Scholars Journal of Engineering and Technology (SJET) **ISSN 2321-435X**
Sch. J. Eng. Tech., 2013; 1(2):78-90
©Scholars Academic and Scientific Publisher
(An International Publisher for Academic and Scientific Resources)
www.saspublisher.com

Research Article

Comparative Study and Analysis of Various Edge Detection Algorithms in Digital Image Processing

Sreemana Datta
Heritage Institute of Technology, Chowbaga Road, Anandapur, P.O.East Kolkata Township, Kolkata – 700107, West Bengal, India

***Corresponding author**
SreemanaDatta
Email: sreemanadatta@gmail.com

Abstract: In the field of image processing, edge detection is an important step for extracting relevant and meaningful information from digital images. The main goal of edge detection techniques is to obtain and detect thin edges of the objects present in the image, so that the result is more suitable for further processing and analysis such as boundary detection, image segmentation, motion detection/estimation, texture analysis, object identification, feature detection, implementing various transformations and so on. We tested six edge detection algorithms that use different methods for detecting edges and compared their results under a variety of situations to determine a generally preferable technique under different sets of conditions. This data could then be used to create a multi-edge-detector system, which analyses the scene and runs the edge detector best suited for the current set of data. For each of these edge detectors we considered two different ways of implementation, the one using intensity only and the other coupling to it, the colour information. We also considered one additional edge detector which takes a different philosophy to edge detection. Rather than trying to find the ideal edge detector to apply to traditional photographs, it would be more efficient to merely change the method of photography to one which is more conducive to edge detection. It makes use of a camera that takes multiple images in rapid succession under different lighting conditions. It has been observed that the Canny's edge detection algorithm performs better than all these operators under almost all scenarios. Evaluation of the images showed that under noisy conditions Canny, LoG(Laplacian of Gaussian), Robert, Prewitt, Sobel exhibit better performance, respectively. It has been observed that Canny's edge detection algorithm is computationally more expensive compared to LoG(Laplacian of Gaussian), Sobel, Prewitt and Robert's cross operator.

Keywords: Boolean Edge Detector, Sobel Operator, Prewitt's Operator, Canny Edge Detector, Vector Angle Detector, Euclidean Distance

1.INTRODUCTION

Edge detection refers to the process of identifying and locating sharp discontinuities in an image. The discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene. Classical methods of edge detection involve convolving the image with an operator (a 2-D filter), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions. There are an extremely large number of edge detection operators available, each designed to be sensitive to certain types of edges. Variables involved in the selection of an edge detection operator include Edge orientation, Noise environment and Edge structure. The geometry of the operator determines a characteristic direction in which it is most sensitive to edges. Operators can be optimized to look for horizontal, vertical, or diagonal edges. Edge detection is difficult in noisy images, since both the noise and the edges contain high frequency content. Attempts to reduce the noise result in blurred and distorted edges. Operators used on noisy images are typically larger in scope, so they can average enough data to discount localized noisy pixels. This results in less accurate localization of the detected edges. Not all edges involve a step change in intensity. Effects such as refraction or poor focus can result in objects with boundaries defined by a gradual change in intensity. The operator needs to be chosen to be responsive to such a gradual change in those cases. So, there are problems of false edge detection, missing true edges, edge localization, high computational time and problems due to noise etc. Therefore, the objective is to do the comparison of various edge detection techniques and analyse the performance of the various techniques in different conditions. Edge detection is a very important area in the field of Computer Vision. Edges define the boundaries or sharp discontinuities between regions in an image, which helps with segmentation and object recognition. They can show where shadows fall in an image or any other distinct change in the intensity of an image. Edge detection is a fundamental of low-level image processing and good edges are necessary for higher level processing. The problem is that in general edge detectors behave very poorly. While their behaviour may fall within tolerances in specific situations, in general edge detectors have difficulty adapting to different situations. The quality of edge detection is highly dependent on lighting conditions, the presence of objects of similar intensities, density of edges in the scene, and noise. While each of these problems can be handled by adjusting certain values in the edge detector and changing the threshold value for what is considered an edge, no good method has been determined for automatically setting these values, so they must be manually changed by an operator each time the detector is run with a different set of data. Since different edge detectors work better under different conditions, it would be ideal to have an algorithm that makes use of multiple edge detectors, applying each one when the scene conditions are most ideal for its method of detection. In order to create this system, you must first know which edge detectors perform better under which conditions, which is the goal of our research. We tested six edge detectors that use different methods for detecting edges and compared their results under a variety of situations to determine which detector was preferable under different sets of conditions. This data could then be used to create a multi-edge-detector system, which analyses the scene and runs the edge detector best suited for the current set of data. For one of the edge detectors we considered two different ways of implementation, one using intensity only and the other using colour information. We also considered one additional edge detector which takes a different philosophy to edge detection. Rather than trying to find the ideal edge detector to apply to traditional photographs, it would be more efficient to merely change the method of photography to one which is more conducive to edge detection. It makes use of a camera that takes multiple images in rapid succession under different lighting conditions. Since the hardware for this sort of edge detection is different than that used with the other edge detectors, it would not be included in the multiple edge detector system but can be considered as a viable alternative to the existent system.

2. Edge Detection

2.1. Variables involved in selection of edge detector

Edge orientation: [1] [5] The geometry of the operator determines a characteristic direction in which it is most sensitive to edges. Operators can be optimized to look for horizontal, vertical, or diagonal edges.

Noise environment: [5] Edge detection is difficult in noisy images, since both the noise and the edges contain high-frequency content. Attempts to reduce the noise result in blurred and distorted edges. Operators used on noisy images are typically larger in scope, so they can average enough data to discount localized noisy pixels. This results in less accurate localization of the detected edges.

Edge structure: [1][3] Not all edges involve a step change in intensity. Effects such as refraction or poor focus can result in objects with boundaries defined by a gradual change in intensity. The operator needs to be chosen to be responsive to such a gradual change in those cases. Newer wavelet-based techniques actually characterize the nature of the transition for each edge in order to distinguish, for example, edges associated with hair from edges associated with a face.

