

# Performance Analysis of Canny Edge Detector and Laplacian of Gaussian Edge Detector Algorithms

Umesh Dahal<sup>1</sup>, Nawaraj Paudel<sup>2\*</sup>, Jagdish Bhatta<sup>3</sup>

<sup>1</sup>Bhaktapur Multiple Campus, Tribhuvan University

<sup>2</sup>Central Department of Computer Science and Information Technology, Tribhuvan University

<sup>3</sup>Department of Computer Science, Tribhuvan University

## Corresponding Author:

Nawaraj Paudel

Email: [nawarajpauldel@cdcsit.edu.np](mailto:nawarajpauldel@cdcsit.edu.np)

## Abstract:

The core of image processing lies from the principal to represent the digital image in signals of 2-3 dimensions and applying the techniques of digital signal processing to obtain the diverse parameters related to image. Edge detection is the preprocessing techniques of image processing which includes mathematical methods that aims in identifying the sharp discontinuities in an image. The theme of this article is particularized on the performance analysis among Canny Edge Detection and Laplacian of Gaussian (LoG) Edge Detection algorithms on behalf of their execution time. The result in this study shows that the performance of LoG Edge detection algorithm is better than the performance of Canny Edge detection algorithm on the basis of their execution time.

**Key Words:** Canny Edge Detector, LoG Edge Detector, Image Gradient, Kernel, Convolution, Sobel Operator, Prewitt's Operator, Gaussian Smoothing.

## 1. Introduction:

The first stage in many computer vision applications is edge detection. By eliminating unnecessary or unimportant information, edge detection drastically lowers the amount of data and extracts the important information from an image. Object detection in image processing is based on this information (Bhardwaj, 2012).

By eliminating unnecessary or unimportant information, edge detection drastically lowers the amount of data and extracts the important information from an image. Object detection in image processing is based on this information. In computer vision and image processing applications, edge detection is a low-level procedure. Finding and recognizing abrupt changes in an image is the primary objective of edge detection. The sharp variations in pixel intensity that define an object's boundary in a scene are the cause of these discontinuities. Boundaries between various parts in the image are indicated by edges. These boundaries are utilized to distinguish things in order to match and segment them (Kaur, 2011).

There are numerous edge detection operators at one's disposal (Canny, 1986). These operators recognize step, corner, vertical, and horizontal edges. Noise, lighting, objects with similar intensities, and the density of edges in the scene all have a significant impact on the quality of edges that these operators are able to identify. These

issues can be resolved by modifying the threshold settings for what constitutes an edge and a number of other edge detector parameters. These operators are highly susceptible to high frequency contents in edges and noise. Thus, it is necessary to remove noise that could cause distorted and blurry margins.

The main problem with edge detection is determining which image pixel is an edge pixel and which one is not, which can be extremely difficult to do. Thus, the employment of Canny Edge Detector and LoG Edge Detector and their respective performance comparisons are the main objectives of this work.

Edge detection is a process used in image processing to identify and locate sharp discontinuities in an image, which often correspond to object boundaries. Discontinuities are typically identified by sudden changes in pixel intensity, and the goal is to highlight the regions where these changes occur, thus outlining the structure of the objects in the image (Ganesan, 2017).

Steps in edge detection include smoothing, gradient calculation, non-maximum suppression, thresholding. Smoothing reduces noise in the image using techniques like Gaussian smoothing. Noise can create false edges, so smoothing is often applied before edge detection. Gradients represent the change in intensity at a pixel.

High gradients indicate a strong change in intensity, which often means the presence of an edge. The gradient magnitude is calculated, often using techniques like the Sobel or Prewitt operators. After determining the gradient magnitude, the edges must be thinned by eliminating any pixels that are not thought to be edges. This step keeps only the local maxima of the gradient. To distinguish between strong edges and weak edges, a threshold is applied. Only the gradients above a certain value are retained, and others are suppressed. Double thresholding is used in techniques like Canny edge detection to classify edges into strong, weak, and non-edges (Muthukrishnan, 2011).

