FPGA Implementation of Distributed Canny Edge Detection Algorithm

C.S.Manju
PG Student
Department of ECE
Government College of Technology, Coimbatore,
Tamil Nadu, India

C.Vasanthanayaki, PhD
Associate Professor
Department of ECE
Government College of Technology, Coimbatore,
Tamil Nadu, India

ABSTRACT
Edge detection is an important pre-processing step for any image processing application, object recognition and object detection. Among different edge detectors that are available, the Canny edge detector has better edge detection performance because it satisfies three main criteria which are low error rate, good localization and minimal response. In this paper, a mechanism to implement the Canny algorithm at block level with enhanced edge detection performance is proposed. By directly applying the original frame-level Canny algorithm at block level leads to more number of edges in smooth regions and to loss of important edges in highly-detailed regions since the original Canny algorithm computes the high and low thresholds based on the frame-level statistics. To solve this problem, a new method called Distributed Canny Edge Detection algorithm is proposed which adaptively calculates the high and low thresholds based on the block type and local distribution of the gradients in a block. In the proposed algorithm, instead of finding the direction of the gradient by calculating the arctangent vertical gradient to the horizontal gradient, the value and sign of the components of the gradient is analyzed to calculate the direction of the gradient. The proposed Distributed Canny edge detection algorithm is implemented in MATLAB. The resulting image shows that the proposed block-level algorithm detects more number of edges than the original frame-level Canny algorithm.

Keywords
Canny edge detector, Distributed Image Processing, MATLAB.

1. INTRODUCTION
Edge detection is an important pre-processing step for any image processing application, object recognition and object detection. Edge detection in noise free images is very simple. But it is difficult to detect edges in noisy images. Noisy images parameters are difficult to analyze and detect. Edge detection is a essential tool used in most image processing applications to obtain information from the frames as a precursor step to feature extraction and object segmentation. It detects object outlines and boundaries between objects and the background in the image. A filter can also be used to refine the appearance of blurred image.

Edge detectors are broadly classified into first order edge detector/gradient operator and second order edge detector/laplacian based operator. Classical edge detectors are first order detectors where the input image is convolved by an adapted mask to generate a gradient image in which edges are detected by thresholding. Robert Cross, Sobel and Prewitt operators are different types of classical operators. There is a set of work in [2], [3] that deal with implementation of classical edge detectors. Robert Cross operator uses kernels which are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. Although the size of the kernel is smaller as shown in [6] which leads to fast computation but this operator is very sensitive to noise. Both Prewitt and Sobel operator kernels respond maximally to edges running vertically and horizontally to the pixel grid, one kernel for each of the two perpendicular orientations. These operators are insensitive to noise but slow computation ability. The first derivative is positive at the points of transition into and out of the ramp; it is constant for points in the ramp; and is zero in areas of constant level.

Another classification of edge detectors is and second order edge detector/laplacian based operator. It has been shown in [13], second derivative is positive at the conversion associated with the dark side of the edge, negative at the conversion associated with the light side and zero along the ramp and in areas of constant gray level. It also points out two additional properties of second derivative: It produces two values for every edge in an image which is an undesirable property and an imaginary straight line joining the extreme positive and negative values of the second derivative would cross zero near the midpoint of an edge. This zero-crossing property is used for detecting edges in an image. Marr-Hildreth edge detector is second order edge detector which uses zero-crossing property to detect edges. But the limitations of Marr-Hildreth edge detector are detection of false edges and localization error is severe at curved edges.

It is concluded form the observations made in [6] and [13] that the magnitude of the first derivative can be used to find the presence of an edge at a point in an image. Likewise, the sign of the second derivative can be used to determine whether an edge pixel lies on the dark side or light side of an image. If an image is corrupted by noise, the usage of second derivative is not a wise choice because in the presence of noise the negative and positive components cannot be identified which is important in detecting the edges.

The rest of the paper is organized as follows. Section 2 gives a brief overview of existing Canny algorithm. Section 3 deals with the steps included in the proposed algorithm and the modifications done. The pseudo code for various steps is also presented in this section. Performance analysis of proposed Canny algorithm is presented in Section 4. Comparison of Distributed Canny with the existing Canny and various other edge detectors are also shown in this section. Finally, conclusion is presented in Section 5.