2.2. Edge detection methodologies

There are many ways to perform edge detection. However, the majority of different methods may be grouped into two categories:

Gradient: The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image.

Laplacian: [8] The Laplacian method searches for zero crossings in the second derivative of the image to find edges. An edge has the one-dimensional shape of a ramp and calculating the derivative of the image can highlight its location. Suppose we have the following signal, with an edge shown by the jump in intensity below:

Suppose we have the following signal, with an edge shown by the jump in intensity below:

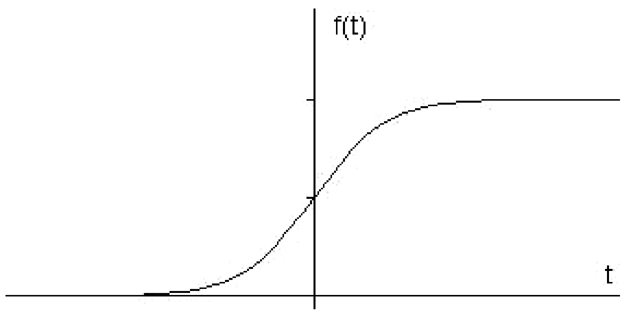


Figure 1: Intensity jump at edge

If we take the gradient of this signal (which, in one dimension, is just the first derivative with respect to t) we get the following:

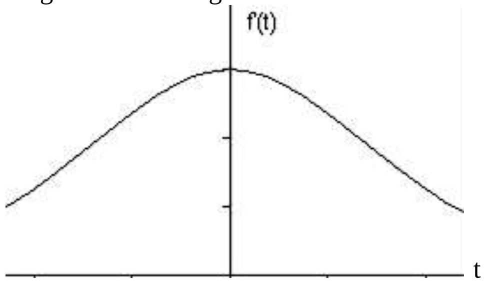


Figure 2: Gradient of the signal

Clearly, the derivative shows a maximum located at the center of the edge in the original signal. This method of locating an edge is characteristic of the “gradient filter” family of edge detection filters and includes the Sobel’s method. A pixel location is declared an edge location if the value of the gradient exceeds some threshold. As mentioned before, edges will have higher pixel intensity values than those surrounding it. So once a threshold is set, you can compare the gradient value to the threshold value and detect an edge whenever the threshold is exceeded. Furthermore, when the first derivative is at a maximum, the second derivative is zero. As a result, another alternative to finding the location of an edge is to locate the zeros in the second derivative. This method is known as the Laplacian and the second derivative of the signal is shown below:

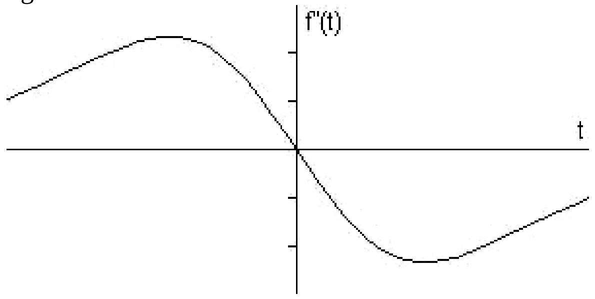


Figure 3: Second derivative of the signal

2.3 Various Edge Detection Techniques

2.3.1 Local Threshold and Boolean Function Based Edge Detection

[9] This edge detector is fundamentally different than many of the modern edge detectors derived from Canny's original. It does not rely on the gradient or Gaussian smoothing. It takes advantage of both local and global thresholding to find edges. Unlike other edge detectors, it converts a window of pixels into a binary pattern based on a local threshold, and then applies masks to determine if an edge exists at a certain point or not. By calculating the threshold on a per pixel basis, the edge detector should be less sensitive to variations in lighting throughout the picture. It does not rely on blurring to reduce noise in the image. It instead looks at the variance on a local level.

The algorithm is as follows:

Step 1

Apply a local threshold to a 3x3 window of the image. Because this is a local threshold, it is recalculated each time the window is moved. The threshold value is calculated as the mean of the 9 intensity values of the pixels in the window minus some small tolerance value. If a pixel has an intensity value greater than this threshold, it is set to a 1. If a pixel has an intensity value less than this threshold, it is set to a 0.

This gives a binary pattern of the 3x3 window.

Step 2

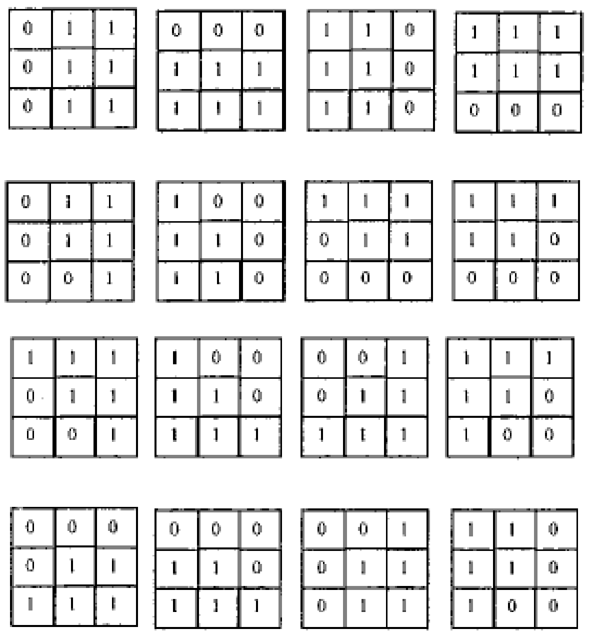
Compare the binary pattern to the edge masks. There are sixteen possible edge-like patterns that can arise in a 3x3 window, as shown in figure 4. If the binary window obtained in step matches any of these sixteen masks, the centre pixel of the window is set to be an edge pixel.

Step 3

Repeat steps a and b for each pixel in the image as the centre pixel of the window. This will give all edges, but it will also give some false edges as a result of noise.