A basic mathematical procedure called convolution is required for many popular image processing operations, including edge detection. A third array of numbers with the same dimensionality can be created by multiplying two arrays of numbers—typically of different sizes—together using convolution. This can be applied to image processing operations that result in simple linear combinations of certain input pixel values as output values (Jain, 1989). Convolution is accomplished by moving the kernel through all of the locations where it completely fits within the image's bounds by sliding it across the image, usually beginning at the upper left corner. Every single output pixel associated with a kernel position is determined by multiplying the kernel values by the underlying picture pixel value for every single cell in the kernel, and then summing these values collectively. For the image with  $M$  rows and  $N$  columns, and the kernel has  $m$  rows and  $n$  columns, then the size of the output image will have  $M - m + 1$  rows, and  $N - n + 1$  columns.

$$O(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i+k-1, j+l-1)K(k, l),$$

where  $i = 1, \dots, M - m + 1$  and  $j = 1, \dots, N - n + 1$ .

A 2-D convolution operator called the Gaussian smoothing operation is used to blur images and eliminate noise and detail. One kind of picture-blurring filter is the Gaussian blur, which determines the modification to be applied to each pixel in the image using a Gaussian function.

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

Here,

- $\sigma$  is the standard deviation of the Gaussian distribution,
- $x$  is the horizontal axis distance from the origin,
- $y$  is the vertical axis distance (where  $y = 0$  for one dimension).

This formula, when used in two dimensions, yields a surface with concentric circles as contours and a Gaussian distribution from the center point. A convolution matrix is constructed using values from this distribution and

applied to the original image.

An image processing method called edge detection is used to identify the borders of objects in pictures. It operates by looking for brightness discontinuities. In fields including image processing, computer vision, and machine vision, edge detection is utilized for data extraction and image segmentation. There are numerous approaches to edge detection. The majority, however, can be divided into two groups: gradient and Laplacian. By locating the maximum and lowest in the image's first derivative, the gradient approach finds the edges. To locate edges, the Laplacian approach looks for zero crossings in the image's second derivative.

The input image is convolved in first order derivative by means of an adaptive mask, producing a gradient image where edges are identified using thresholding. The majority of first-order derivative operators are the classical operators, such as Sobel, Prewitt, and Robert. Gradient operators are another name for these operators. These gradient operators search for the highest and lowest intensity values in order to identify edges. These operators decide whether to classify a particular pixel as an edge by looking at the distribution of intensity values nearby. These operators cannot be employed in real-time applications due to their longer computation times.

The extraction of zero crossing points, which suggests the existence of maxima in the image, is the foundation for second order derivatives. Here, an adaptive filter is used to first smooth the image (Kumar Singh et al., 2014). Since noise greatly affects the second order derivative, the filtering function plays a crucial role. These operators are proposed by the authors in (Marr, 1980) and are derived from the Laplacian of a Gaussian (LoG). Here, a Gaussian filter is used to smooth the image. One major issue with LoG is that the localization of asymmetrically profiled edges by zero-crossing points generates a bias that gets worse as filtering has a smoothing impact (Akhtar et al., 2012).

Canny (Canny, 1986) offered an intriguing solution to this issue, stating that an ideal operator for step edge detection should meet three requirements: good detection, good localization, and only one reaction to a single edge. When recognizing edge types other than those for which they are optimal, these operators have some limitations regarding localization, although they nevertheless perform well against noisy images (Joshi, 2012).

A gradient-based edge detector for the spatial domain is the Sobel edge detector. Two gradient masks, each measuring  $3 \times 3$ , make up the Sobel operator: one along the horizontal direction and another along the vertical direction. The two masks move across the picture in an attempt to compute the gradient at every pixel in the two-dimensional grayscale image (Jain, 1989; Joshi, 2012).

The Prewitt operator removes an image's edges exclu-

sively in the vertical and horizontal directions, just like the Sobel operator does. The procedures in the detecting process are the same as those used by the Sobel operator. There are two masks in this operator. The image's absolute gradient is created by convolving the masks with the image. The square of the absolute gradients' magnitudes (along the x and y axes) at a given pixel indicates the edge strength there (Jain, 1989)(Joshi, 2012).

Among the most well-known and established edge detection operators is Robert's Cross Operator. Every pixel in the image under consideration has its spatial gradient determined by the operator (Jose, 2013). The output edge image is the absolute magnitude of the gradient at every single pixel in the input image. This operator is substantially faster and more straightforward than the Prewitt and Sobel operators.