2. CANNY EDGE DETECTION ALGORITHM

Canny edge detector have advanced algorithm derived from the previous work of Marr and Hildreth. The Canny edge detection algorithm is known to many as the optimal edge detector. Canny proposed a list of criteria to improve existing methods of edge detection. Low error rate is the first criterion. It is important that true edges should not be missed and that there are no responses to non-edges. The second criterion is the localization of edge points. 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 third criterion was necessary because the first two criteria were not enough to completely eliminate the possibility of multiple responses to an edge.

The original Canny algorithm [8] works on frame-level statistics and consists of the following steps: 1) The first step is to filter out any noise in the original image before trying to locate and detect any edges. 2) After smoothing the image and removing the noise, the next step is to find the strength if the edge by taking the gradient of the image. This involves the calculation of the horizontal gradient G_x and G_y at each pixel location by convolving with gradient masks.3) Computation of gradient magnitude G at each pixel location which tells how quickly the image changes and computation of gradient direction θ_G at each pixel location which gives the direction in which the image is changing. 4) Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced. So if the pixels of a 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

It can be seen by looking at pixel "a", there are only four possible directions when describing the surrounding pixels- 0^o, 45^o, 90^o or 135^o. Therefore, any edge direction falling within the range (0^o to 22.5^o & 157.5^o to 180^o) is set to 0^o. Any edge direction falling within the range (22.5^o to 67.5^o) is set to 45^o. Any edge direction falling within the range (67.5^o to 112.5^o) is set to 90^o. Any edge direction falling within the range (112.5^o to 157.5^o) is set to 135^o. 5) Apply Non-Maximal Suppression (NMS) to thin edges. After converting the edge direction is one of 4 possible main directions (0^o, 45^o, 90^o or 135^o), the gradient magnitude of this pixel is compared with two of its immediate neighbours along the gradient direction and the gradient magnitude is set to zero if it does not correspond to a local maximum. 6) Calculation of high and low thresholds based on the gradient for the entire image. To select the threshold values some properties of gradient histogram are analyzed in [4]. The high threshold (thigh) is chosen from a flat part between the background peak and the edge peak in the histogram to reduce error detection.

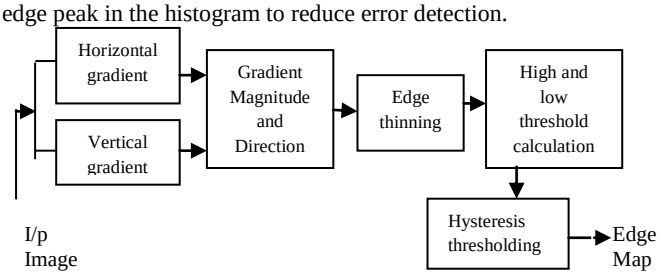


Figure 1. Block diagram of existing Canny algorithm

The low threshold (tlow) is set as 40% of high threshold. 7) Performing hysteresis thresholding to determine edge map. Pixels with a gradient magnitude greater than high threshold are kept as an edge. Pixels with a gradient magnitude lesser than low threshold are discarded immediately. If a pixel gradient magnitude lies between high and low thresholds, that pixel will be considered as an edge only if its neighbors in a 3 x 3 region around it have gradient magnitudes greater than high threshold. The block diagram of original Canny algorithm is shown in Figure 1.

3. PROPOSED DISTRIBUTED CANNY ALGORITHM

To enhance the edge detection performance of original frame-level Canny algorithm, a new algorithm called Distributed Canny algorithm is proposed in this paper. As discussed in Section 2, the classical Canny algorithm computes high and low thresholds based on the gradient magnitude histogram of an entire image. By applying same high and low threshold values for each block in an image will lead to more number of edges in smooth regions and loss of important edges in highly detailed regions as shown in [1]. In order to overcome this problem the high and low threshold values are chosen according to block-level statistics instead of frame-level statistics.