Step 4

Use a global threshold to remove false edges. The variance for each 3x3 window is calculated, which will have a maximum at an edge. This value is then compared with a global threshold based on the level of noise in the image. If the value is greater than the threshold, it is kept as an edge. If it is not greater than the threshold, it is removed.



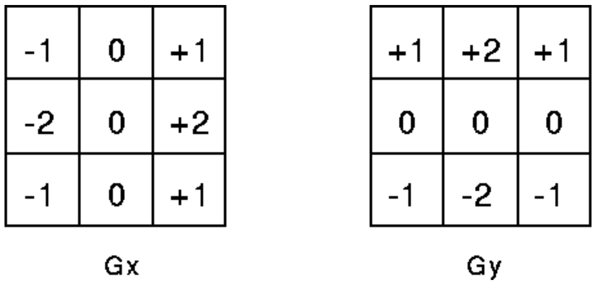
0	1	1		0	0	0		1	1	0		1	1	1
0	1	1		1	1	1		1	1	0		1	1	1
0	1	1		1	1	1		1	1	0		0	0	0
0	1	1		1	0	0		1	1	1		1	1	1
0	1	1		1	1	0		0	1	1		1	1	0
0	0	1		1	1	0		0	0	0		0	0	0
1	1	1		1	0	0		0	0	1		1	1	1
0	1	1		1	1	0		0	1	1		1	1	0
0	0	1		1	1	1		1	1	1		1	0	0
0	0	0		0	0	0		0	0	1		1	1	0
0	1	1		1	1	0		0	1	1		1	1	0
1	1	1		1	1	1		0	1	1		1	0	0

Figure 4: Possible Edge Masks for 3X3 Window

2.3.2 Sobel Operator

[3] The operator consists of a pair of 3×3 convolution kernels as shown in Figure 1.

One kernel is simply the other rotated by 90°.



-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these *Gx* and *Gy*). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

Typically, an approximate magnitude is computed using:

$$|G| = |Gx| + |Gy|$$

which is much faster to compute.

The angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by:

$$\theta = \arctan(Gy/Gx)$$

2.3.3 Robert's cross operator:

[2] The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point. The operator consists of a pair of 2×2 convolution kernels as shown in Figure. One kernel is simply the other rotated by 90°. This is very similar to the Sobel operator.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

These kernels are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these *Gx* and *Gy*). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

although typically, an approximate magnitude is computed using:

$$|G| = |Gx| + |Gy|$$

which is much faster to compute.

The angle of orientation of the edge giving rise to the spatial gradient (relative to the pixel grid orientation) is given by:

$$\theta = \arctan(Gy/Gx) - 3\pi/4$$

2.3.4 Prewitt's operator:

Prewitt operator is similar to the Sobel operator and is used for detecting vertical and horizontal edges in images.

$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \qquad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

2.3.5 Laplacian of Gaussian (Marr-Hildreth Edge Detector) :

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian Smoothing filter in order to reduce its sensitivity to noise. The operator normally takes a single gray level image as input and produces another gray level image as output.

The Laplacian *L(x,y)* of an image with pixel intensity values *I(x,y)* is given by:

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

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Three commonly used small kernels are shown in Figure 1.

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

−1	2	−1
2	−4	2
−1	2	−1

Figure 1 Three commonly used discrete approximations to the Laplacian filter.

Because these kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often

Gaussian Smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

In fact, since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter first of all, and then convolve this hybrid filter with the image to achieve the required result. Doing things this way has two advantages:

- Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations.
- The LoG (`Laplacian of Gaussian') kernel can be precalculated in advance so only one convolution needs to be performed at run-time on the image.

The 2-D LoG function centered on zero and with Gaussian standard deviation σ has the form:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

and is shown in Figure 2.

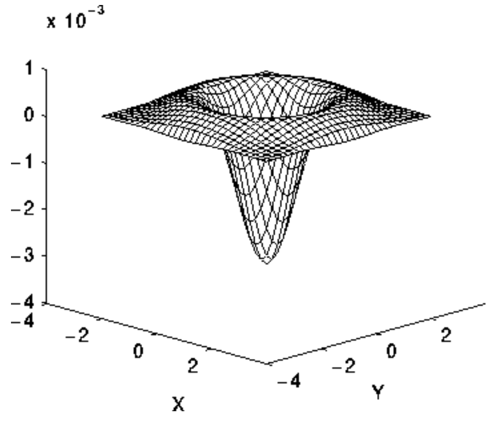


Figure 5: The Mexican hat operator

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Figure 6: Discrete approximation to LoG function with Gaussian σ = 1.4

Note that as the Gaussian is made increasingly narrow, the LoG kernel becomes the same as the simple Laplacian kernels shown in Figure 1. This is because smoothing with a very narrow Gaussian (σ < 0.5 pixels) on a discrete grid has no effect. Hence on a discrete grid, the simple Laplacian can be seen as a limiting case of the LoG for narrow Gaussians.

2.3.6 Canny's Edge Detection Algorithm

[11][12] The Canny edge detection algorithm is known to many as the optimal edge detector. Canny's intentions were to enhance the many edge detectors already out at the time he started his work. He was very successful in achieving his goal and his ideas and methods can be found in his paper, "A Computational Approach to Edge Detection". In his paper, he followed a list of criteria to improve current methods of edge detection. The first and most obvious is low error rate. It is important that edges occurring in images should not be missed and that there be NO responses to non-edges. The second criterion is that the edge points be well localized. In other words, the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum. A third criterion is to have only one response to a single edge. This was implemented because the first 2 were not substantial enough to completely eliminate the possibility of multiple responses to an edge.

Based on these criteria, the canny edge detector first smoothes the image to eliminate and noise. It then finds the image gradient to highlight regions with high spatial derivatives. The algorithm then tracks along these regions and suppresses any pixel that is not at the maximum (non-maximum suppression). The gradient array is now further reduced by hysteresis. Hysteresis is used to track along the remaining pixels that have not been suppressed. Hysteresis uses two thresholds and if the magnitude is below the first threshold, it is set to zero (made a nonedge). If the magnitude is above the high threshold, it is made an edge. And if the magnitude is between the 2 thresholds, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above T2.