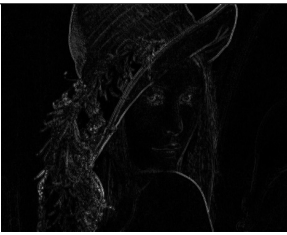






## 2. Methods And Methodology:

The main exploration of this work focuses on determining

the best edge detection on behalf of their performance. A sample image will be fed to both the edge detection algorithms. Output of the sample images from both the algorithms were analyzed in a quantitative approach. Finally, conclusion was drawn by comparing their performances on the basis of execution time.

The Table 1 shows the detail of the images used. The images used for the verification of this research consists of almost all the variations required to assure the performance analysis of these algorithms effectively. Sample images used for the testing purpose are Leena, TU logo, building, and TU cricket ground. The resulting image after edge detection (whether using Canny or LoG) is a grayscale image where the edges in the input image are highlighted. The output images are also in the right side of the Table 1. The output images depict huge variations with respect to their edges among one another. Such testing verifies the flexibility of the algorithm.

Table 1: Images used for verification of algorithms

Name	Pixel	Input Image	Output Image
Image 1	960×658		
Image 2	2362×2650		
Image 3	814×1080		
Image 4	1248×935		

### LoG Edge Detector:

The Laplacian of Gaussian (LoG) Edge Detector is another popular method for detecting edges in images. It combines Gaussian smoothing and the Laplacian opera-

tor to detect edges, particularly where there is a rapid intensity change. LoG is especially useful for detecting edges in noisy images because the Gaussian smoothing step helps reduce noise before applying the Laplacian for edge detection (Shang, 2012).

First, a Gaussian filter is applied to smooth the image and reduce noise. The Gaussian function is represented as:

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

Here,  $\sigma$  controls the level of smoothing. The larger  $\sigma$ , the more smoothing is applied.

After smoothing, the Laplacian operator is applied to the smoothed image to find regions of rapid intensity change. The Laplacian is a second-order derivative operator that highlights areas of the image where the intensity changes sharply (edges). In 2D, the Laplacian is defined as:

$$\nabla^2 I(x, y) = \frac{d^2 I}{dx^2} + \frac{d^2 I}{dy^2}$$

This operation is typically applied to the smoothed image, and it detects edges by finding zero-crossings in the Laplacian of the image. Zero-crossings occur where the intensity changes from positive to negative or vice versa, indicating the presence of an edge.

After applying the Laplacian, the algorithm looks for zero-crossings in the filtered image. These zero-crossings indicate the locations of edges. The idea is that an edge corresponds to where the second derivative of the intensity values goes from positive to negative (or vice versa). The LoG operator is the Laplacian applied to a Gaussian-smoothed image, often written as:

$$\log(x, y) = \nabla^2 [G(x, y) * I(x, y)]$$

Where:

- $G(x, y)$  is the Gaussian filter,
- $I(x, y)$  is the input image,
- $\nabla^2$  is the Laplacian operator,
- $*$  denotes the convolution operator.

### Canny Edge Detection:

The Canny Edge Detector identifies edges by finding areas of the image where the intensity of pixels changes sharply.

It aims to detect only real edges and avoid detecting false edges (noise), making it both effective and efficient (Canny, 1986)(Ansari et al., 2017). The steps that must be followed are (Pirzada, 2013).

Before detecting edges, noise in the image is reduced by applying a Gaussian blur. This helps in smoothing the image and preventing false edge detection caused by noise.

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

Here,  $\sigma$  is the standard deviation of the Gaussian distribution.

After smoothing the image, the algorithm calculates the gradient magnitude and direction at each pixel using edge detection operators like the Sobel operator. The gradient is found in both the horizontal ( $G_x$ ) and vertical ( $G_y$ ) directions.

$$G = \sqrt{G_x^2 + G_y^2}$$

The angle or direction of the gradient is

$$\theta = \tan^{-1} \left( \frac{G_y}{G_x} \right)$$

The gradient image is further processed to thin out the edges. Non-maximum suppression examines each pixel to determine if it is a local maximum in the gradient direction. This step sharpens the edges by suppressing all non-edge pixels.

The double threshold step is used to classify pixels as strong, weak, or non-edges based on their gradient magnitude.

Finally, weak edges that are connected to strong edges are kept, while the rest are discarded. This step connects edge segments and ensures that only continuous, meaningful edges are retained.

