(P) = P1 + P2 + P3 + P4 + P5 + P6 + P7 + P8$$

Equation 49: The Computation of B(P).

C(P) stands for the connectivity number which represents the number of white pixels directly above and below, and to the left and right. It does not consider diagonal pixels. For every white pixel (P2, P4, P6 and P8) it checks that at least one of the next two pixels (in clockwise direction) is black. This calculation is shown in Equation 50.

$$\begin{aligned} C(P) = {} & (\,!P2 \ \&\& \ (P3 \text{ OR } P4)) + (\,!P4 \ \&\& \ (P5 \text{ OR } P6)) \\ & + (\,!P6 \ \&\& \ (P7 \text{ OR } P8)) + (\,!P8 \ \&\& \ (P1 \text{ OR } P2)) \end{aligned}$$

Equation 50: The Computation of C(P).

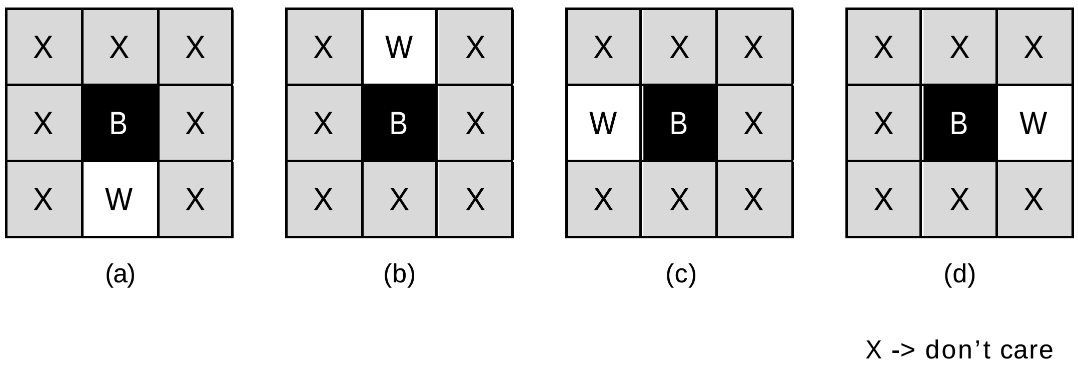
For example, in Figure 72, for pixel P2, the next two pixels are P3 and P4 and for pixel P4, the next two pixels are P5 and P6. Applying the concept, if P2 is white, and any of the next two or both pixels are black, then C(P) becomes one. The final value of C(P) depends on the calculation for the remaining three pixels, P4, P6, and P8.

Thus, the value of C(P) is always between 0 and 4, inclusive. C(P) is exactly 1 if and only if the pixel is a part of contour lines. Similarly, from Equation 48, the value of A(P) is between 0 (all pixels black or white) and 4 (alternate black and white), inclusive. From Equation 49, the value of B(P) is between 0 (all pixels white) and 8 (all pixels black), inclusive. Whenever the value of B(P) is one, the pixel P is an end-point of a line or an edge. However, when B(P) has a value of more than one, then it is difficult to say whether pixel P is an end-point.

## 6.2. Stefanelli-Rosenfeld thinning filter

In this method, [5], first the contour pixels (pixels lying on the boundary) are identified and then they are deleted. This process of deletion is called 'peeling'. This method continues until there are no contour pixels left, that is, we are left only with final pixels, which are part of the skeleton. For any pixel, there are two possibilities; it can be either a contour pixel or a final pixel (part of the skeleton).

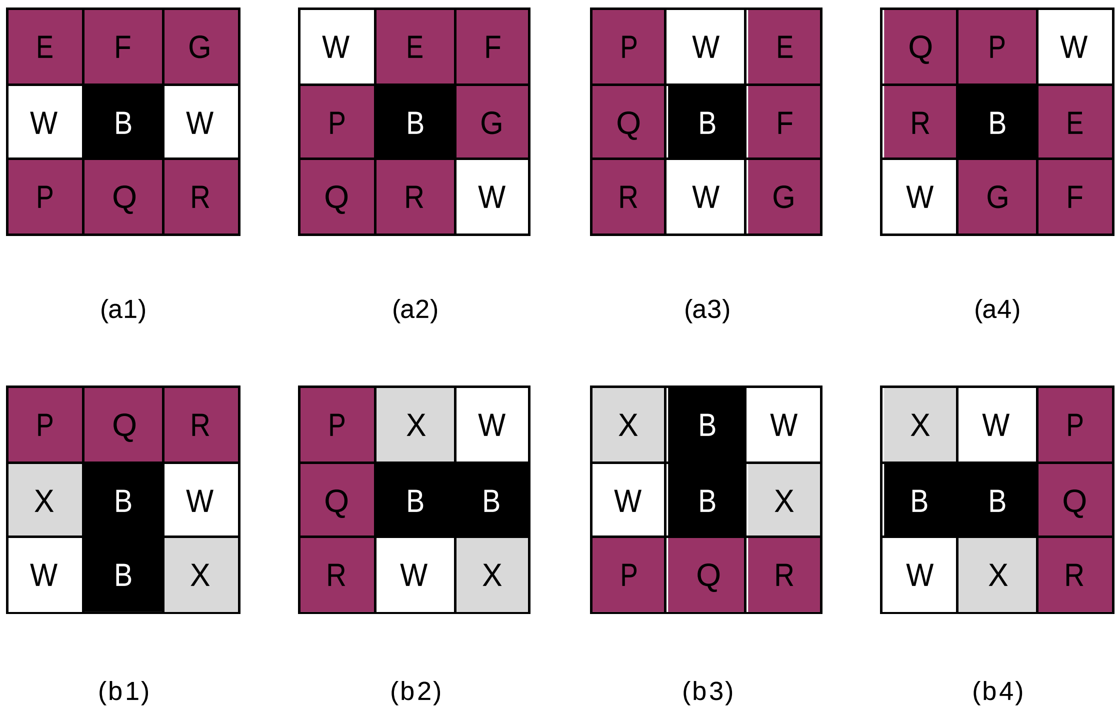
For a pixel to be considered as a contour pixel, it must satisfy any of the four conditions in Equation 51.



Equation 51: The condition for a pixel to be a contour pixel [5, 6].

In Equation 51, B stands for black, W stands for white and X stands for black or white. This notation is only valid for binary images. The process starts only after the operators detect the centre pixel as black (part of the object). Using the notation described in Equation 51 (a), the pixel P6 (Figure 72) is white. Similarly, in Equation 51 (b), (c), and (d), P2, P8 and P4 are white.

For a pixel to be considered as a final pixel, it must satisfy any one of the conditions (a1-a4) or any two of the conditions (b1-b4) depending upon the direction of the peeling process. These conditions are shown in Equation 52.



Equation 52: The different conditions for a pixel to be a final-pixel [5, 6].

In Equation 52, B stands for black; W stands for white. At least one of E, F and, G, and P, Q, and R should be black. X means that pixel can be either black or white.

For every pixel there are four sub-iterations (bottom, top, left, and right) running one after the other, checking for the contour pixels and final pixels. Table 4 shows the checking of these conditions in each of the four directions. These conditions ensure that no final pixel is accidentally deleted. Let us name this algorithm Stefan-1.

| Sub-iteration | Contour-pixel type | Contour-pixel condition (Equation 51) | Final-pixel condition (Equation 52) |
|---|---|---|---|
| 0 | Bottom | (a) | (b1), (b2) |
| 1 | Top | (b) | (b3), (b4) |
| 2 | Left | (c) | (b1), (b4) |
| 3 | Right | (d) | (b2), (b3) |

Table 4: Conditions for the calculation of redundant pixels [6].

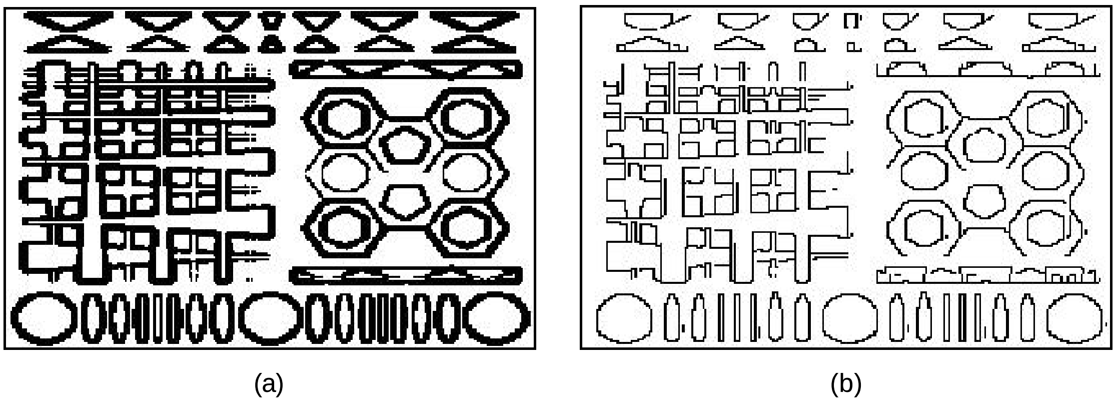


Figure 73: Stefanelli-Rosenfeld thinning filter: (a) Original image with binary edges, (b) Thinned image.

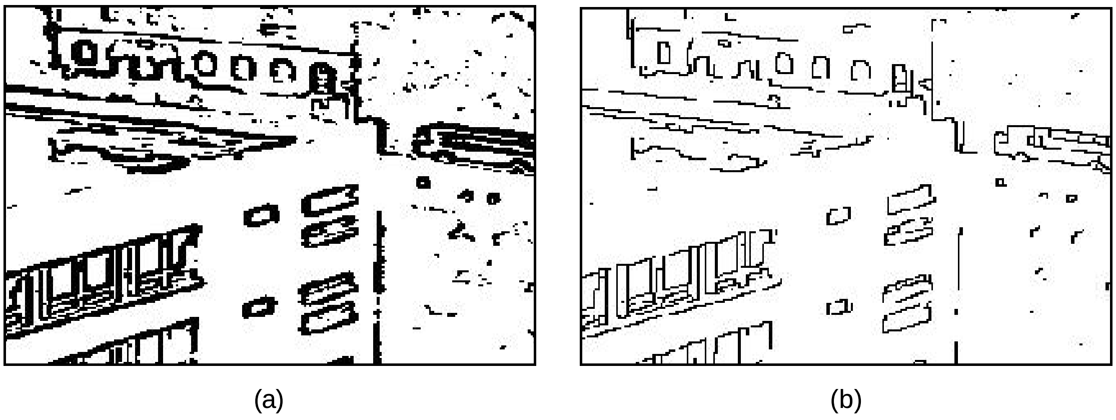


Figure 74: Stefanelli-Rosenfeld thinning filter: (a) Original image with binary edges, (b) Thinned image.