Step 1

In order to implement the canny edge detector algorithm, a series of steps must be followed. The first step is to filter out any noise in the original image before trying to locate and detect any edges. And because the Gaussian filter can be computed using a simple mask, it is used exclusively in the Canny algorithm. Once a suitable mask has been calculated, the Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The larger the width of the Gaussian mask, the lower is the detector's sensitivity to noise. The localization error in the detected edges also increases slightly as the Gaussian width is increased. The Gaussian mask used in my implementation is shown below.

$\frac{1}{115}$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

Figure 7: Discrete approximation to Gaussian function with σ = 1.4

Step 2

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). They are shown below:

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

The magnitude, or edge strength, of the gradient is then approximated using the formula:

|G| = |Gx| + |Gy|

Step 3

The direction of the edge is computed using the gradient in the x and y directions. However, an error will be generated when sumX is equal to zero. So in the code there has to be a restriction set whenever this takes place. Whenever the gradient in the x direction is equal to zero, the edge direction has to be equal to 90 degrees or 0 degrees, depending on what the value of the gradient in the y-direction is equal to. If GY has a value of zero, the edge direction will equal 0 degrees. Otherwise the edge direction will equal 90 degrees. The formula for finding the edge direction is just:

Theta = invtan (Gy / Gx)

Step 4

Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced in an image. So if the pixels of a 5x5 image are aligned as follows:

```
x     x     x     x     x
x     x     x     x     x
x     x     a     x     x
x     x     x     x     x
x     x     x     x     x
```

Then, it can be seen by looking at pixel "a", there are only four possible directions when describing the surrounding pixels - 0 degrees (in the horizontal direction), 45 degrees (along the positive diagonal), 90 degrees (in the vertical direction), or 135 degrees (along the negative diagonal). So now the edge orientation has to be resolved into one of these four directions depending on which direction it is closest to (e.g. if the orientation angle is found to be 3 degrees, make it zero degrees). Think of this as taking a semicircle and dividing it into 5 regions.

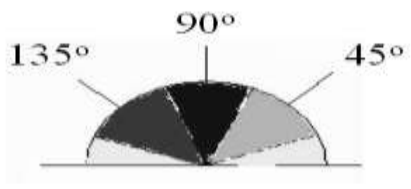


Figure 8: Angles describing neighbouring pixels

Therefore, any edge direction falling within the yellow range (0 to 22.5 & 157.5 to 180 degrees) is set to 0 degrees. Any edge direction falling in the green range (22.5 to 67.5 degrees) is set to 45 degrees. Any edge direction falling in the blue range (67.5 to 112.5 degrees) is set to 90 degrees. And finally, any edge direction falling within the red range (112.5 to 157.5 degrees) is set to 135 degrees.

Step 5

After the edge directions are known, non-maximum suppression now has to be applied. Non maximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (sets it equal to 0) that is not considered to be an edge. This will give a thin line in the output image.

Step 6

Finally, hysteresis is used as a means of eliminating streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold. If a single threshold, T1 is applied to an image, and an edge has an average strength equal to T1, then due to noise, there will be instances where the edge dips below the threshold. Equally it will also extend above the threshold making an edge look like a dashed line. To avoid this, hysteresis uses 2 thresholds, a high and a low. Any pixel in the image that has a value greater than T1 is presumed to be an edge pixel, and is marked as such immediately. Then, any pixels that are connected to this edge pixel and that have a value greater than T2 are also selected as edge pixels. If you think of following an edge, you need a gradient of T2 to start but you don't stop till you hit a gradient below T1.

2.3.7 Colour Edge Detection Using Euclidean Distance and Vector Angle

[14][16] Most edge detectors work on the grayscale representation of the image. This cuts down the amount of data you have to work with (one channel instead of three), but you also lose some information about the scene. By including the colour component of the image, the edge detector should be able to detect edges in regions with high colour variation but low intensity variation.

This edge detector uses two operators: Euclidean Distance and Vector Angle. The Euclidean Distance is a good operator for finding edges based on intensity and the Vector Angle is a good operator for finding edges based on hue and saturation. The detector applies both operators to the RGB colour space of an image, and then combines the results from each based on the amount of colour in a region. There is a difference vector and a vector gradient version; we chose to implement the vector gradient version.

The Euclidean Distance between two pixels is defined as:

$$D(\vec{v}_1, \vec{v}_2) = \left\| \vec{v}_1 - \vec{v}_2 \right\|$$

where v1 and v2 are RGB triplets (v = [R G B]).

The Vector Angle between two pixels is approximated by:

$$\sin\theta = \left(1 - \left(\frac{\vec{v}_1^{\,T} \vec{v}_2}{\left\|\vec{v}_1\right\| \cdot \left\|\vec{v}_2\right\|} \right)^2 \right)^{1/2}$$

because sin θ ≈ θ for small angles. Again, v1 and v2 are RGB triplets (v = [R G B]). The Euclidean Distance and Vector Angle are combined using a saturation-based combination method. This combination is defined as:

$$C_{GV} = \rho(S_1, S_2) \sqrt{1 - \left(\frac{v_i^T(x,y) \cdot v_0(x,y)}{\left\|v_i(x,y)\right\| \left\|v_0(x,y)\right\|} \right)^2} + (1 - \rho(S_1, S_2)) \cdot \left\| v_i(x,y) - v_0(x,y) \right\|$$

Where

$$\rho(S_1, S_2) = \sqrt{\alpha(S_1) \cdot \alpha(S_2)}$$

And

$$\alpha(S) = \frac{1}{1 + e^{-slope(S - offset)}}$$

The "slope" and "offset" values in the sigmoid α(S) are set experimentally, and S1 and S2 are the saturation values of each pixel. This combination weights the Vector Angle more heavily in areas of high colour saturation and the Euclidean Distance more heavily in areas of low saturation.

The algorithm for finding edges in the image is as follows:

Step 1

For each pixel in the image, take the 3x3 window of pixels surrounding that pixel.

