

*Received Date: June 2023 Revised: August 2023 Accepted: October 2023*

---

## **Performance Evaluation of Different Images Using Edge Detection Algorithms**

- Chhetra Bahadur Chhetri
- Manish Pandey(BCA 6<sup>th</sup> sem)

### **Abstract**

To determine which edge detection method performs best and worst on different image types, numerous edge detection algorithms are examined. For the performance analysis, some sample photos from the web and some from Java are used as sources. The entropy and signal noise ratio are used to gauge how well the edged image performs. In image processing, conducting a thorough investigation of various edge detection techniques is highly worthwhile two widely used edge detection algorithms Log, and Canny—are taken into consideration in this analysis. Here in this paper, the analysis is focused on the performance of different edge detector algorithms. All candidate algorithms of edge detection are implemented in JAVA. The result of empirical performance shows that two variants namely canny perform better results for the edge detection algorithm. The result shows that when considering only the performance aspect. Cycle/byte is calculated for comparing different variants. Cycle/byte is decreased when the canny edge detector is examined. The canny edge detection algorithm shows a better performance than LoG. LoG has more than 3 times higher cycle/byte than Canny Edge detection.

Keywords: Canny Edge detection, Log Edge detection, cycle/byte, image processing, empirical performance

### **1. Introduction**

Edge detection is a method of image processing that locates the edges of objects in pictures. It operates by looking for changes in brightness. In fields including image processing, computer vision, and machine vision, edge detection is utilized for image segmentation and data extraction.

There are numerous approaches to edge detection. The majority, however, can be divided into two categories: Laplacian and gradient. By looking for the greatest and minimum in the first derivative of the image, the gradient approach finds the edges. The Laplacian approach uses the second derivative of the image to find edges by looking for zero crossings. To create a gradient image in which edges are recognized by thresholding, the input image in the first-order derivative is convolved with a customized mask. The initial stage in many computer vision applications is edge detection. Edge detection drastically minimizes the data and offers the important information in a picture by filtering out irrelevant or undesired data. In image processing, these details are utilized to identify objects. Edge detection is a basic procedure used in computer vision and image processing software. In digital images, it is crucial to identify the features of the images. Just like humans, algorithms should pay attention to these features in order to perceive images. Edge detection application is a visual processing technique used at this point. Edge detection's primary objective is to find and recognize distinct discontinuities in an image. These breaks are brought on by sharp variations in pixel intensity that define the borders of objects in a scene. Edges define the boundaries between the image's various areas. For the purposes of segmentation and matching, these boundaries are employed to identify objects (Bhardwaj & Mittal, 2012). Many of these processes start with these object boundaries. Edge detection is an important and basic operation to be completed for any image processing activities, image analysis, and pattern recognition on various images such as satellite images, medical images, etc.

## **2. Literature review**

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision. There are many ways to perform edge detection. However, most may be grouped into two categories, gradient and Laplacian. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. The Laplacian method searches for zero crossings in the second derivative of the image to find edges. (Bhardwaj & Mittal, 2012) The first-order derivative creates a gradient image in which edges are recognized by thresholding by convolving the input image with a modified mask. The majority of traditional operators, including Sobel, Prewitt, and Robert, are first-order derivative operators.

Also known as gradient operators, these operators. By looking for the greatest and minimum intensity values, these gradient operators find edges. These operators decide whether a given pixel should be labeled as an edge by looking at the distribution of intensity values in its immediate vicinity. Since they need additional computing time, these operators cannot be applied in real time.

These are based on the extraction of zero crossing points in second-order derivatives, which show the presence of maxima in the image. First, an adaptive filter is used to smooth the image (J.F, 1986). Since the filtering function is crucial since the second-order derivative is highly sensitive to noise. These operators, developed by Marr and Hildreth, are derivations of the Laplacian of a Gaussian (LOG), in which a Gaussian filter is used to smooth the image. We must fix a few parameters for this operator, such as the Gaussian filter's variance and thresholds. Although there are some ways for their automatic calculating (Meghana & G.K, 2012), their values must often be set by the user. The localization of edges with an asymmetric profile by zero-crossing points generates a bias, which worsens with the smoothing effect of filtering, which is a significant issue with LoG. Canny offered an intriguing solution to this issue, stating that the ideal operator for step-edge detection meets the following three criteria: good detection, good localization, and only one reaction to a single edge. Other operators have since been suggested. These operators have strong noise immunity, but they have certain localization limitations when detecting edge types other than their preferred ones. Finally, we draw the conclusion that, as a result of the blurring, none of the actual edge detectors based on the first or second derivative of a picture satisfy our criterion. (Joshi & Koju, 2012)