In addition to the conditions explained in *Stefan-1*, Stefanelli and Rosenfeld [5] proposed another set of conditions that are based on A(P), B(P) and C(P). These are shown in Equation 53.

$$(a)\ 2 <= B(P) <= 6$$

$$(b)\ A(P) = 1$$

$$(c)\ P2 \,.\, P4 \,.\, P8 = 0 \text{ OR } A(P2) \,!= 1$$

and

$$(d)\ P2 \,.\, P4 \,.\, P6 = 0 \text{ OR } A(P4) \,!= 1$$

Equation 53: Conditions for a pixel to be deleted [5].

The decision to delete the pixel is taken when all the conditions for *Stefan-1* and Equation 53 are satisfied. Let us name this algorithm as *Stefan-2.* The conditions listed in Equation 53 ensures connected skeletons which are not sensitive to contour noise [5].

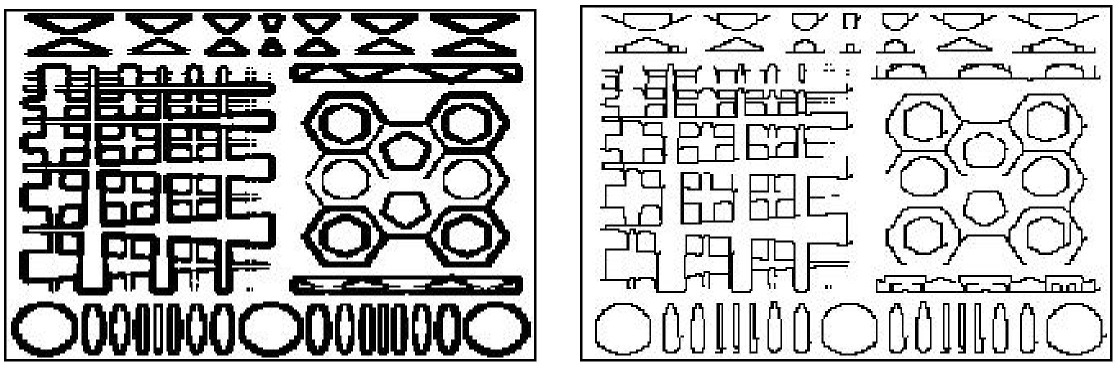

(a) (b)

Figure 75: Stefanelli-Rosenfeld thinning filter with factors in Equation 53: (a) Original image with binary edges, (b) Thinned image.

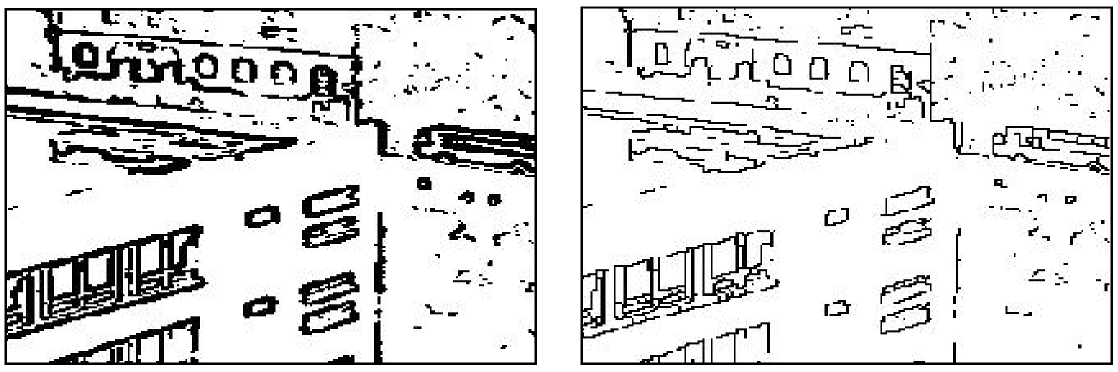

(a) (b)

Figure 76: Stefanelli-Rosenfeld thinning filter with factors in Equation 53: (a) Original image with binary edges, (b) Thinned image.

## 6.3. Lü-Wang thinning filter

Lü-Wang [26] is based on two sub-iterations and does not test whether a pixel is a final pixel but just decides whether it can be deleted or not, that is, whether it is a contour pixel. It uses different conditions for deciding whether to delete the pixel. These are shown in Equation 54.

*(a) 3 <= B(P) <= 6*

*(b) A(P) = 1*

AND

*(c1) P2 && P4 && P6 = FALSE OR P4 && P6 && P8 = FALSE*

OR

*(c2) P2 && P4 && P8 = FALSE OR P2 && P6 && P8 = FALSE*

Equation 54: Conditions for a pixel to be deleted [6, 26].

For a particular pixel to be deleted,

- On even sub iterations, *(a)* AND *(b)* AND *(c1)* should be TRUE.
- On odd sub iterations, *(a)* AND *(b)* AND *(c2)* should be TRUE.

Hence, if the pixel has to be deleted then it has to meet all the criteria. In Equation 54, Condition *(a)* retains the end-points of the lines. Condition *(b)* makes sure that no lines are broken when they pass through P. This is true because when a line passes through P, the number of `FALSE-TRUE (01, A(P))` will always be greater than one.

Conditions *(c1)* and *(c2)* can be rewritten as:

$$(c1)\ P4\ \&\&\ P6\ \&\&\ (P2\ ||\ P8) = FALSE$$
$$(c2)\ P2\ \&\&\ P8\ \&\&\ (P4\ ||\ P6) = FALSE$$

Equation 55: Alternative Representation of conditions c1 and c2 of Equation 54 [6].

Conditions *c1* and *c2* cause the top-right (northeast) and bottom-left (southwest) contour pixels to be deleted alternately.

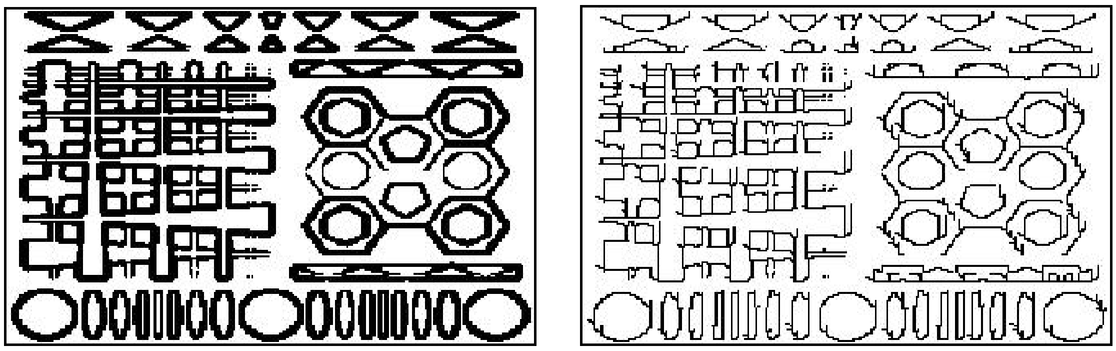

(a) (b)

Figure 77: Lü-Wang thinning filter: (a) Original image with binary edges, (b) Thinned image.

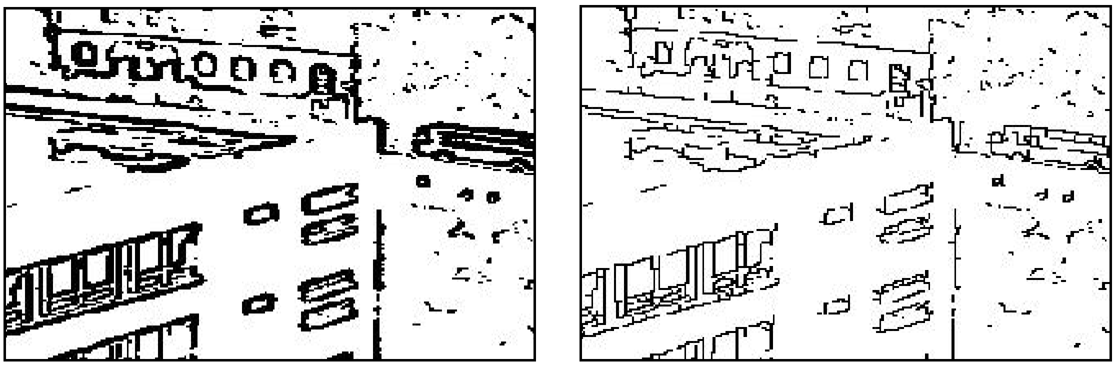

(a) (b)

Figure 78: Lü-Wang thinning filter: (a) Original image with binary edges, (b) Thinned image.

The disadvantage of this method is that the diagonal lines can produce a skeleton that is two pixels wide, that is, thicker than the ideal.

## 6.4. Hall-Guo thinning filter

*[8]* and *[7]* have a different way of splitting the sub-iterations. At the start of the process, this method marks the image like a chessboard (alternate black and white pixels). Two sub-iterations delete redundant pixels in alternating white and black sub-sections. This method consists of a single operator based on the conditions in Equation 56.

$$\begin{gathered}
(a)\ \ C(P) = 1 \\
(b)\ \ B(P) > 1 \\
\text{AND} \\
(c)\ \ (P1\ \&\&\ P3\ \&\&\ P5\ \&\&\ P7) \\
\ ||\ (P2\ \&\&\ P4\ \&\&\ P6\ \&\&\ P8) = FALSE
\end{gathered}$$

Equation 56: Conditions for a pixel to be deleted [6, 8].

In Equation 56, condition *(a)* avoids the possibility of deleting the pixel belonging to the centre of the object; condition *(b)* avoids deleting the end pixels of a line; and condition *(c)* makes sure that the edge pixels are deleted. If the diagonal pixels or the side pixels have all the values as ones, then the condition *(c)* fails. In that case, the pixel P is not deleted. Therefore, for the pixel P to meet the condition *(c)*, not all of the diagonal pixels or side pixels can be black at the same time. If this happens, then the pixel P is the intersecting pixel of two lines.

The advantage of this thinning procedure is that the diagonal lines are preserved. Let us name this algorithm as Hall-1.