Step 2

Calculate the saturation-based combination of the Euclidean Distance and Vector Angle between the center point and each of the eight points around it.

Step 3

Assign the largest value obtained to the centre pixel.

Step 4

When each pixel has had a value assigned to it, run the results through a threshold to eliminate false edges.

2.3.8 Edge Detection of coloured images using the Canny Operator

[2][9]Another approach to edge detection using colour information is simply to extend a traditional intensity based edge detector into the colour space. This method seeks to take advantage of the known strengths of the traditional edge detector and tries to overcome its weaknesses by providing more information in the form of three colour channels rather than a single intensity channel. As the Canny edge detector is the current standard for intensity based edge detection, it seemed logical to use this operator as the basis for colour edge detection. The algorithm we used for applying colours to the Canny edge detector was a very simple one:

- Read in a colour image and divide it into its three separate colour channels.
- Run each colour channel through the Canny edge detector separately to find a resulting coloured edge map.
- Combine the resulting edge maps from each of the three colour channels into one complete edge map.

For this step there are a variety of ways you can combine the edges found for each different colour, but we found that a simple additive approach provided the best results. So if there was an edge in any of the three coloured edge maps, we added it to the general edge map.

2.3.9 Depth Edge Detection using Multi-Flash Imaging Technique

[4][7]This is another edge detector following the principle that using more data in the edge detection process should result in better detection of edges. However, in this case rather than merely extending from one channel of intensity to three channels of color, this edge detector actually makes use of multiple different images. The approach is based on taking successive photos of a scene, each with a different light source close to and around the camera's center of projection. The location of the shadows abutting depth discontinuities are used as a robust cue to create a depth edge map in both static and dynamic scenes. The idea is that rather than using complicated mathematical techniques to try to extract edges from existing photographs, we should change the way we take photographs in general. This technique uses a camera with four flashes located at cardinal directions around the lens to take four successive pictures. The differences in shadows between each picture suggest "depth edges", or edges caused by depth discontinuities in a scene. This method suppresses "texture edges", or edges caused by the texture of a surface which all lie at a relatively equivalent depth. This is accomplished by calculated the epipolar geometry of the shadows in the different images.

The general algorithm is as follows:

Step 1
Capture an image using only ambient light. Label this image as I_0.

Step 2
For 'n' different light sources located a positions P1-Pn, capture n pictures I+k, with k=1-n where I+k is the picture taken with light source position P_k

Step 3
Remove the ambient component from each image: I_k = I+ k - I0

Step 4
For all pixels x, Imax(x) = $\max_k$(I_k(x)), k = 1..n. Imax is the base image, which is an approximation of what image you would get if the light source were exactly at the center of the camera lens.

Step 5
For each image k, create a ratio image, R_k where R_k(x) = I_k(x)/ I_{max} (x). The intensity of a point in an image, if it is lit, will follow the following equation:

$$I_k(x) = \mu_k \rho(x) \left(\hat{L}_k(x) \cdot N(x)\right)$$

where μ_k is the magnitude of the light intensity, p(x) is the reflectance at point X, L_k(x) is the normalized light vector L_k (x) = P_k – X and N(x) is the surface normal. If the surface is not lit, I_k(x) = 0. So the equation for the ratio is:

$$R_k(x) = \frac{I_k(x)}{I_{\max}(x)} = \frac{\mu_k \left(\hat{L}_X(x) \cdot N(x)\right)}{\max_i \left(\mu_i \left(\hat{L}_i(x) \cdot N(x)\right)\right)}$$

However, if the objects are relatively diffuse and far from the camera relative to the positions of the light sources, the ratio can be approximated by simply

$$R_k(x) = \mu_k / \max_i(\mu_i).$$

This ratio will be close to 1 in areas lit by light source k and close to 0 in areas not lit by light source k.

Step 6
For each image R_k, traverse each epipolar ray from epipole e_k (location of the flashes). A sharp negative transition in the image indicates a depth edge. So if a pixel has a negative transition, mark that pixel as an edge. Since the flashes are oriented along the cardinal directions, tracing the epipolar rays is equivalent to walking along a row or column of the image.

2.4 Visual Comparison of Various Edge Detection Algorithms

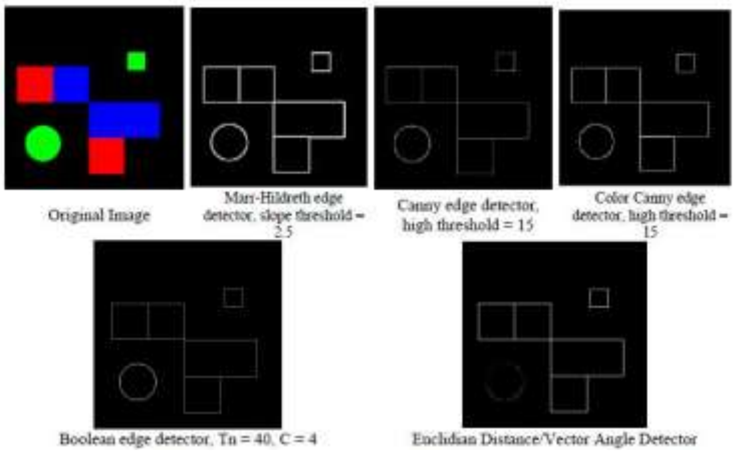

Original Image
Marr-Hildreth edge detector, slope threshold = 2.5
Canny edge detector, high threshold = 15
Color Canny edge detector, high threshold = 15
Boolean edge detector, Tn = 40, C = 4
Euclidian Distance/Vector Angle Detector