## **2.1 Sobel Operator**

The Sobel edge detector is a spatial domain gradient-based edge detector. The Sobel operator consists of two gradient masks of size  $3 \times 3$  one along the horizontal direction and another along the vertical direction. The pair of masks slides over the image and aims to calculate the gradient at every single pixel of the 2D grayscale image. The Sobel operator is a simple edge detector and allows fast computation. The detection process performs a small amount of mathematical calculation and hence makes the detection process computationally cheap. However, the Sobel operator has a limited capability to detect the edges in an arbitrarily oriented direction. Due to this limited directional capability, it only extracts the edge direction only along the horizontal

and vertical direction. Besides, this edge detector is also not robust to noise. Hence it has limited application where the image is noisy and rich in edges along different directions.

### 2.2 Prewitt's Operator

Like the Sobel operator, the Prewitt operator also extracts the edges of an image only along vertical and horizontal directions. The detection process follows the same steps as followed by the Sobel operator. This operator consists of a pair of masks. The masks are convolved with the image to produce the absolute gradient of the image. The edge strength at a particular pixel is given by the square of the magnitude of the absolute gradients (along the x and y axis) at the same point. Like the Sobel operator, the Prewitt edge detector also adopts only limited directions and hence it has a limited application.

### 2.3 Robert's Cross Operator

It is one of the oldest and most popular edge detection operators. The operator calculates the spatial gradient at each and every pixel of the image under consideration. The absolute magnitude of the gradient at each single pixel of the input image is the resultant edge image. Unlike Prewitt and Sobel operator this operator is quite simple and much faster. Due to its fast computation and easy inclusion, it has a frequent application in hardware implementation.

### 2.4 LoG edge detector

The LoG (Laplacian of Gaussian) edge detector exploits the second-order derivatives of pixel intensity to locate edges. The Laplacian  $L(x, y)$  of an input image  $I(x, y)$  is given by

$$L(x, y) = \frac{d^2I}{dx^2} + \frac{d^2I}{dy^2} \dots\dots\dots(3.1)$$

and, can be computed by convolving the image with any of the convolution mask. This operator is sensitive to noise and is often applied to the image after it has been smoothed with Gaussian smoothing filter. The Gaussian smoothing filter  $G_1$  is defined by

$$\text{LoG}(x, y) = -\frac{1}{\pi\sigma^4} \left[ 1 - \frac{x^2+y^2}{2\sigma^2} \right] e^{-\frac{x^2+y^2}{2\sigma^2}} \dots\dots\dots(3.2)$$

Convoluting the image with any one of the smoothing kernels having a different value will result in the Gaussian smoothing.

This operator locates the edges in the second-derivative maps by fusing the zero-crossing points and the image-gradient magnitude. To locate the edge's subpixel location, linear interpolation is used. The standard deviation of the Gaussian smoothing filter employed in the LoG filter has a significant role in the behavior of the LoG edge detector. The Gaussian filter is wider and the smoothing is greater the higher the value. An excessive amount of smoothing could make edge identification challenging. Edges can be viewed as gradient points with a high intensity. (Bhardwaj & Mittal, 2012)

### 2.5 Canny edge detector

The list of requirements to enhance edge detection came next. The low error rate comes first, which means that true edges shouldn't be overlooked. The second is the localization of the edge point. The smallest distance between the edge pixels identified by the detector and the real edge pixels is that. The third is to just have one response to one edge. Canny edge detector implementation requires a sequence of stages.

Step 1: Prior to detecting the edge, the image's noise was first filtered out. This challenge makes use of the Gaussian filter. Convolution can be used to accomplish Gaussian smoothing. A smaller-sized mask should be able to be moved over a picture and be adjusted one square of pixels at a time. Mask width must be carefully chosen because it directly relates to localization mistakes.