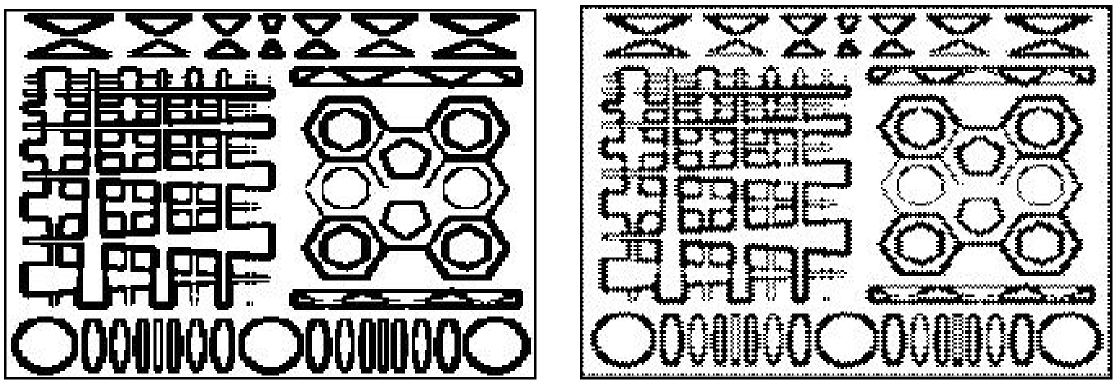

(a) (b)

Figure 79: Original Hall-Guo thinning filter (Hall-1): (a) Original image with binary edges, (b) Thinned image.

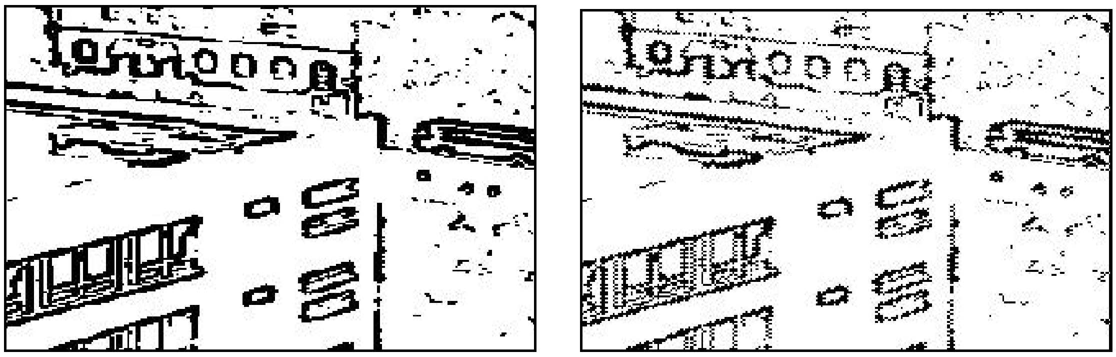

(a) (b)

Figure 80: Original Hall-Guo thinning filter (Hall-1): (a) Original image with binary edges, (b) Thinned image.

The following steps show how the chess labelling of the image was implemented.

```
For every pixel P(i,j)
     chesslabel = (i + j) % 2.
     :
     :
     Hall-Guo conditions (Equation 56)
     :
     :
     chesslabel = !chesslabel.
end for.
```

The results seen in Figure 79 and Figure 80, inspired the modification of the standard Hall-Guo method to obtain better results. The factor 'chesslabel' was used but in a different way. Instead of considering the value for 'chesslabel' based on the position of the pixel, it was always considered one. In the second iteration, the value was changed to zero. However, the factors affecting thinning, as stated in Equation 56, were not modified. Let us name this algorithm as Hall-2.

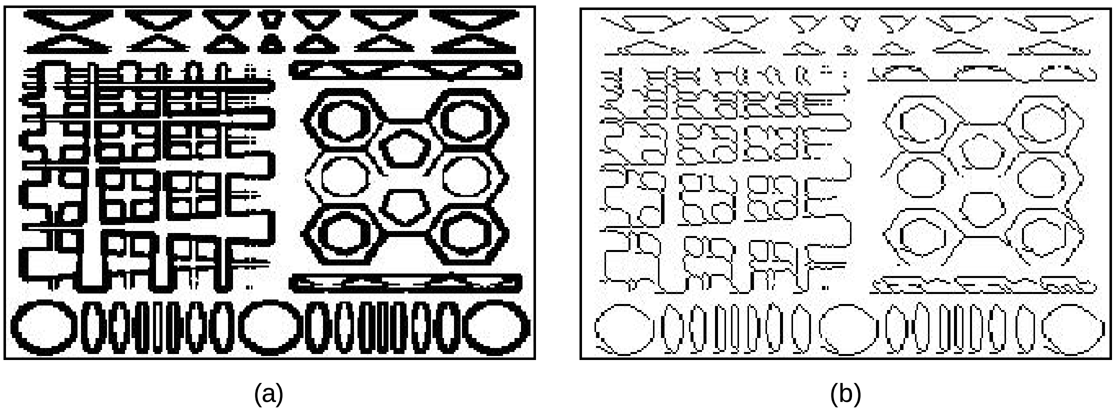


Figure 81: Modified Hall-Guo thinning filter (Hall-2): (a) Original image with binary edges, (b) Thinned image.

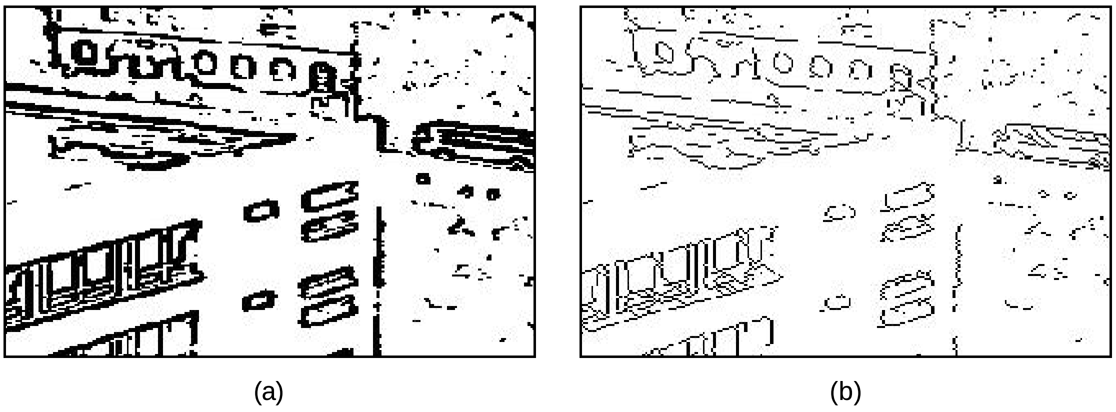


Figure 82: Modified Hall-Guo thinning filter (Hall-2): (a) Original image with binary edges, (b) Thinned image.

As can be seen from the results of Hall-2 in Figure 81 and Figure 82 and Hall-1 in Figure 79 and Figure 80, Hall-2 produces better results.

## 6.5. Comparison of thinning filters

The performance of the modified Hall-Guo filter (Hall-2) was observed to be better than Hall-1. For the Stefanelli-Rosenfeld thinning filter, *Stefan-2*, was observed to be better than *Stefan-1*. However, *Stefan-1* was faster than *Stefan-2*. Considering only the quality of the output, the filter proposed by Lü-Wang showed the best results. Table 5 shows the results of applying the different thinning filters, discussed earlier in the chapter. The following are some of the key parameters that were selected for measurement of performance of the thinning detectors.

- Original speed: This is the maximum speed, measured in fps (frames per second), possible with no thinning filters applied to the image. This value remains same for every thinning filter as the same image (Figure 73 (a)) was used for testing all detectors. For the image used in these tests, the original speed was at a maximum of 222 fps.

- Other filters and Threshold: Before any of the thinning filters were applied, a smoothing filter was used to smooth the image and edges of the object were found using the Sobel edge detector. Then, all the pixels having a value greater than the threshold (51) were projected as edges. The output of the Sobel edge detector is white edges on black background. This was negated, as black edges are needed for the thinning filters.

- Final speed: This is the maximum speed with a thinning filter applied on the result image containing only edges.

|  | Final Speed |
|---|---|
| Stefan-1 | 10.2 |
| Stefan-2 | 9.8 |
| Lü-Wang | 19.1 |
| Hall-1 | 23.5 |
| Hall-2 | 24.0 |

Table 5: Comparison of different thinning filters.

# 7. Correlation-based optic flow computation

*Optic flow* is a vector field that shows the direction and magnitude of intensity changes of a pixel in an image. It can be used for different purposes like finding correspondence between two images, measuring tilt angle of moving objects, and obstacle avoidance in robot navigation. The definition of obstacle, as given in [48], is "*any region in space where a vehicle should not or cannot traverse, such as protrusions, depressions or steep terrain.*" These obstacles can be static or dynamic. Like humans, a robot can avoid an obstacle if its location, shape, size, and distance to it are known. The degree of change of the size of the object is inversely proportional to the distance of that object from the robot. A human eye is an example that illustrates this concept. When a human eye moves closer to an object, its size appears to increase. Nearer objects would appear to change size relatively more compared to objects further away. Hence, this change in the size of the object can assist in determining the distance to that object. Thus, distance can be calculated by computing the change in the intensity of pixels of an object in successive images. These changes are calculated using various techniques like gradient-based optic flow, correlation-based optic flow, and spatiotemporal-based optic flow.

"*Correlation is the degree to which two or more quantities are linearly associated*" [49]. In optic flow computation, correlation is used to find similarities between two successive images of a sequence. In essence, the displacement of every pixel in the (*i+1*)th image is calculated with respect to its position in the *i* th image.

## 7.1. The working of standard correlation-based optic flow

Correlation-based techniques typically use the assumption of conservation of the local intensity distribution [11]. It works on two successive images of a given scene. Each pixel is identified by its own and the neighbouring pixels' intensities. To compute the optical flow vector for each pixel of the *i* th image, that particular pixel is traced in the (*i+1)*th image. Each pixel is traced by using the *sum of the squared differences* (*SSD*) function, as formulated in Equation 57.

$$M(x+dx, y+dy) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} (I_1(x+i, y+j) - I_2(x+dx+i, y+dy+j))^2$$
$$-N \leq dx, dy \leq N$$

Equation 57: The calculation for Correlation-based optic flow.

| | |
|---|---|
| p | pixel under consideration |
| M | matching value for each pixel |
| Least value of M | best match for that pixel |
| $I_1$ | first image |
| $I_2$ | second image |
| (x,y) | position of the pixel in the image |
| n | range of correlation window on either side |
| i | vertical range of correlation window |
| j | horizontal range of correlation window |
| (dx,dy) | position of the pixel in the search window with respect to original position (x,y) |
| N | range of search window on either side |

Equation 57 can be explained with the help of Figure 83.