Figure 9: Test case 1

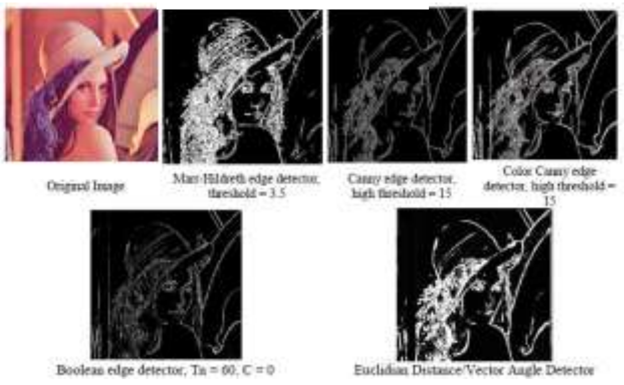

Original Image
Marr-Hildreth edge detector, threshold = 3.5
Canny edge detector, high threshold = 15
Color Canny edge detector, high threshold = 15
Boolean edge detector, Tn = 60, C = 0
Euclidian Distance/Vector Angle Detector

Figure 10: Test case 2

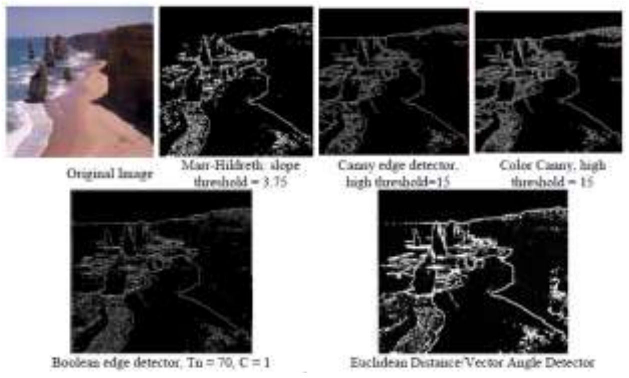

Original Image
Marr-Hildreth: slope threshold = 3.75
Canny edge detector, high threshold=15
Color Canny, high threshold = 15
Boolean edge detector, Tn = 70, C = 1
Euclidean Distance/Vector Angle Detector

Figure 11: Test case 3

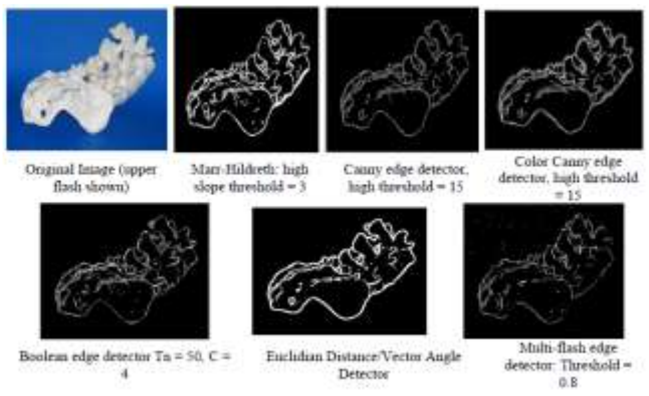

Original Image (upper flash shown)
Marr-Hildreth: high slope threshold = 3
Canny edge detector, high threshold = 15
Color Canny edge detector, high threshold = 15
Boolean edge detector Tn = 50, C = 4
Euclidian Distance/Vector Angle Detector
Multi-flash edge detector: Threshold = 0.8

Figure 12: Test case 4

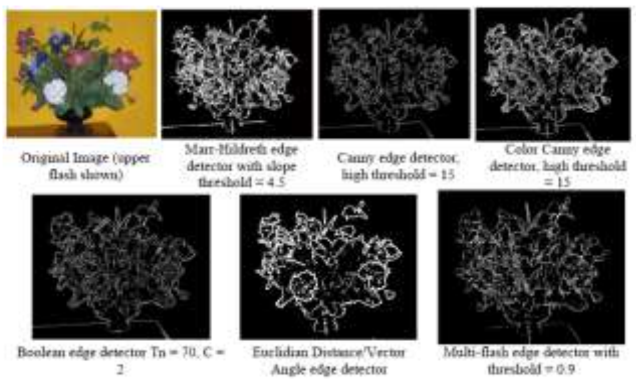

Original Image (upper flash shown)
Marr-Hildreth edge detector with slope threshold = 4.5
Canny edge detector, high threshold = 15
Color Canny edge detector, high threshold = 15
Boolean edge detector Tn = 70, C = 2
Euclidian Distance/Vector Angle edge detector
Multi-flash edge detector with threshold = 0.9

Figure 13: Test case 5

3. RESULTS

3.1. Performance of Edge Detection Algorithms

Gradient-based algorithms such as the Prewitt filter have a major drawback of being very sensitive to noise. The size of the kernel filter and coefficients are fixed and cannot be adapted to a given image. An adaptive **edge-detection** algorithm is necessary to provide a robust solution that is adaptable to the varying noise levels. Gradient-based algorithm**s** such as the Prewitt filter have a major drawback of being very sensitive to noise. The size of the kernel filter and coefficients are fixed and cannot be adapted to a given image. An adaptive **edge-detection** algorithm is necessary to provide a robust solution that is adaptable to the varying noise levels of these images to help distinguish valid image contents from visual artifacts introduced by noise.

The performance of the Canny algorithm depends heavily on the adjustable parameters, σ, which is the standard deviation for the Gaussian filter, and the threshold values, ‘T1’ and ‘T2’. σ also controls the size of the Gaussian filter. The bigger the value for σ, the larger the size of the Gaussian filter becomes. This implies more blurring, necessary for noisy images, as well as detecting larger edges. As expected, however, the larger the scale of the Gaussian, the less accurate is the localization of the edge. Smaller values of σ imply a smaller Gaussian filter which limits the amount of blurring, maintaining finer edges in the image. The user can tailor the algorithm by adjusting these parameters to adapt to different environments.

Canny’s edge detection algorithm is computationally more expensive compared to Sobel, Prewitt and Robert’s operator. However, the Canny’s edge detection algorithm performs better than all these operators under almost all scenarios.