Step 2: Edge strength is find out by taking the gradient of the image. A Robert mask or a Sobel mask can be used for this purpose .The magnitude of gradient is approximated using the formula,

$$|G| = \sqrt{G_x^2 + G_y^2} \dots\dots\dots(3.3)$$

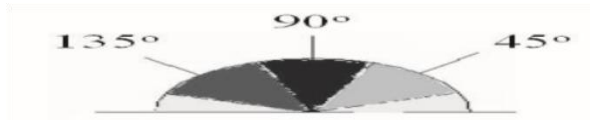
$$|G| = |G_x| + |G_y| \dots\dots\dots(3.4)$$

where  $G_x$  and  $G_y$  are the gradient in X and Y direction respectively.

Step 3: Find the edge direction by using the gradient in x and y directions. Formula used is

$$\alpha = \arctan\left(\frac{|G_x|}{|G_y|}\right) \dots \dots \dots (3.5)$$

Step 4: After knowing the edge direction relate it to the specific degree. Resolve the edge direction in horizontal, positive, vertical, negative diagonal (Chandra Sekhar & Abin, 2013)



Step 5: Use non-maxima suppression to suppress any pixel value that isn't regarded as an edge by tracing along the edge direction. It provides a fine edge line.

Step 6: Use double / hysteresis thresholding to eliminate streaking. (Bhardwaj & Mittal, 2012)

### 3. Measuring Cost

There is some extra cost that may be added to the absolute cost for creating edges using different edge detector algorithms but this is equally affected by all candidate's algorithms on the execution. The system time in nanoseconds is taken just before the execution of the code segment for generating edges in each algorithm and the completion of the execution. The time spent for creating the edge is calculated by subtracting the start time taken before execution from the completion time taken after completing the execution of a specific code segment that is used to produce the edge. The time required for each algorithm is calculated as follows:

```
long    startTime = System.nanoTime();
// canny.cannyedge();
//and logedge.logedge();
long    timeRequired = System.nanoTime() - startTime;
```

Various processes may be run in the background of the system so absolute measurement may not be measured due to this reason, the time needed for creating an edge in all algorithms may not be observed in every run of program. Therefore at least 5 times, the program implemented in java is run in architecture describes as section below and finally average required time observed in every run is calculated as:


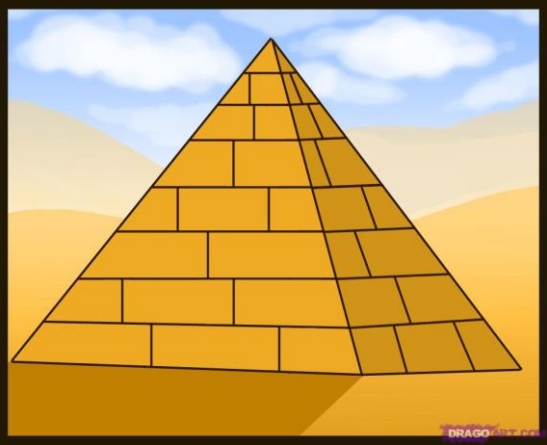
$$\text{Average required Time} = \sum_{i=1}^5 \frac{T_i}{5} \quad \text{where } T_i \text{ represent time obtained in } i^{\text{th}} \text{ run of execution.}$$



This average calculated time is used to calculate cycle per byte.

## 4. Measuring Performance

### 4.1 Data Collection

All the secondary data (image) are extracted from the internet. Table below shows the detail of the image used.

| S.N | Name              | Pixel     | Figure  |
|-----|-------------------|-----------|---|
| 1.  | Sample Image<br>1 | 512×512   |   |
| 2.  | Sample image<br>2 | 1000×1000 |  |

|    |                   |         |   |
|----|-------------------|---------|---|
| 3. | Sample image<br>3 | 250×250 |   |
| 4. | Sample image<br>4 | 512×512 |  |

#### 4.1.1 Testing Data:

After the implementation of the algorithms and selection of proper data set, the modules will be tested using two algorithms Canny Edge Detector and LoG Edge Detector algorithm to get the best result.

### 4.2 Experiment & Result

#### 4.2.1 Experimental Setup

The aim is to experimentally determine the best algorithm among Canny Edge Detection and LoG Edge Detection.



The algorithms used for various flavors were implemented in JAVA. The candidate's algorithms were executed on J2SETM (Java™ Platform, Standard Edition 17 Development Kit) environment with Windows10, 64 bits machine having 8GB RAM, with Intel CORE™ i7 processor.