### Experimental Setup:

Both algorithms were implemented using Java Programming Language (JDK 21), OpenCV 4.10.0 computer vision library, Apache NetBeans IDE 22, Microsoft Windows 10 Pro, Intel® Core™ i5-10210U CPU @ 1.60 GHz 2.11 GHz, and 8 GB RAM.

## 3. Results And Discussion:

Both algorithms were tested for all four sample images given in Table 1 above. For each sample image, five executions were performed in order to obtain the mean value. Both LoG edge detection and Canny edge detection algorithms performance were measured in nanoseconds.

For all four images, time taken by the Gaussian blur is calculated before applying both edge detection algorithms. Gaussian blur plays a crucial role in edge detection by reducing noise and smoothing the image before applying edge detection. The time taken by the Gaussian blur are 11968500, 11451700, 27887400, and 15336900 nanoseconds respectively for image 1, image 2, image 3, and image 4 respectively. The figure below shows Gaussian blur execution time for all four images.

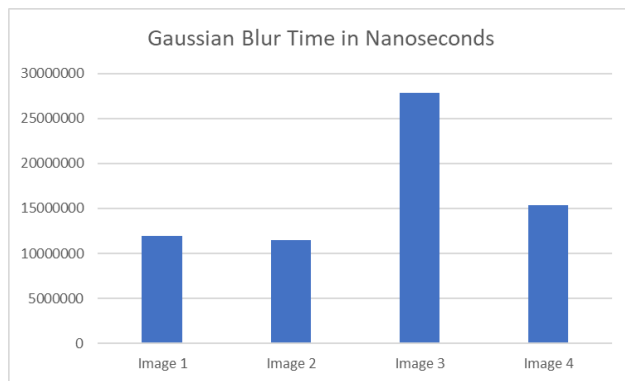


Figure 1: Gaussian blur time in nanoseconds for all images

Table 2: Execution time in nanoseconds for Image 1

Observation	Canny	LoG
1	2515600	1058400
2	2460700	786600
3	1864400	806200
4	1831500	922400
5	1785700	646000
<b>Average Time</b>	<b>2091580</b>	<b>843920</b>

The Figure 2 demonstrates that, in all five observations, the execution time required by the LoG Edge detection technique is less than that of the canny edge detection approach. The LoG Edge Detection algorithm takes 843920 nanoseconds, and the Canny Edge Detection algorithm takes 2091580 nanoseconds on average to process five observations for Image 1.

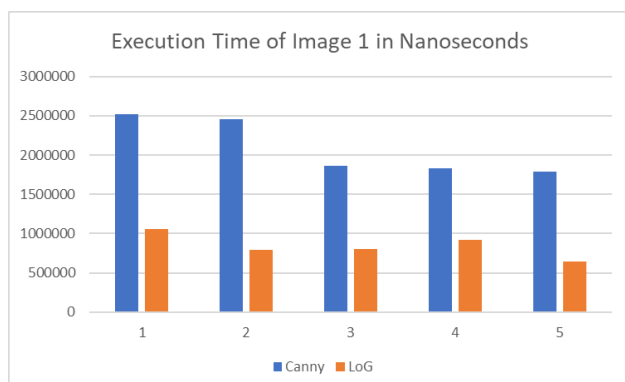


Figure 2: Execution time in nanoseconds for Image 1

Table 3: Execution time in nanoseconds for Image 2

Observation	Canny	LoG
1	10802700	6706600
2	14656300	7882600
3	11717000	7721000
4	12071300	8507600
5	10990500	7922200
<b>Average Time</b>	<b>12047560</b>	<b>7748000</b>

The Figure 3 demonstrates that, in all five observations, the computational time required by the LoG Edge detection technique is less than that of the canny edge detection approach. The LoG Edge Detection algorithm takes 7748000 nanoseconds, and the Canny Edge Detection algorithm takes 12047560 nanoseconds on average to process five observations for Image 2.