3.2 Analysis of various Edge Detection Algorithms

We ran five separate test images through our edge detectors (except the Multi-Flash edge detector, which only had data sufficient data to run on two of the images). One image was artificial and the rest were real world photographs. The results are shown in the figures below. All coloured images were converted to grayscale using Matlab’s RBG2GRAY function except when using an edge detector that required colour information. Various threshold, sigma, slope, etc. values were chosen by hand. The images were blurred by a Gaussian filter (3x3 filter, sigma=1) before being fed into the Euclidean Distance and Vector Angle colour edge detector as we found this significantly decreased the effect of noise. The other edge detectors do their own smoothing (Where applicable). The low threshold in the Canny edge detector was always defined to be 40% of the high threshold. The average of the four images from the Multi-Flash edge detector was used as input to the other edge detectors. The first thing to notice about the Boolean and Euclidean Distance/Vector Angle edge detectors is that neither algorithm identifies edges down to a single point as in Canny's edge detector. The edges are often spotty and disconnected. Edge following to trace out a continuous edge is not possible in these edge detectors because the direction of the edge is not known (since they are not based on the gradient of the image). The Marr-Hildreth edge detector will give more nicely connected edges if hysteresis is used to threshold the image, and will not give connected edges if a single threshold is used. Regardless, it is usually gives spotty and thick edges.

Figure 9 shows the ability of the edge detectors to handle corners as well as a wide range of slopes in edge on the circle. The Canny edge detector becomes fairly confused at corners due to the Gaussian smoothing of the image. Also, since the direction of the edge changes instantly, corner pixels looks in the wrong directions for its neighbours. The colour version of Canny produces many double lines as the edge may be detected in a different location in each channel. The Boolean edge detector usually omits the corner pixel since the pattern for a corner is not one of the bit masks. Other than the omitted pixels, the Boolean edge detector performs well on the boxes and circle. The Euclidean Distance/Vector angle edge detector performs well on the blocks, creating a 2 pixel wide line along every edge. It became a little confused on the circle. This could possibly be remedied with a different threshold between the Euclidean Distance metric and the Vector Angle metric. The Marr-Hildreth edge detector creates lines even thicker than the Euclidean Distance.

Figure 10 is the standard edge detector benchmarking image. Overall, the Boolean edge detector performs a decent job of marking the locations of edges, but many of the edges are spotty and not contiguous. For the most part it detects the same edges as the Canny edge detector. The colour version of the Canny detector was able to find a few more edges than the grayscale version given the same input parameters. These edges are most noticeable on the hat. The Euclidean Distance detector creates much wider edges than the other two methods. It appears as if it marks edge pixels on both sides of the edge. As a result the output image shows the major edges, but not much fine grained detail. In general, the Boolean edge detector makes no guarantees about finding thin edges, but it usually does a reasonable job.

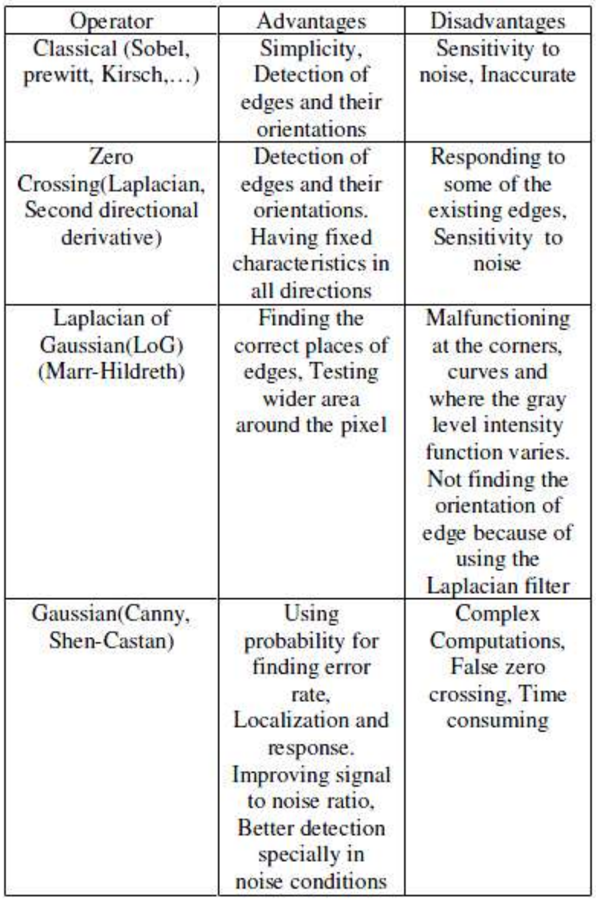
Figure 11 is a picture of a shoreline. All edge detectors had problems detecting the different ridges of the cliff. The foam of the waves also provided some inconsistent results. There are a lot of discrepancies in colour at these locations, but no clear edges. Similar to the Figure 2, the Euclidean Distance detector produces much thicker lines and less detail than the other edge detectors. The Boolean edge detector does a better job of maintaining contiguous curves for the edges, but they still have a few discontinuities.

Figure 12 is the first of the Multi-Flash images. The bright flashes in the different directions caused drop shadows around some edges of the object. Even though the average of all four images was used as the input to the other edge detectors, some of them still picked up on the faint shadows. Surprisingly, the Multi-Flash edge detector did not perform better than other edge detectors for this image. For example, it missed parts of the vertebrae. It could pick up these images if the threshold was reduced, but much more noise was introduced into the image. Other than the Boolean edge detector, the Multi-Flash method is the only detector that no smoothing of the image before processing, so it is more sensitive to noise in the image. The other edge detectors all identify almost the same edges, but follow similar behaviour to previous images.

Figure 13 more accurately shows the capabilities of each edge detector. The Marr-Hildreth detector perceives many edges, but they are too spotty and wide to really identify any features. The Canny edge detector gives nice outlines of the table, the vase, and many of the flowers on the border. Features in the middle of the arrangement are missed, but some are recovered with the addition of colour. For example, the red flower in the center and the leaves to its right are found. The Boolean edge detector does a good job at detecting a large number of edges, but many of them are noisy. The Euclidean Distance/Vector Angle edge detector finds strong edges around the colour flowers, but finds almost no edges in the middle of the green leafy region. It also misses the table entirely. The Multi-flash edge detector finds most of the geometry in the scene but misses parts of the table. If the background is too far away from the objects, the drop shadows will not be sharp enough to detect a discontinuity. The inventors of this algorithm supplement the edge detection with a step that finds the colour difference between the object and its background.