The sample image mentioned in Figure 5.1 were tested for both the algorithms ( Canny and LoG). Each sample was executed 5 times to give the output as a mean value. The performance of Canny Edge detection and LoG Edge Detection were calculated in terms of time complexity (nanosecond). The result of the experiment is shown as follows:

#### 4.2.2 Evaluation

##### 4.2.2. (a) Sample Image1



#### Results:

| No. of observation        | Canny Edge (Execution time in ns) | LoG Edge(Execution time in ns) |
|---------------------------|-----------------------------------|--------------------------------|
| 1                         | 925571902                         | 3577939702                     |
| 2                         | 797692462                         | 3032482393                     |
| 3                         | 830369636                         | 3097272533                     |
| 4                         | 717162871                         | 2696814838                     |
| 5                         | 839265043                         | 3209588811                     |
| <b>Average Time in NS</b> | 822012382.8                       | 3122819655                     |

Table 4.2.2.(a): Computational Time for the given Sample Image 1

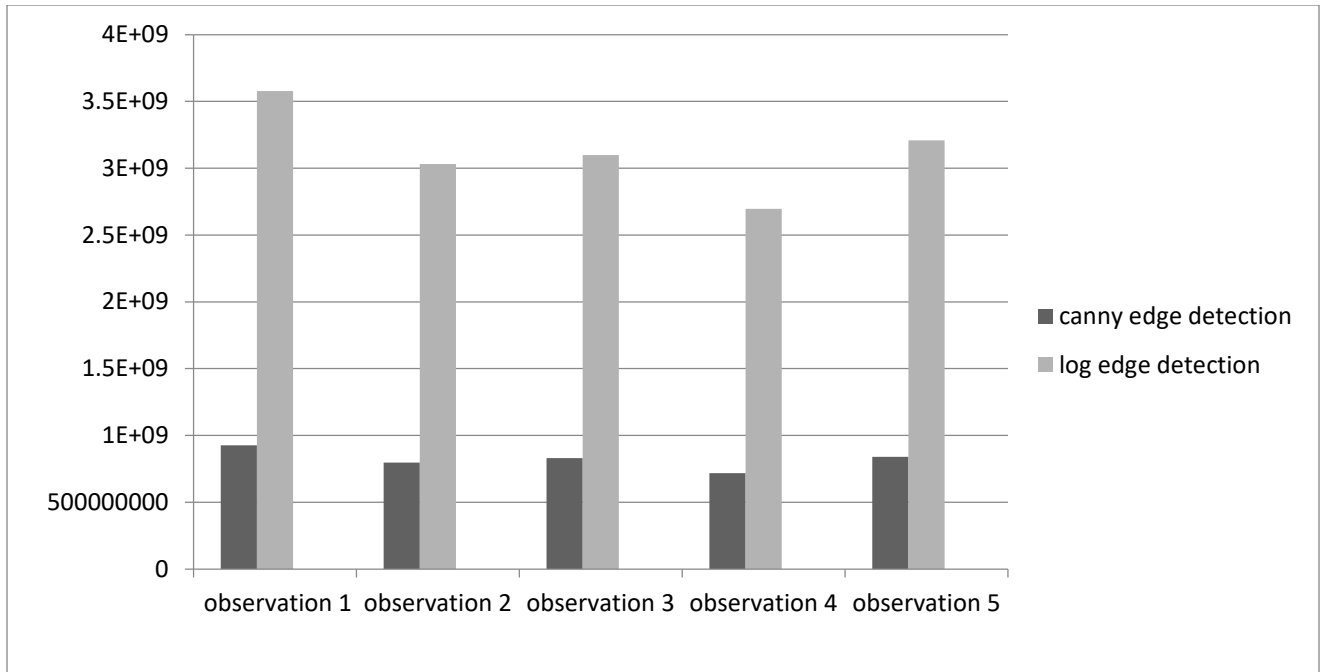
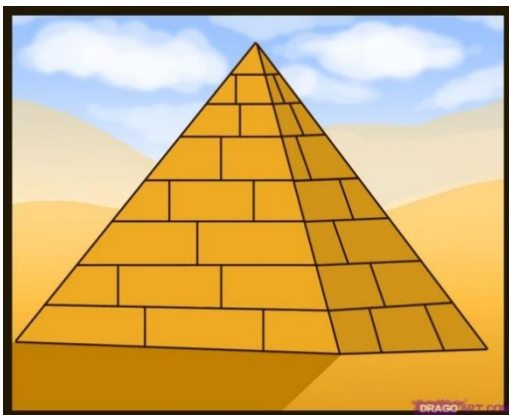


Figure 4.2.2.(a): Computational Time for the given Sample Image 1

The above graph shows that the computational time taken by canny edge detection is lesser in all five observations than that of LoG Edge detection algorithm. The average computational time for five observations required for sample image 1 by Canny Edge Detection algorithm is 822012382.8 ns and LoG Edge Detection algorithm is 3122819655 ns.