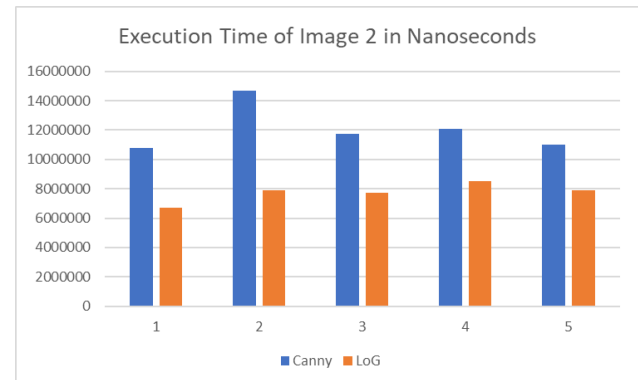


Figure 3: Execution time in nanoseconds for Image 2

Table 4: Execution time in nanoseconds for Image 3

Observation	Canny	LoG
1	7033900	2892500
2	6744500	3127400
3	6739900	3229300
4	7033000	2245500
5	3502100	1651300
<b>Average Time</b>	<b>6210680</b>	<b>2629200</b>

The Figure 4 demonstrates that, in all five observations, the computational time required by the LoG Edge detection technique is less than that of the canny edge detection. The LoG Edge Detection algorithm takes 2629200 nanoseconds, and the Canny Edge Detection algorithm takes 6210680 ns on average to process five observations for Image 3.

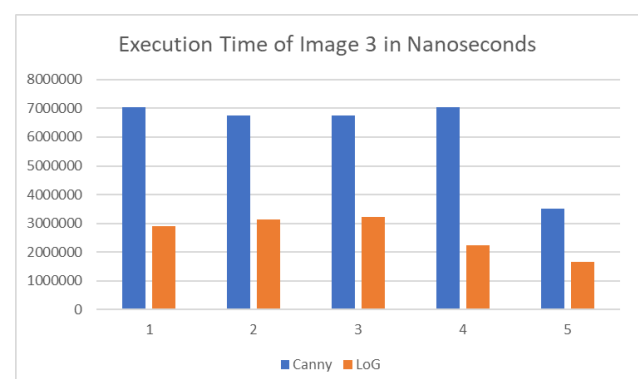


Figure 4: Execution time in nanoseconds for Image 3

Table 5: Execution time in nanoseconds for Image 4

Observation	Canny	LoG
1	6352500	2443700
2	7381400	2688300
3	9985200	3144000
4	10516800	3054600
5	11564700	3315500
<b>Average</b>	<b>9160120</b>	<b>2929220</b>

Figure 5 demonstrates that, in all five observations, the computational time required by the LoG edge detection technique is less than that of the Canny edge detection approach.

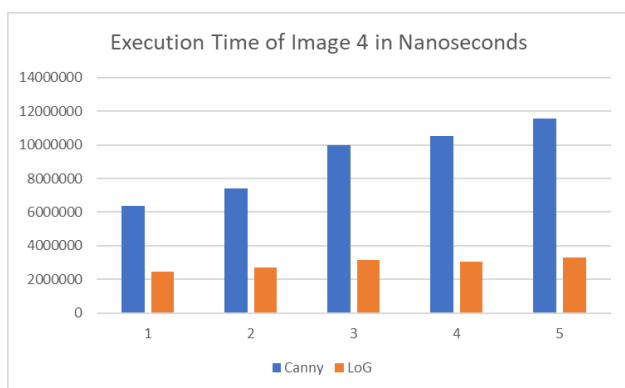


Figure 5: Execution time for Image 4

Table 6: Average execution time for all images

Image	Canny	LoG
Image 1	2091580	843920
Image 2	12047560	7748000
Image 3	6210680	2629200
Image 4	9160120	2929220

Figure 6 shows average execution time for both Canny and LoG edge detection algorithms across four images.

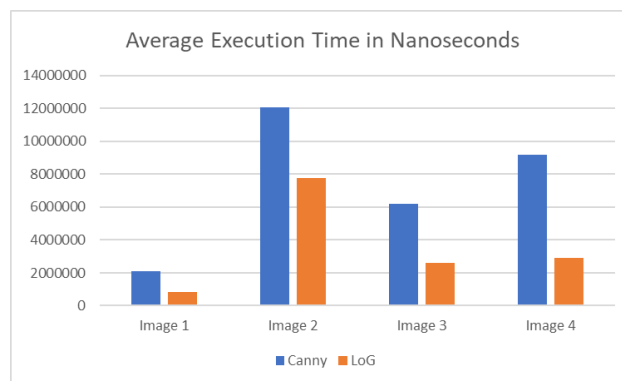


Figure 6: Average execution time for all four images

#### 4. Conclusion:

Based on all the above observations, this study clearly shows that the performance of LoG edge detection algorithm is better than the performance of Canny edge detection algorithm. The execution time of LoG edge detection is less than that of Canny edge detection. Based on the execution time, the LoG edge detection algorithm outstands Canny edge detection algorithm. While the approach is rather efficient at returning results in the shortest amount of time, it might still be improved by concentrating on areas such as the Gaussian filter, gradient calculation techniques, picture noise reduction techniques, and so on.