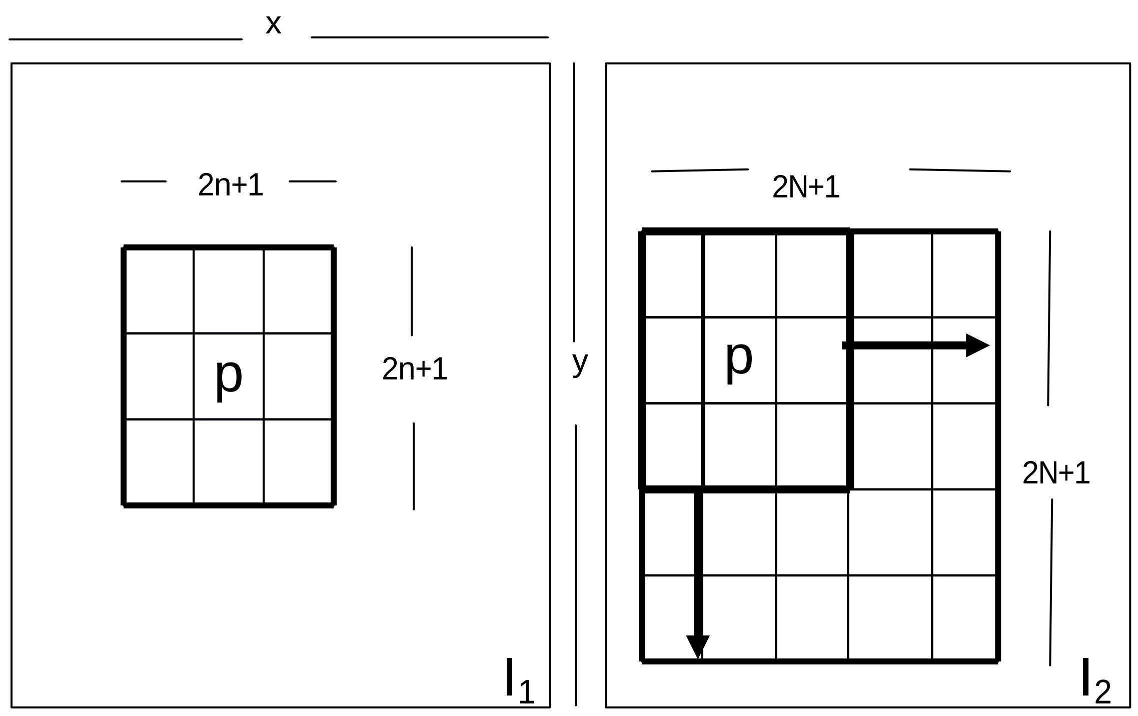


Figure 83: Two images $I_1$ and $I_2$ with pixel p and its neighbouring pixels, along with the correlation window of size (2n+1) x (2n+1) in $I_1$ and $I_2$, and search window of size (2N+1) x (2N+1) in $I_2$.

This tracing technique involves identifying pixel 'p' of image $I_1$ in image $I_2$. Now, for every pixel (2N+1) x (2N+1) candidate pixels are possible. The region covered by the candidate pixels is called the '*search window*'. Candidate pixels are determined by a physical constraint on the maximum displacement that can take place between two consecutive images in an image sequence, for that pixel. Each of these candidate pixels will have some match measure for the pixel under consideration. Match measure is a correlation between a small '*correlation window'* around the pixel under consideration in $I_1$ and a corresponding window around the candidate pixels in $I_2$. In this approach, the weighted sum of squared differences between the intensities at the corresponding pixels in the two windows acts as a match measure. Out of all the candidate pixels, the one having the least match measure is considered a '*best match'*.

If the best match is found uniquely, the exact displacement vector is immediately known. This is true for points in the image that are sufficiently distinct, such as corners. However, in case of the areas of strongly oriented intensity gradients, such as edges, several points have values similar to the match measure. It is not possible to disambiguate them. Thus, they suffer from what is technically known as the ‘*aperture problem*’. Corner points do not face this problem.

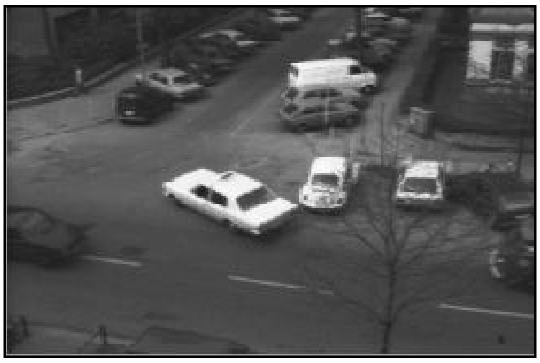
The following algorithm is a simplified description for the correlation-based optic flow.

```
1 for every pixel p in image $I_1$ at ‘spacing’
2    calculate a neighbourhood of size (2n+1) * (2n+1).

3    compute a smallest value in the neighbourhood for
     the term M(x,y) in Equation 57 by taking dx=0 and
     dy=0.

4    for every pixel in ‘search window’ of size (2N+1) * 9
     (2N+1) in image $I_2$

5         calculate M(x+dx,y+dy) from Equation 57.

6         if (M(x+dx,y+dy) of $I_2$ < M(x,y) of $I_1$)

7              M(x,y) = M(x+dx,y+dy).
8              xoffset = dx.
9              yoffset = dy.

10        end if

11   end for

12   draw vector from (x,y) to (x+xoffset,y+yoffset)

13 end for
```

Algorithm 1: Correlation-based optic flow computation

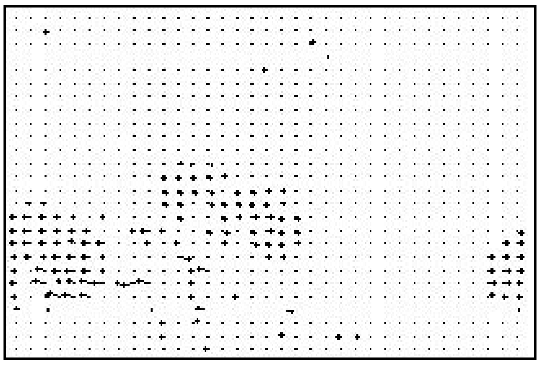
In Algorithm 1, the term ‘*spacing*’ gives the positions of the pixels to be considered for correlation. This is because it is impractical to perform correlation for every pixel in an image, as this is an expensive process in time and space. The term '*search window*' means the range of possible positions for pixel 'p' of $I_1$ in $I_2$.

Figure 84 shows the results of correlation-based optic flow. Here the mechanism of SSDs is used to compute the '*best match*'. Figure 84 (a) shows the initial test image at frame 1. Figure 84 (b), (d), and (f) are the original test image at frames 2, 5, and 13 respectively. Figure 84 (c), (e), and (g) are the results on applying the above discussed method.



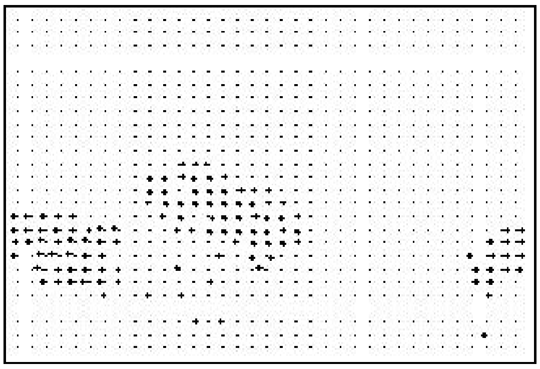
(a)

(b)

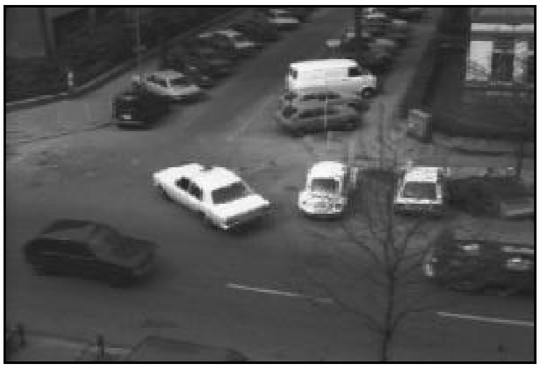


(c)

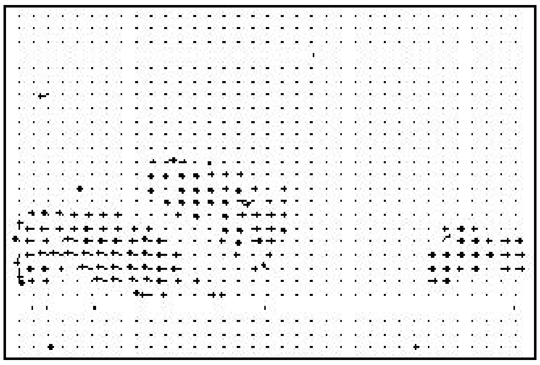
(d)



(e)



(f)



(g)

Figure 84: Optic flow computation using SSD: (a) The initial image at frame 1. (b), (d), and (f) are the original images at frame 2, 5 and 13 respectively. (c), (e), and (g) show the corresponding optic flow (SSD - Equation 57) results for frame 2, 5, and 13 respectively.

## 7.2. An alternative to standard correlation-based optic flow

Instead of taking the sum of the squared differences (SSD), an alternative is to take the sum of the absolute values of the differences. Let us name this alternative SAD. Equation 57 can thus be re-written as in Equation 58.

$$M(x+dx, y+dy) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} \left| I_1(x+i, y+j) - I_2(x+dx+i, y+dy+j) \right|$$

$$-N \leq dx, dy \leq N$$

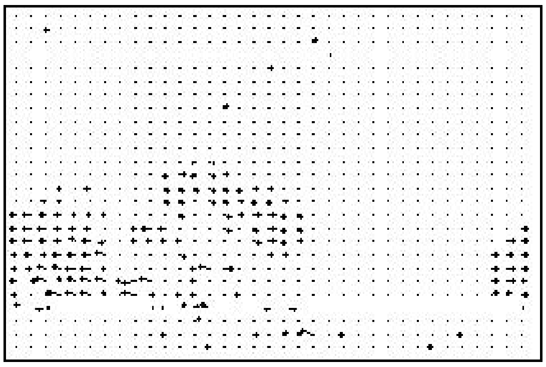
Equation 58: An alternative to standard Correlation-based optic flow.