#### 4.2.2.(b) Sample Image2



## Result

| No. of observation        | Canny Edge (Execution time in ns) | LoG Edge(Execution time in ns) |
|---------------------------|-----------------------------------|--------------------------------|
| 1                         | 1536459171                        | 7686080108                     |
| 2                         | 1566676516                        | 8177579810                     |
| 3                         | 1548296837                        | 7627500357                     |
| 4                         | 1469454116                        | 7556601738                     |
| 5                         | 1514410708                        | 7626201817                     |
| <b>Average Time in ns</b> | <b>1527059470</b>                 | <b>7734792766</b>              |

Table 4.2.2.(b): Computational Time for the given Sample Image 2

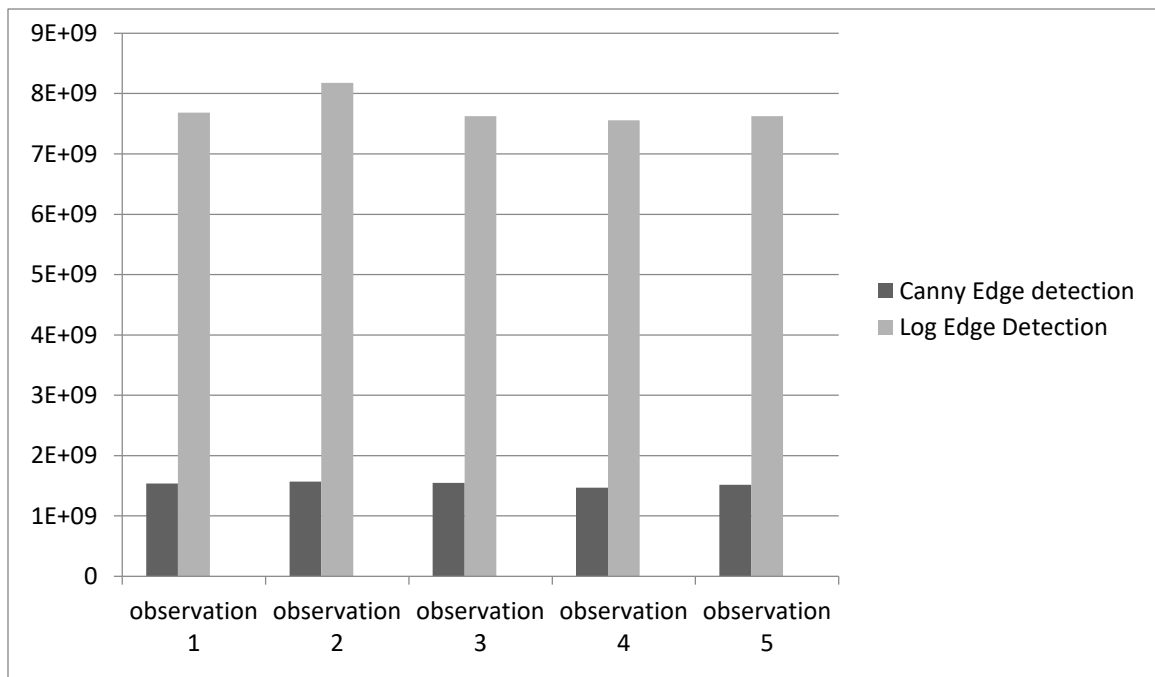


Figure 4.2.2.(b): Computational Time for the given Sample Image 2

The above graph shows that the computational time taken by canny edge detection is lesser in all five observations than that of LoG Edge detection algorithm. The average computational time for five observations required for sample image 1 by Canny Edge Detection algorithm is 1527059470 ns and LoG Edge Detection algorithm is 7734792766 ns.