Edge detection is a critical topic in image processing and computer vision, used to identify object boundaries within an image. Edge detection algorithm with higher speed can be used across various domains such as object detection and recognition, medical imaging, robotics and autonomous systems, image compression, satellite and aerial imaging, augmented reality and gaming, industrial quality control, security and surveillance.

This research can be explored further with deep learning integration to learn edge features and to improve robustness to noise, lighting changes and texture variations. Quantum computing can also be explored to achieve unprecedented processing speeds for edge detection in large datasets.

#### 5. Funding Statement:

This study did not receive funding from any specific grant or funding agency.

#### 6. Completing Interest:

The authors confirm that they have no competing interests to declare.

#### 7. Acknowledgment:

The authors would like to thank all the respondents for participating in this study.

## References

Akhtar, S., Rajyalakshmi, D., & Sattar, S. A. (2012). A theoretical survey for edge detection techniques and watershed transformation. *International Journal of Computer Technology and Electronics Engineering*.



<https://www.yumpu.com/en/document/read/29330468/a-theoretical-survey-for-edge-detection-techniques-and-watershed->

- Ansari, M. A., Kurchaniya, D., & Dixit, M. (2017). A comprehensive analysis of image edge detection techniques. *International Journal of Multimedia and Ubiquitous Engineering*, 12(11). <https://doi.org/10.14257/ijmue.2017.12.11.01>
- Bhardwaj, A., S. & Mittal. (2012). A survey on various edge detector techniques. *Procedia Technology*, 4. <https://doi.org/10.1016/j.protcy.2012.05.033>
- Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6). <https://doi.org/10.1109/TPAMI.1986.4767851>
- Ganesan, G., P. & Sajiv. (2017). A comprehensive study of edge detection for image processing applications. *International Conference on Innovations in Information, Embedded & Communication Systems, ICIIECS 2017*. <https://doi.org/10.1109/ICIIECS.2017.8275968>
- Jain, A. K. (1989). *Fundamentals of digital image processing*. Prentice-Hall. <http://ultra.sdk.free.fr/docs/DxO/Fundamentals%20of%20Digital%20Image%20Processing.pdf>
- Jose, C. S., A. & Seelamantula. (2013). Bilateral edge detectors. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. <https://doi.org/10.1109/ICASSP.2013.6637891>
- Joshi, R., S. R. & Koju. (2012). Study and comparison of edge detection algorithms. *Asian Himalayas International Conference on Internet*. <https://doi.org/10.1109/AHICI.2012.6408439>
- Kaur, A., B. & Garg. (2011). Mathematical morphological edge detection for remote sensing images. *ICECT 2011 - 2011 3rd International Conference on Electronics Computer Technology*, 5. <https://doi.org/10.1109/ICECTECH.2011.5942012>
- Kumar Singh, R., Shekhar, S., Bhawan Singh, R., & Chauhan, V. (2014). A comparative study of edge detection techniques. *International Journal of Computer Applications*, 100(19). <https://doi.org/10.5120/17631-5949>
- Marr, E., D. & Hildreth. (1980). Theory of edge detection. *Proceedings of the Royal Society of London - Biological Sciences*, 207(1167). <https://doi.org/10.1098/rspb.1980.0020>
- Muthukrishnan, M., R. & Radha. (2011). Edge detection techniques for image segmentation. *International Journal of Computer Science and Information Technology*, 3(6). <https://doi.org/10.5121/ijcsit.2011.3620>
- Pirzada, A., S. J. H. & Siddiqui. (2013). Analysis of edge detection algorithms for feature extraction in satellite images. *International Conference on Space Science and Communication, IconSpace*. <https://doi.org/10.1109/IconSpace.2013.6599472>
- Shang, F., J. & Jiang. (2012). An algorithm of edge detection based on soft morphology. *International Conference on Signal Processing Proceedings, ICSP*, 1. <https://doi.org/10.1109/ICoSP.2012.6491626>