Instead of Equation 57, Algorithm 1 now uses Equation 58 to perform the calculations to find the best match 'M'.

Figure 85 shows the results of correlation-based optic flow. Here the mechanism of SSD is used to compute the '*best match*'. Figure 85 (a) shows the initial test image at frame 1. Figure 85 (b), (d), and (f) are the original test image at frames 2, 5, and 13 respectively. Figure 85 (c), (e), and (g) are the results on applying the above discussed method.

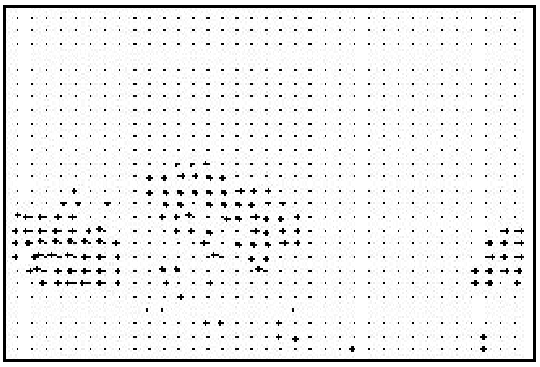
(a)

(b)

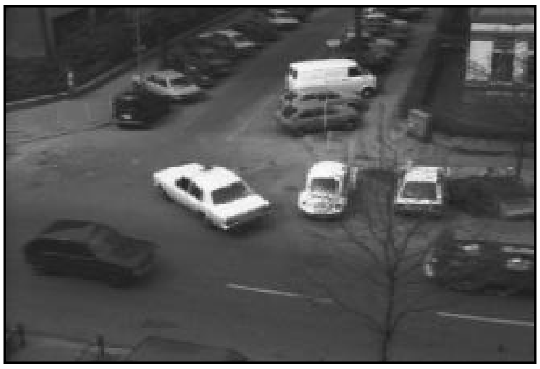


(c)

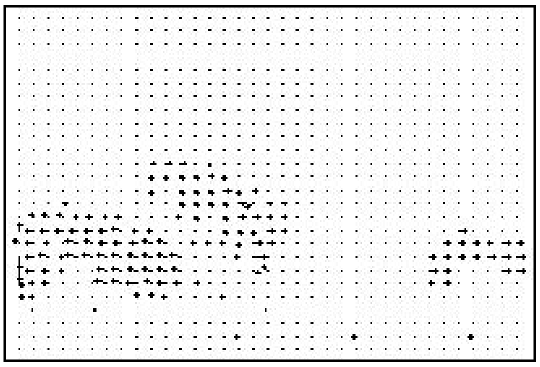
(d)



(e)



(f)



(g)

Figure 85: Optic flow computation using SAD. (a) The initial image at frame 1. (b), (d), and (f) are the original images at frame 2, 5 and 13 respectively. (c), (e), and (g) show the corresponding optic flow results (SAD - Equation 58) for frame 2, 5, and 13 respectively.

## 7.3. Comparison of optic flow algorithms

The performance of SSD based optic flow was observed to be better than SAD based optic flow. In quality, SSD had better results and fewer errors in the output image. Table 6 shows the results of applying the different parameters for each of SSD and SAD based optic flow computations, discussed earlier in the chapter.

The following are some of the key parameters that were selected for quantitative measurement of perfo rmance of SSD and SAD optic flow calculations.

- Original speed: This is the maximum speed, measured in fps (frames per second), possible with no optic flow method applied to the image. This value remains same for both the optic flow methods as the same image (Figure 84 (a)) was used for testing the methods. For the image used in these tests, the original speed was at a maximum of 43 fps.

- Final speed: This is the maximum speed with an optic flow method applied on the image.

- Spacing: Every 'spacing' pixel was considered for correlation.

- Search window: The search area in Image 2 of size (2N+1) x (2N+1) used to find the position of the pixel displaced. This is constant for every pixel of Image 1 for finding a corresponding match in Image 2.

- Correlation window: The area considered for correlation in Image 1 and Image 2. This is constant of size (2n+1) x (2n+1) for every comparison made between Image 1 and Image 2.

|  | Final Speed | Spacing | Search window (2N+1) x (2N+1) | Correlation window (2n+1) x (2n+1) |
|---|---|---|---|---|
| SSD | 0.6 | 6 | 13 x 13 | 13 x 13 |
|  | 1.1 | 7 | 11 x 11 | 13 x 13 |
|  | 2.1 | 7 | 9 x 9 | 11 x 11 |
|  | 2.1 | 8 | 11 x 11 | 9 x 9 |
|  | 1.5 | 8 | 11 x 11 | 11 x 11 |
| SAD | 0.5 | 6 | 13 x 13 | 13 x 13 |
|  | 0.9 | 7 | 11 x 11 | 13 x 13 |
|  | 1.8 | 7 | 9 x 9 | 11 x 11 |
|  | 1.8 | 8 | 11 x 11 | 9 x 9 |
|  | 1.3 | 8 | 11 x 11 | 11 x 11 |

Table 6: The parameters and their corresponding values used for SSD (Equation 57) and SAD (Equation 58) optic flow calculations.

# 8. Conclusions and discussions

We have implemented and analysed several image processing algorithms related to edge detection, corner detection and thinning. Apart from these techniques, smoothing filters were also studied. In addition, correlation-based optic flow was also studied and implemented. All the experiments were performed on ImprovQT version 5.1 (Figure 1), which performs real-time image processing similar to that done by robots.

We observed that the Mean filter could perform blurring but could not suppress salt and pepper, and impulse noises. The Gaussian smoothing filter gave better blurring results compared to the Mean filter, but it also failed in reducing salt and pepper, and impulse noises. Unlike the previous two, Median filter was able to remove these noises. Unlike Gaussian, the Mean filter uniformly assigns weights to the pixels. Hence, the Mean filter was faster compared to the Gaussian, when window size was kept constant. The Median filter was slower compared to the Mean and the Gaussian due to the extra processing required for sorting. It can be concluded that different smoothing filters are applicable in different circumstances.

In the case of edge detection, the Sobel and Kirsch edge detectors showed the best results. Compared to the Sobel edge detector, the Kirsch edge detector shows thicker edges. This is because Kirsch involves a larger number of pixels in either direction and assigns higher weights. However, Sobel is faster compared to Kirsch; hence, Sobel is the most widely used edge detector. Apart from these two, Laplace, Laplacian of Gaussian (LoG), Difference of Boxes (DoB), Robert's, Prewitt and Robinson edge detectors were implemented. Laplace and Robert's showed broken edges at some junctions. LoG was computationally the most expensive and the computation time increased with increase in standard deviation '$s$'. However, the best results for LoG were obtained with $s$ = 1.5. The results given by DoB improved after selecting mean filters with higher odd-sized dimensions. The best results were shown by using Mean 5 x 5 and Mean 3 x 3 filters. Prewitt and Robinson detectors gave average results.

The Harris-Stephens corner detector had the best results with speed (29.3 fps), accuracy (100%), and error-rate (9%). It had the best localization compared to other popular corner detectors, namely Plessey, Kitchen-Rosenfeld, and Beaudet. Plessey and Kitchen-Rosenfeld showed the nearly the same error-rate (32% and 36% respectively) but Plessey had better accuracy (82%) compared to Kitchen-Rosenfeld (64%). Noble showed nearly the same accuracy (86%) as Plessey but was slower with speed 21.1 fps. Beaudet detected more corners (91%) but with error-rate (32%). In this detector, apart from detecting a true corner pixel, even some surrounding pixels are detected as corners. This is because, the detector assigns the same intensity values to these pixels as that of the true corner pixel. This leads to a cluster of neighbouring pixels being labelled as corners. It was observed that where edges of distinct objects intersect, this detector identified the junction as multiple corners. In addition to these, the *Zero-1* and *Zero-2* corner detectors, based on the zero-crossing concept, were implemented. Both these detectors failed to show good results, with accuracy (50%) and (36%) respectively. *Zero-1* outperformed *Zero-2* because of the fact that the second order derivative of a corner response is not always symmetric along x and y directions. Apart from these, corner detectors based on non-maxima suppression were implemented. *Maxima-1* showed good results with accuracy (82%) at the cost of the error-rate (18%). On the other hand, *Maxima-2* had an accuracy of 64% with the error rate of 50%. On selecting a high value for threshold, *Binary-1* was able to detect corners with accuracy 64%.

Among the thinning filters discussed, the original Hall-Guo (*Hall-1*) and the modified filter (*Hall-2*) had faster processing speed compared to Stefanelli-Rosenfeld (*Stefan-1* and *Stefan-2*) and Lü-Wang. The performance of modified Hall-Guo filter (*Hall-2*) was observed to be better in terms of connectivity, speed, and accuracy than the original Hall-Guo filter (*Hall-1*). In addition, the performance of the Stefanelli-Rosenfeld thinning filter *Stefan-2*, was observed to be better than the *Stefan-1*. However, taking quality into consideration, the filter proposed by Lü-Wang showed the best results.

A basic form of correlation-based optic flow was implemented to study motion analysis. It was experimented with across a varying range of various parameters like 'spacing', 'search window', and 'correlation window'. This was done in order to find the best combination that gave reasonably accurate flow vectors at optimal speed. Two techniques for measuring match strength (SSD and SAD) were implemented. SSD was observed to be faster and more accurate compared to SAD. For SSD, the following parameter settings '*spacing*' of 8, '*search window*' of 11 x 11 and '*correlation window*' of 11 x 11 showed the best results at a speed of 1.5 fps. For the same parameter settings, SAD gave results at a speed of 1.3 fps.

Studying different image processing algorithms is necessary for selecting one that is best suited for an application. Factors to consider in selecting an algorithm are like accuracy, speed, and quality of its results in comparison to other algorithm of the same family. This thesis aims to compare these factors for several image processing techniques. It is worth further investigating the re-interpretation of standard corner detectors (*Zero-1*, *Zero-2*, and *Maxima-2*) and thinning filters (*Hall-2*) as described in this thesis. As can be seen, the results produced by these modified techniques were promising and could prove to be beneficial upon refinement.

# References

[1] "Chapter II, Digital Learning Center Glossary of Terms: Section I - Glossary," in Kodak - Digital Learning Centre.
http://www.kodak.com/US/en/digital/dlc/book4/chapter2/glossaryI.shtml

[2] E. W. Weisstein, "Convolution," in MathWorld--A Wolfram Web Resource.
http://mathworld.wolfram.com/Convolution.html

[3] T. Bräunl, "Improv - Image Processing for Robot Vision."
http://www.robotics.ee.uwa.edu.au/improv/index.html

[4] R. Gonzalez and R. Woods, Digital Image Processing: Addison-Wesley Publishing Company, 1992.