#### 4.2.2. (c): Sample Image3



#### Result

| No. of observation        | Canny Edge (Execution time in ns) | LoG Edge(Execution time in ns) |
|---------------------------|-----------------------------------|--------------------------------|
| 1                         | 691395901                         | 2802071746                     |
| 2                         | 830203210                         | 2744704348                     |
| 3                         | 788477762                         | 2960011556                     |
| 4                         | 812946604                         | 2811954207                     |
| 5                         | 720794674                         | 2822784685                     |
| <b>Average Time in ns</b> | 768763630.2                       | 2828305308                     |

Table 4.2.2.(c): Computational Time for the given Sample Image 3

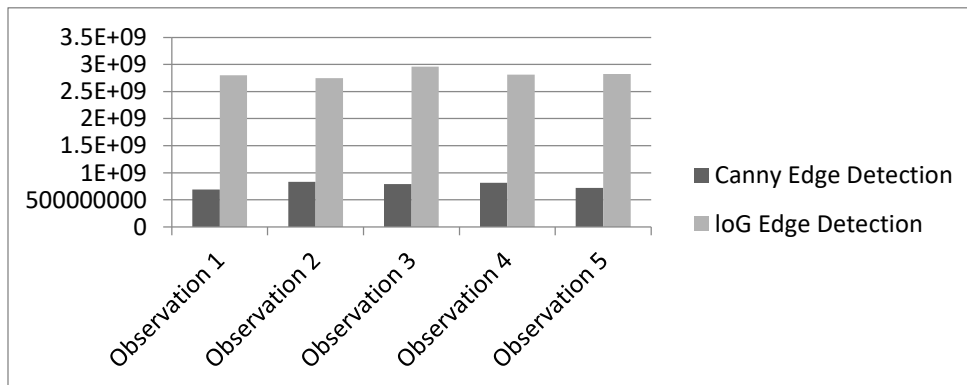
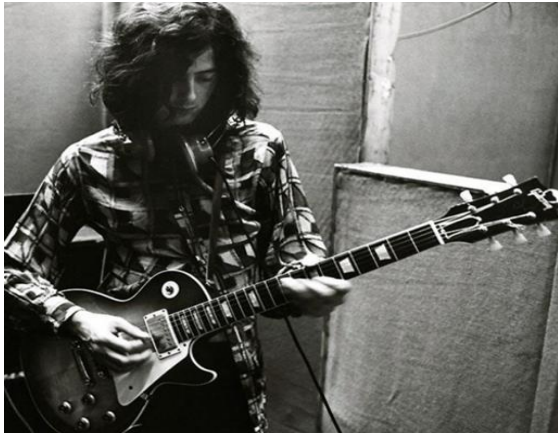


Figure 4.2.2.(c): Computational Time for the given Sample Image 3

The above graph shows that the computational time taken by canny edge detection is lesser in all five observations than that of LoG Edge detection algorithm. The average computational time for five observations required for sample image 1 by Canny Edge Detection algorithm is 768763630.2 ns and LoG Edge Detection algorithm is 2828305308 ns.

**4.2.2.(d) Sample Image4**



**Result**

| No. of observation        | Canny Edge (Execution time in ns) | LoG Edge(Execution time in ns) |
|---------------------------|-----------------------------------|--------------------------------|
| 1                         | 817099876                         | 2886975689                     |
| 2                         | 887798371                         | 2838352367                     |
| 3                         | 735308730                         | 2716432238                     |
| 4                         | 817987485                         | 2844300993                     |
| 5                         | 798767044                         | 2843363662                     |
| <b>Average Time in ns</b> | 811392301.2                       | 2825884990                     |