3.3 Relative advantage and disadvantages of various edge detection operators

It was found during our analysis that there was no optimal algorithm which would give the best result in all possible cases. However, each one of these operators had their own advantages and disadvantages. We tabularised these details as follows:



Operator	Advantages	Disadvantages
Classical (Sobel, prewitt, Kirsch,…)	Simplicity, Detection of edges and their orientations	Sensitivity to noise, Inaccurate
Zero Crossing(Laplacian, Second directional derivative)	Detection of edges and their orientations. Having fixed characteristics in all directions	Responding to some of the existing edges, Sensitivity to noise
Laplacian of Gaussian(LoG) (Marr-Hildreth)	Finding the correct places of edges, Testing wider area around the pixel	Malfunctioning at the corners, curves and where the gray level intensity function varies. Not finding the orientation of edge because of using the Laplacian filter
Gaussian(Canny, Shen-Castan)	Using probability for finding error rate, Localization and response. Improving signal to noise ratio, Better detection specially in noise conditions	Complex Computations, False zero crossing, Time consuming

Table: 1. Relative Advantage and Disadvantages of Edge Detection Operators

4. DISCUSSION

The Boolean edge detector performs surprisingly similarly to the Canny edge detector even though they both take drastically different approaches. Canny's method is still preferred since it produces single pixel thick, continuous edges. The Boolean edge detector's edges are often spotty. Colour edge detection seems like it should be able to outperform grayscale edge detectors since it has more information about the image. In the case of the Canny's colour edge detector, it usually finds more edges than the grayscale version. Finding the optimal way to combine the three colour challenges may improve this method. The other colour edge detection scheme we implemented, the Euclidian Distance/Vector Angle detector, did a decent job of identifying the borders between regions, but misses fine grained detail. If the direction of the edge were known, a non-maximal suppression on the colour edge detector's output may help the granularity of its output.

Multi-flash edge detection shows some promise as it strives to produce photographs that will be easy to edge detect, rather than running on an arbitrary image. One problem inherent to the Multi-flash edge detector is that it will have difficulty finding edges between objects that are at almost the same depth or are at depths which are very far away. For example, the Multi-flash method would not work at all on an outdoor scene such as the shoreline.

5. CONCLUSION

Since edge detection is the initial step in object recognition, it is important to know the differences between edge detection techniques. In this paper we studied the most commonly used edge detection techniques of Gradient-based and Laplacian based Edge Detection. Gradient-based algorithms such as the Prewitt filter have a major drawback of being very sensitive to noise.

The size of the kernel filter and coefficients are fixed and cannot be adapted to a given image. An adaptive edge-detection algorithm is necessary to provide a robust solution that is adaptable to the varying noise levels of these images to help distinguish valid image contents from visual artifacts introduced by noise. The performance of the Canny algorithm depends heavily on the adjustable parameters, _, which is the standard deviation for the Gaussian filter, and the threshold values, 'T1' and 'T2'. _ also controls the size of the Gaussian filter. The bigger the value for _, the larger the size of the Gaussian filter becomes. This implies more blurring, necessary for noisy images, as well as detecting larger edges. As expected, however, the larger the scale of the Gaussian, the less accurate is the localization of the edge. Smaller values of _ imply a smaller Gaussian filter which limits the amount of blurring, maintaining finer edges in the image. The user can tailor the algorithm by adjusting these parameters to adapt to different environments. Canny's edge detection algorithm is computationally more expensive compared to Sobel, Prewitt and Robert's operator. However, the Canny's edge detection algorithm performs better than all these operators under almost all scenarios. Evaluation of the images showed that under noisy conditions, Canny, LoG, Sobel, Prewitt, Roberts's exhibit better performance, respectively.

REFERENCE

1. Argyle E. Techniques for edge detection. Proc. IEEE, 1971; 59:285-286.
2. Bergholm F. Edge focusing," in Proc. 8th Int. Conf. Pattern Recognition, Paris, France, 1986; 597- 600,
3. Matthews J. An introduction to edge detection: The sobel edge detector, Available at http://www.generation5.org/content/2002/im01.asp, 2002.
4. Roberts LG. Machine perception of 3-D solids ser. Optical and Electro-Optical Information Processing. MIT Press, 1965 .
5. Gonzalez RC and Woods RE. Digital Image Processing". 2nd ed. Prentice Hall, 2002.

6. Torre V and Poggio TA. On edge detection. IEEE Trans. Pattern Anal. Machine Intell., 1986;8 (2):187-163,
7. Davies ER. Constraints on the design of template masks for edge detection. Pattern Recognition Lett., 1986; 4(11) 1-120.
8. Frei W and Chen CC. Fast boundary detection: A generalization and a new algorithm . lEEE Trans.Comput., 1977;26(10): 988-998.
9. Grimson WE and Hildreth EC. Comments on Digital step edges from zero crossings of second directional derivatives. IEEE Trans. Pattern Anal. Machine Intell., 1985; 7(1): 121-129.
10. Haralick RM. Digital step edges from zero crossing of the second directional derivatives," IEEE Trans. Pattern Anal. Machine Intell. 1984;, 6(1): 58-68.
11. Canny JF. A computational approach to edge detection. IEEE Trans. Pattern Anal. Machine Intell., 1986; 8(6):679-697.
12. Canny J. Finding edges and lines in image. Master's thesis, MIT, 1983.
13. Kirsch RA. Computer determination of the constituent structure of biomedical images. Comput.Eiorned. Res., 1971;4:315-328.
14. Hueckel MH. A local visual operator which recognizes edges and line. J. ACM, 1977; 20(4): 634-647.
15. Yakimovsky Y, Boundary and object detection in real world images. J.ACM, 1976; 23(4): 598-619.
16. Yuille A and Poggio TA . Scaling theorems for zero crossings. IEEE Trans. Pattern Anal. Machine Intell., 1986;8(1):187-163.