[5] R. Stefanelli and A. Rosenfeld, "Some Parallel Thinning Algorithms for Digital Pictures," Journal of the Association for Computing Machinery, vol. 18, pp. 255-264, 1971.

[6] T. Bräunl, S. Freyer, W. Raph, and M. Reinhardt, Parallel Image Processing: Springer-Verlag Berlin Heidelberg New York, 2001.

[7] R. W. Hall, "Fast Parallel Thinning Algorithms: Parallel Speed and Connectivity Preservation," Communications of the Association for Computing Machinery, vol. 32, pp. 124-131, 1989.

[8] Z. Guo and R. W. Hall, "Parallel Thinning with Two-Subiteration Algorithms," Communications of the Association for Computing Machinery, vol. 32, pp. 359-373, 1989.

[9] E. Trucco and A. Verri, Introductory Techniques for 3-D Computer Vision: Prentice Hall, 1998.

[10] T. Camus, "Real-time Optical flow," in Department of Computer Science: Brown University, 1994.

[11] A. Singh, Optical Flow Computation: A Unified Perspective: IEEE Computer Society Press, 1991.

[12] R. Deriche and G. Giraudon, "A computational approach for corner and vertex detection," The International Journal of Computer Vision, vol. 10, pp. 101-124, 1993.

[13] P. R. Beaudet, "Rotational Invariant Image Operators," International Conference on Pattern Recognition, pp. 579-583, 1978.

[14] L. Kitchen and A. Rosenfeld, "Gray-level corner detection," Pattern Recognition Letters, vol. 1, pp. 95-102, 1982.

[15] C. Harris, "Determination of ego-motion from matched points," presented at Proceedings Alvey Vision Conference, Cambridge, UK, 1987.

[16] J. A. Noble, "Finding corners," Image and Vision Computing, vol. 6, pp. 121-128, 1988.

[17] C. Harris and M. Stephens, "A combined corner and edge detector," Proceedings 4th Alvey Vision Conference, pp. 189-192, 1988.

[18] M. Sonka, V. Hlavac, and R. Boyle, Image Processing, Analysis, and Machine Vision, Second ed: PWS Publishing, 1999.

[19] P. Kovesi, "Matlab functions for computer vision and image analysis."
http://www.csse.uwa.edu.au/~pk/Research/MatlabFns

[20] P. Kovesi, "Phase Congruency Detects Corners and Edges," presented at The Australian Pattern Recognition Society Conference: DICTA, Sydney, 2003.

[21] J. Cooper, S. Venkatesh, and L. Kitchen, "The Dissimilarity Corner Detector," Fifth International Conference on Advanced Robotics'91, pp. 1377-1382, 1991.

[22] J. Cooper, S. Venkatesh, and L. Kitchen, "Early Jump-out Corner Detectors," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-15, pp. 823-828, 1993.

[23] J. Bernd, Digital image processing : concepts, algorithms, and scientific applications, 3rd ed: Springer-Verlag, 1995.

[24] R. Jain, R. Kasturi, and B. G. Schunck, Machine Vision: New York : McGraw-Hill, 1995.

[25] F. L. Alt, "Digital pattern recognition by moments," Journal of Association for Computing Machiney, vol. 9, pp. 240-258, 1962.

[26] H. E. Lü and P. S. P. Wang, "A Comment on "A Fast Parallel Algorithm for Thinning Digital Patterns"," Communications of the Association for Computing Machinery, vol. 29, pp. 239-242, 1986.

[27] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," Communications of the Association for Computing Machinery, vol. 27, pp. 236-239, 1984.

[28] C. Arcelli and G. S. Di Baja, "A width-independent fast thinning algorithm," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 7, pp. 463-474, 1985.

[29] K. Preston and M. J. B. Duff, Modern Cellular Automata. New York: Plenum, 1984.

[30] A. Rosenfeld and A. Kak, "Digital Picture Processing," in Academic Press, vol. 2. New York, 1982.

[31] J. Wei and N. Härle, "Use of temporal redundancy of motion vectors for the increase of optical flow calculation speed sa a contribution to real-time robot vision," Queensland University of Technology 1997.

[32] S. Temizer, "Optical flow based robot navigation."
http://www.ai.mit.edu/people/lpk/mars/temizer_2001/Optical_Flow/

[33] H. K. Nishihara, "Real-Time Implementation of a Sign-Correlation Algorithm for Image-Matching," in Technical report, Teleos Research, 1990.

[34] M. Grünewald and J. Sitte, "A Resource-Efficient Approach to Obstacle Avoidance via Optical Flow," presented at Proceedings of the 5th International Heinz Nixdorf Symposium: Autonomous Minirobots for Research and Edutainment (AMIRE), Heinz Nixdorf Institute, 2001.

[35] P. H. Batavia, D. A. Pomerleau, and C. E. Thorpe, "Detecting Overtaking Vehicles with Implicit Optical Flow," Carnegie Mellon University 1998.

[36] E. Kreyszig, Advanced engineering mathematics, 8th ed. New York: John Wiley, 1999.

[37] "Mathworld," Wolfram Web Resource.
http://mathworld.wolfram.com/

[38] A. Jain, Fundamentals of Digital Image Processing: Prentice-Hall, 1989.

[39] R. Klette and P. Zamperoni, Handbook of Image Procssing Operators: John Wiley & Sons, 1996.

[40] J. R. Parker, Algorithms for Image Processing and Computer Vision. New York: Wiley Computer Publishing, 1997.

[41] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Image Processing Operator Worksheets," HIPR2, 2003.
http://homepages.inf.ed.ac.uk/rbf/HIPR2/wksheets.htm

[42] B. Woodham, "Image Understanding I: Image Analysis."
http://www.cs.ubc.ca/~woodham/cpsc505/examples/log.html

[43] "Glossary," Image Processing Solutions.
http://www.ipsimaging.com/support/glossary.htm

[44] M. Jiang, "Digital Image Processing," Department of Information Science, School of Mathematics, Peking University.
http://ct.radiology.uiowa.edu/~jiangm/courses/dip/html/dip.html

[45] J. R. Parker, Practical computer vision using C. New York: Wiley Computer Publishing, 1994.

[46] F. Shen and H. Wang, "Real Time Gray Level Corner Detector," presented at Proc. 6th International Conference on Control, Automation, Robotics and Vision (ICARCV2000), Singapore, 2000.

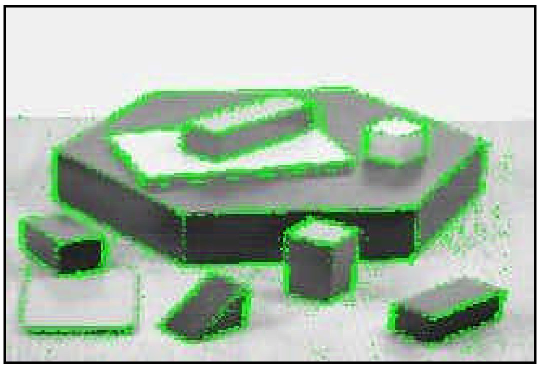
[47] T. C. Manjunath, Fundamentals of Robotics: Nandu Printers and Publishers Private Limited, 2001.

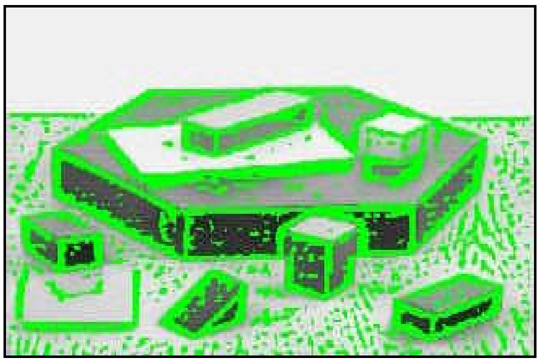
[48] G.-S. Young, T.-H. Hong, M. Herman, and J. C. S. Yang, "New visual invariants for obstacle detection using optical flow induced from general motion," 1992.

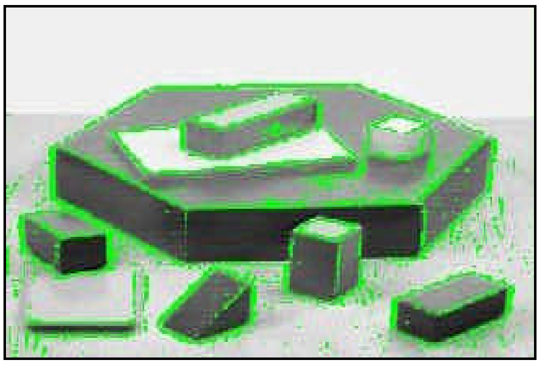
[49] E. W. Weisstein, "Correlation," in MathWorld--A Wolfram Web Resource.
http://mathworld.wolfram.com/Correlation.html

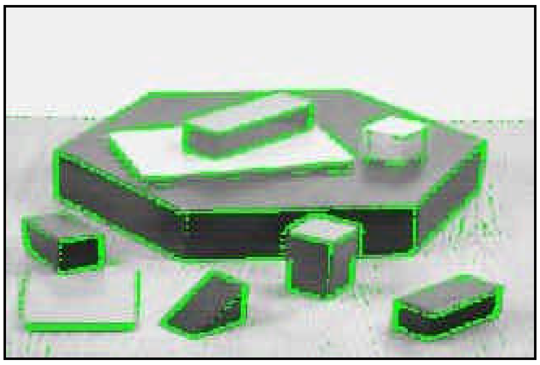
# Appendix: Colour Slides

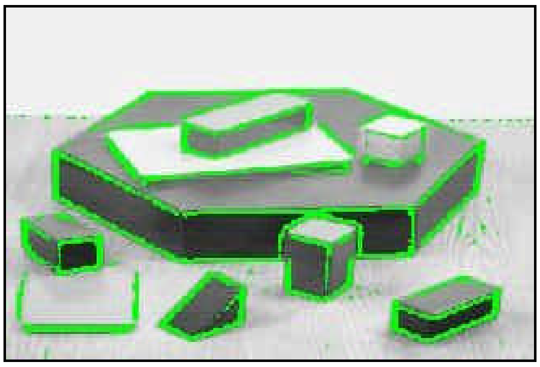
Edge detectors:

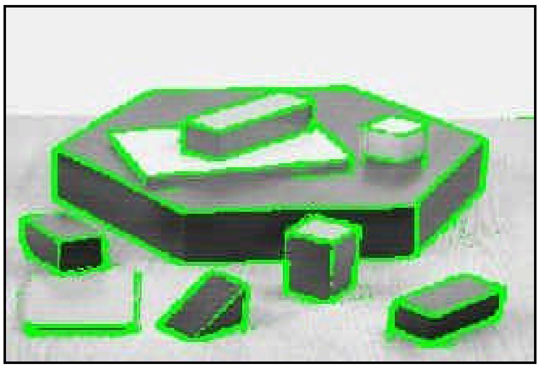

(a)

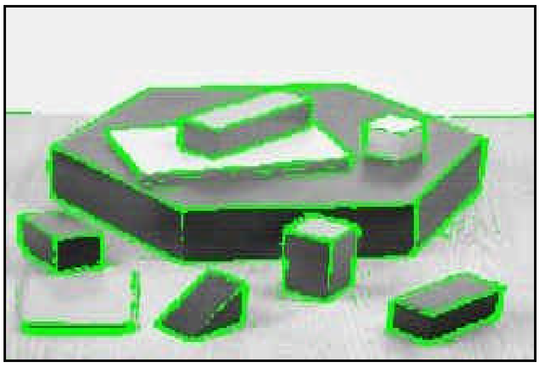

(b)

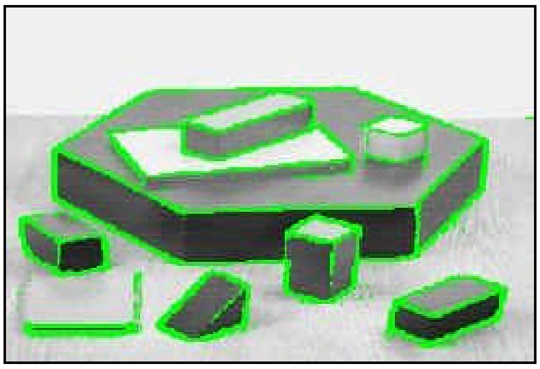

(c)

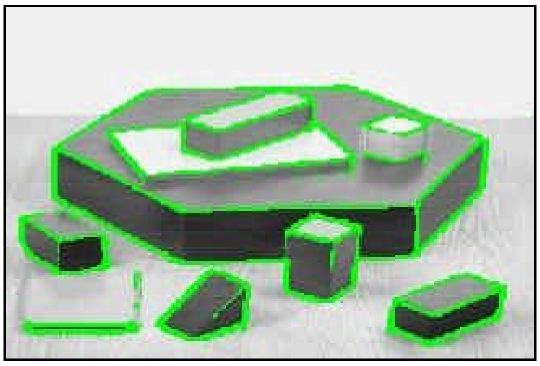

(d)


(e)


(f)


(g)


(h)


(i)

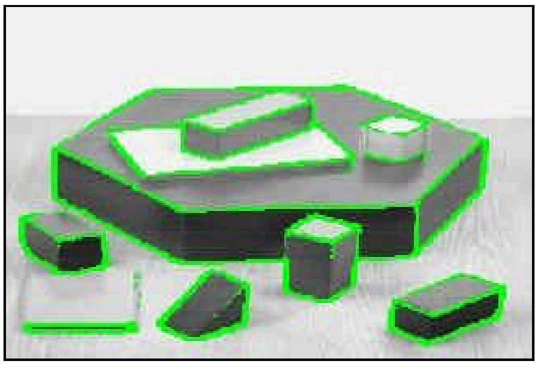
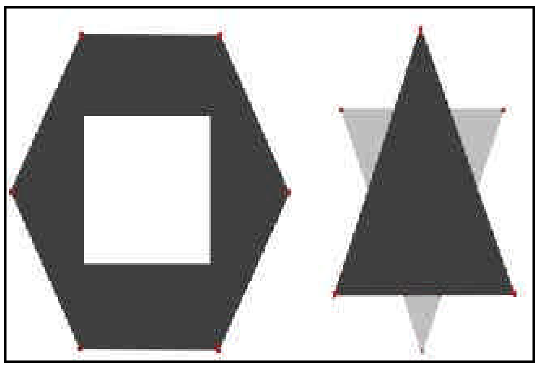
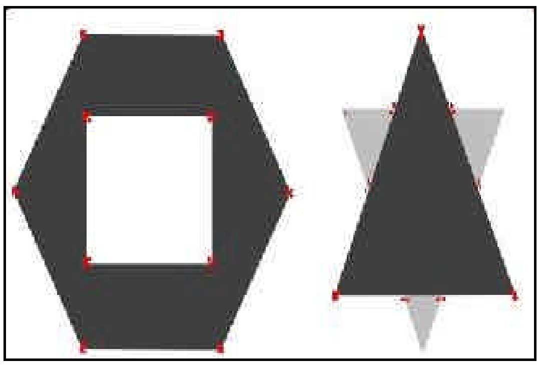
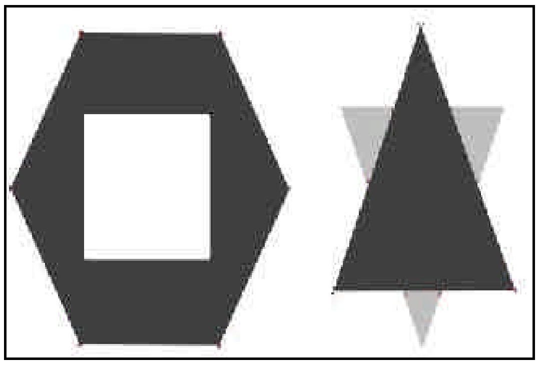

(j)

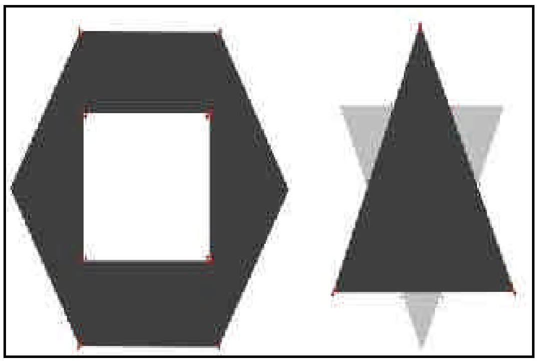
Figure 86: The results of the edge detectors overlayed on the original image (Figure 21). (a) Laplace (Figure 21, threshold value = 26). (b) LoG with predefined 9 x 9 template (Figure 24, threshold value = 255). (c) LoG with $s$ = 1.5 (Figure 26, threshold value = 230). (d) DoB with mean 5 x 5 and mean 3 x 3 (Figure 31, threshold value = 5). (e) DoB with mean 7 x 7 and mean 5 x 5 (Figure 32, threshold value = 8). (f) Sobel (Figure 33, threshold value = 38). (g) Robert's (Figure 35, threshold value = 31). (h) Kirsch (Figure 37, threshold value = 92). (i) Prewitt (Figure 39, threshold value = 51). (j) Robinson (Figure 41, threshold value = 64).

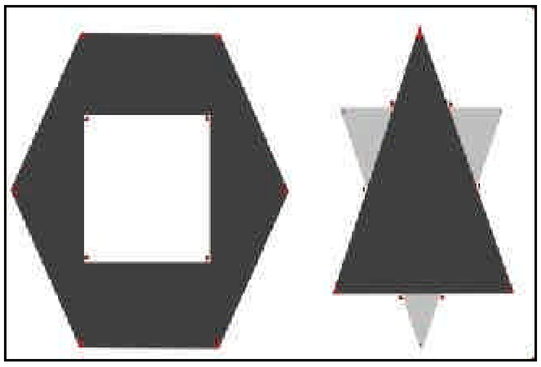
Corner detectors:

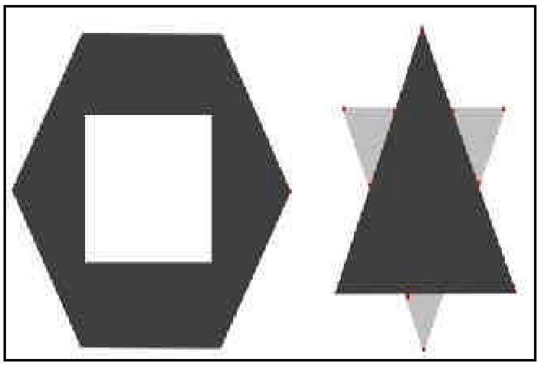

(a)

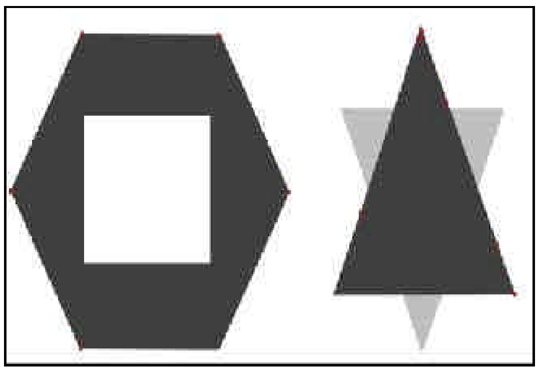

(b)

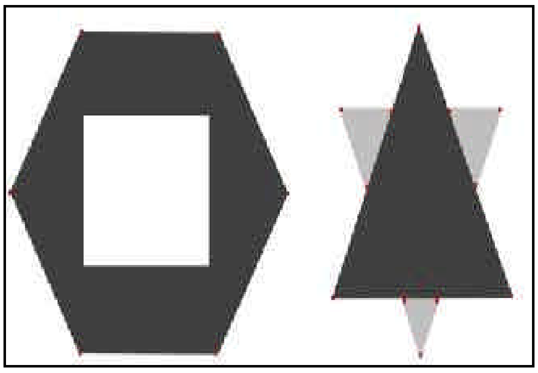

(c)

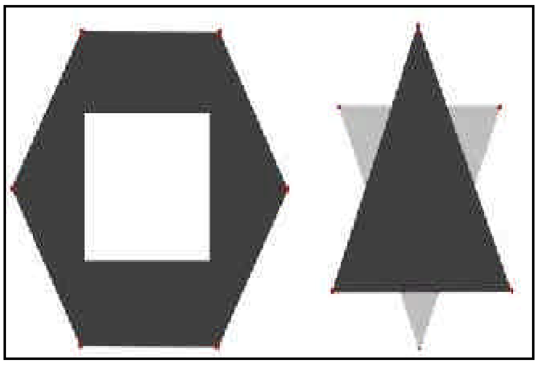

(d)