Table 4.2.2.(d): Computational Time for the given Sample Image 4

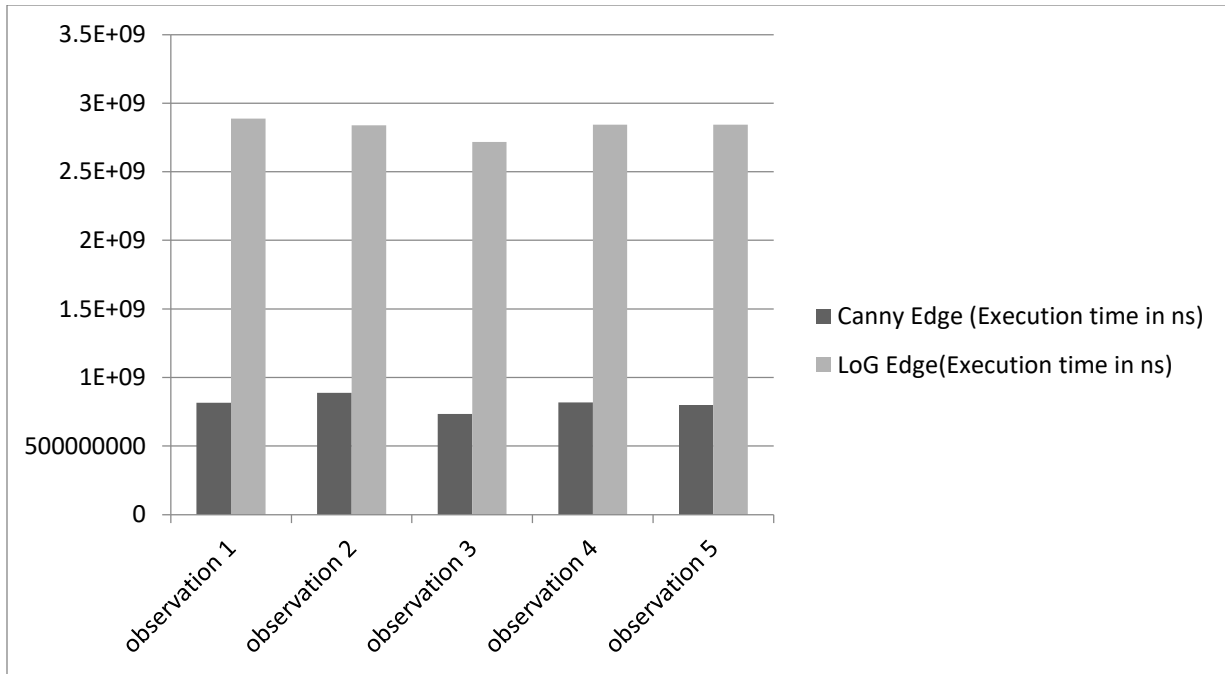


Figure 4.2.2.(d): Computational Time for the given Sample Image 4

The above graph shows that the computational time taken by canny edge detection is lesser in all five observations than that of LoG Edge detection algorithm. The average computational time for five observations required for sample image 1 by the Canny Edge Detection algorithm is 811392301.2 ns and LoG Edge Detection algorithm is 2825884990 ns.

## 5. Result

For every candidate image canny edge detection algorithm seems to be the better edge detector variant whatever the size of the image is taken to generate the edge. However, the nature of time required to detect edge remain unchanged according to input size of image and types of image. Canny seems to be best model in edge detection algorithm by this way. It is observed that Canny yields 19%-28% better performance (cycle/byte) than LoG edge detector when various kinds of image were tested. For comparing two edge detection algorithms, Canny and LoG, it seems Canny algorithm is best to use for every types of image. LoG has more than 3 times higher cycle/byte than Canny.

## 6. Conclusion

From the analysis, it is identified that the Canny edge detection algorithm is performing better among the two algorithms. Out of the four image information, Canny algorithm on Dither binary image information yields the high entropy and Signal-to-Noise Ratio values. However, the LoG algorithm with indexed image information generates very low entropy with low SNR values. Though the algorithm is fair in giving results in optimal time further improvements can be made in the algorithm focusing on areas like the Gaussian filter, methods to calculate the gradients etc. Other areas include the noise suppression techniques in the image as the canny edge detection algorithm is highly affected by noise by making the image inappropriate for detection of the weak edges as they are also filtered while filtering the noise. Hence by making the advancements in the above given areas, this algorithm can be effectively used in areas like Computer Vision, Asphalt Concrete applications, Machine learning, etc.

## **6. References**

- Bhardwaj, S., & Mittal, A. (2012). A survey On various Edge Detector Techniques. *Procedia Technology, Volume 4*, 220-226.
- Chandra Sekhar, S., & Abin, J. (2013). Bilateral Edge Dectors. *ICASSP ,IEEE*, pp 1449-1453.
- J.F, C. (1986). A computational approach to edge detection. *IEEE Trans Pattern Anal Machine Intel, vol PAMI-8*, pp. 679-697.
- Joshi, S. R., & Koju, R. (2012). Study and Comparision of Edge Detection Algorithms. *IEEE*.
- Meghana , D., & G.K, A. (2012). Edge detection Technique ; a comparative approach. *World Journal Of science and technology*.