(e)


(f)


(g)


(h)


(i)

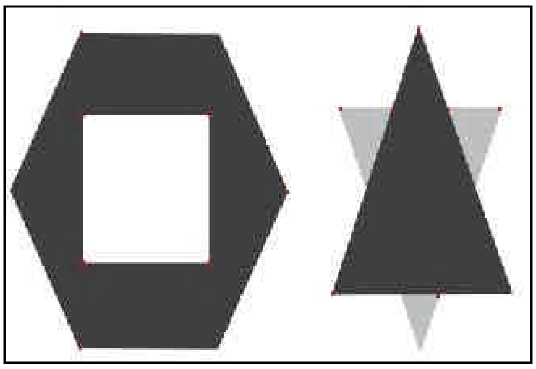
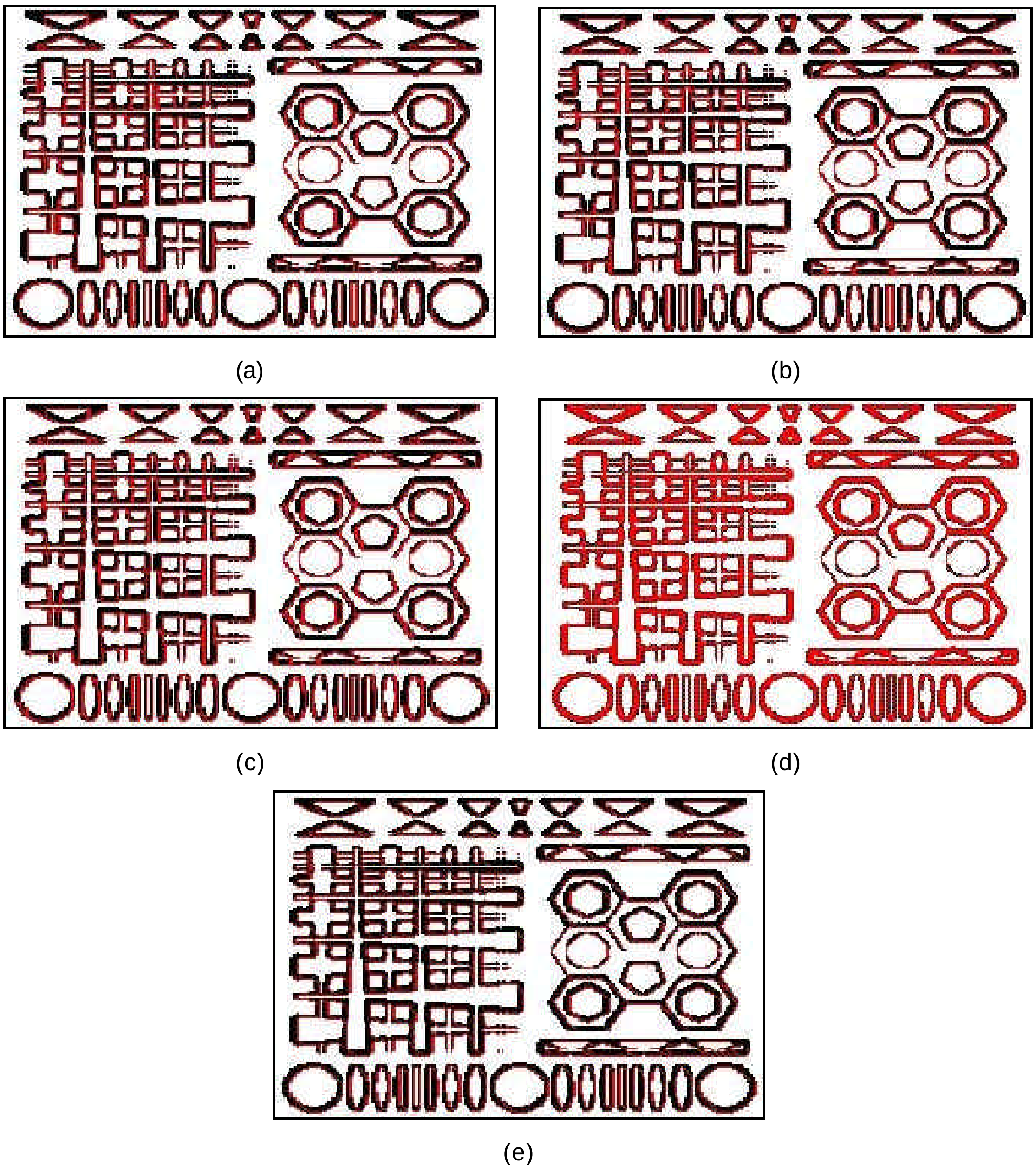

(j)

Figure 87: The results of the corner detectors overlayed on the original image. (a) Kitchen-Rosenfeld (Figure 47, threshold value = 18). (b) Beaudet (Figure 49, threshold value = 255). (c) Plessey (Figure 51, threshold value = 43). (d) Noble (Figure 53, threshold value = 255, $s$ = 0.35). (e) Harris-Stephens (Figure 55, threshold value = 217, k = 0.04). (f) Zero-1 (Figure 61, threshold value = 13). (g) Zero-2 (Figure 62, threshold value = 71). (h) Maxima-1 (Figure 67, threshold value = 13). (i) Maxima-2 (Figure 68, threshold value = 18). (j) Binary-1 (Figure 70, threshold value = 255).

Thinning filters:



(a)

(b)

(c)

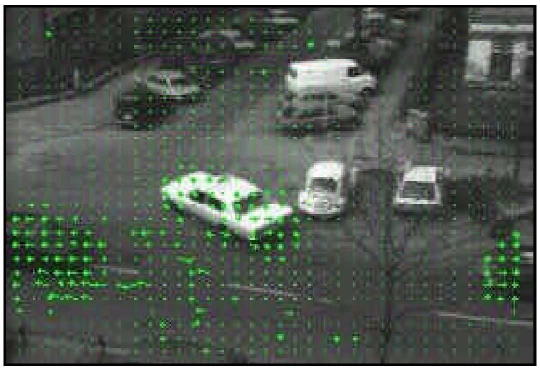
(d)

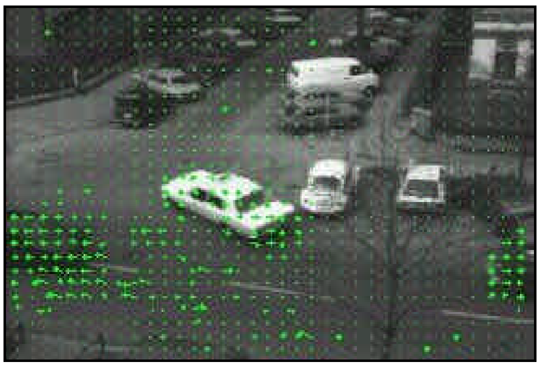
(e)

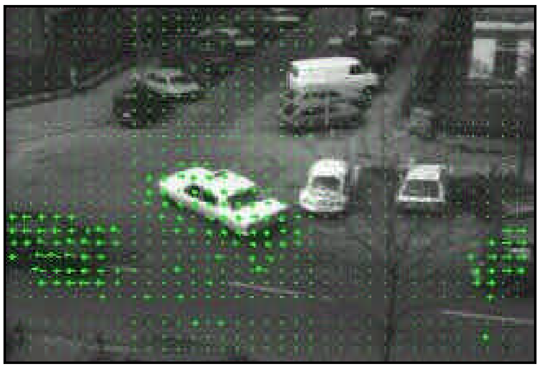
Figure 88: The results of the thinning filters overlayed on the original image. The black pixels (part of the original edge) are the ones that were deleted by the respective filters. (a) Stefan-1 (Figure 73), (b) Stefan-2 (Figure 75), (c) Lü-Wang (Figure 77), (d) Hall -1 (Figure 79), (e) Hall-2 (Figure 81).

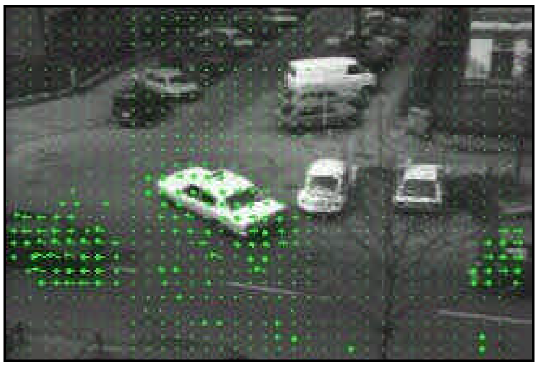
All the thinning filters were applied to the result of Mean 3 x 3 filter, Sobel edge detector, Threshold (value 51) and Negation (for considering black edges).

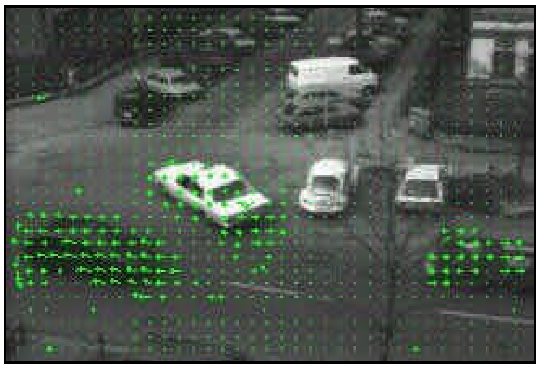
Optic Flow:


(a)


(b)


(c)


(d)


(e)

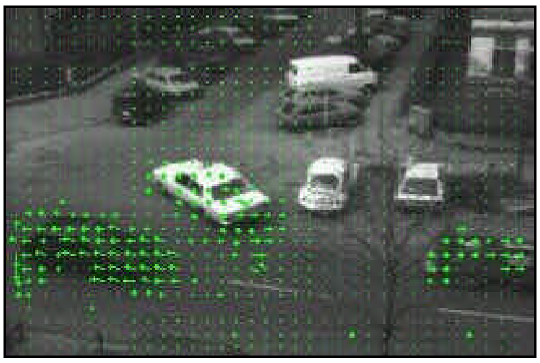

(f)

Figure 89: The Optic flow computation. (a), (c), and (e) SSD at frames 2, 5, and 13 respectively (Figure 84). (b), (d), and (f) SAD at frames 2, 5, and 13 respectively (Figure 85). The factors 'spacing' = 8, 'search window' = 11 x 11, and 'correlation window' = 11 x 11 was kept constant for both the calculations.