In the proposed method, first the input image is divided into m x m overlapping blocks. In order to do this, first the image should be divided into n x n non-overlapping blocks where n<m. For an L x L gradient mask, each block can be extended by (L+1)/2 along left, right, up and down. The non-overlapping blocks need to be extended in order to prevent edge artifacts and loss of edges at lock boundaries while computing the gradients and due to the fact that NMS operation at boundary requires gradient values of the neighboring pixels. Figure 3, shows an example of non-overlapping block and its extended overlapping block version in case when a 3x3 gradient mask. In order to perform NMS for the border pixel (i,j), the gradient information of the adjacent pixels (i-1, j-1), (i-1, j), (i-1, j+1) and (i+1, j-1) are taken as in [1]. In order to compute the gradient of the adjacent pixels (i-1, j-1), (i-1, j), (i-1, j+1) and (i+1, j-1) for the 3 x 3 gradient mask, the block has to be extended by 2 (where (L-1)/+1 = 2) pixels on all sides in order to generate a block of size (n+4) x (n+4). Thus m=n+4 for this example. Steps 1, 2 and 4 to 7 are same as in original Canny algorithm except those steps are applied for each blocks in an image.

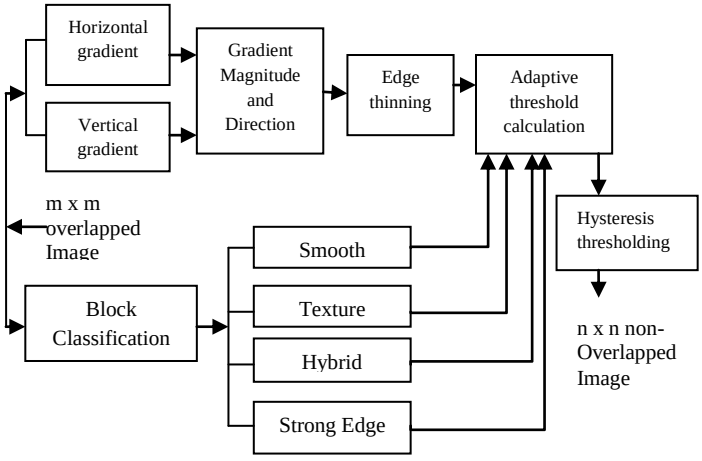


Figure 2. Block diagram of proposed Canny algorithm

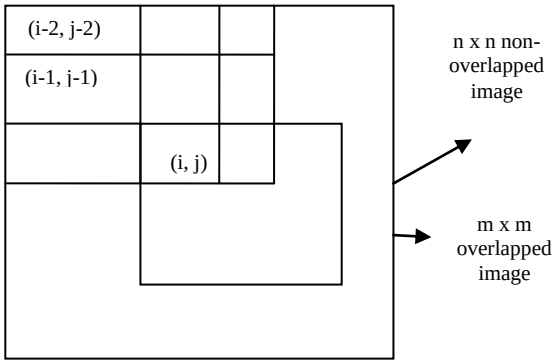


Figure 3. An example for m x m overlapping block

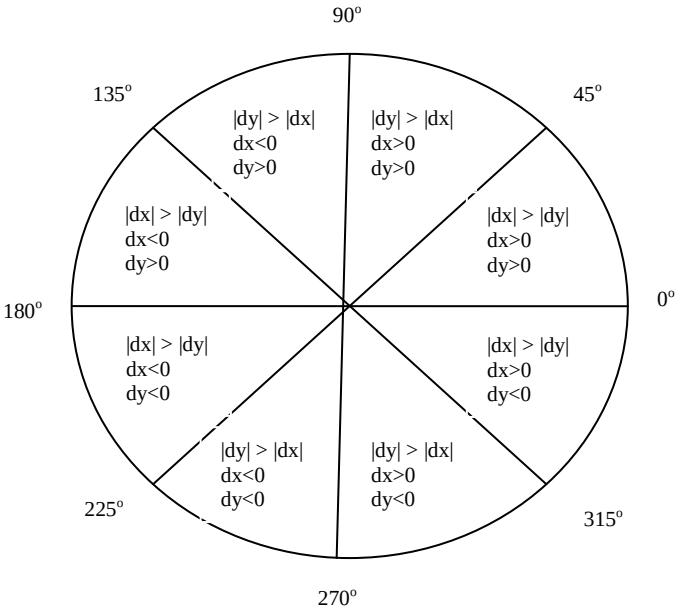


Figure 4. Gradient orientation

In this paper, another modification is carried out in computation of gradient direction step. Instead of finding the direction of the gradient by calculating the arctangent vertical gradient to the horizontal gradient θ = arctan(dy/dx), the value and sign of the components of the gradient is analyzed to calculate the direction of the gradient. If the current pixel is Px,y and the derivative values at that pixel are dx and dy, the gradient direction at P can be approximated to one of the sectors shown in Figure 4. Once the gradient direction is known, the values of the pixels found in the neighbourhood of the pixel under consideration are interpolated. The pixel which does not have local maximum gradient magnitude is removed. The comparison is made between the actual pixel and its neighbors, along the gradient direction. For example, if the approximate direction of the gradient is between 0^o and 45^o, the magnitude of the gradient at $P_{x,y}$. is compared with the magnitude of the gradient at adjacent points as shown in Figure 5, where $P_{x,y} = | dx_{x,y} | + | dy_{x,y} |$. The values of the gradient at the point P_a and P_b are defined as follows:

$$P_a = \frac{P_{x+1,y-1} + P_{x+1,y}}{2}, \quad (1)$$

Where $P_{x+1,y-1} = | dx_{x+1,y-1} | + | dy_{x+1,y-1} |$ *and*
$P_{x+1,y-1} = | dx_{x+1,y-1} | + | dy_{x+1,y-1} |$

$$P_b = \frac{P_{x-1,y+1} + P_{x+1,y-1}}{2}, \quad (2)$$

Where $P_{x-1,y+1} = | dx_{x-1,y+1} | + | dy_{x-1,y+1} |$ *and*
$P_{x+1,y-1} = | dx_{x,y+1} | + | dy_{x,y+1} |$

The centre pixel $P_{x,y}$ is considered as an edge, if $P_{x,y} > P_a$ and $P_{x,y} > P_b$. If both conditions are not satisfied then the centre pixel is removed. Block diagram of proposed algorithm is shown in Figure 2.

After dividing the input image into n x n overlapping blocks, we classify each block into six types as, uniform, uniform/texture, texture, edge/texture, medium edge and strong edge block, by adopting the block classification method of [7]. In order to classify the blocks, first we have to identify each pixel in a block as uniform, texture and edge pixels. This pixel classification method [7] utilized the local variance of each pixel using a 3 x 3 window that is centered around the considered pixel in order to label them. Then blocks are classified according to the table followed in [7].

A practice database of 200 images is formed from the Berkeley Segmentation Image Database [10]. From the practice set images, for each block type, the appropriate percentage value P1 that would result in high and low threshold values similar to the ones that are obtained for entire image is determined. By following the adaptive selection scheme described in [1], appropriate P1 values are selected.

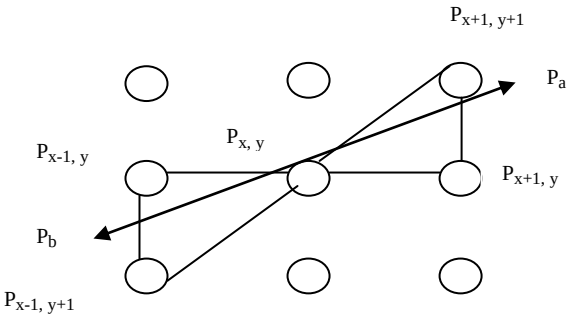


Figure 5. Pixel interpolation

Table 1. P1 values for each block type with different block sizes

Block Size	Block Type					
	Uniform	Uniform/ Texture	Texture	Edge/ Texture	Medium Edge	Strong Edge
8x8	0	0	0.0312	0.1022	0.2183	0.4820
16x16	0	0	0.0307	0.1016	0.2616	0.4830
32x32	0	0	0.0305	0.1117	0.2079	0.4852
64x64	0	0	0.0318	0.1060	0.2218	0.4670
128x128	0	0	0.0302	0.0933	0.2375	0.4842
256x256	0	0	0.0299	0.0911	0.2304	0.4893

Step 1: Pixel Classification

$$\text{Pixel type} = \begin{cases} uniform, & \text{var}(x,y) \le T_u \\ texture, & T_u < \text{var}(x,y) \le T_e \\ edge, & T_e < \text{var}(x,y) \end{cases}$$

Step 2: Block Classification
var(x,y): the local (3x3) variance at pixel (x,y).
T_u and T_e : two thresholds as in [7].
Total_Pixel: the total number of pixels in the block.
$N_{uniform}$: the total number of uniform pixels in the block.
N_{edge}: the total number of edge pixels in the block.

(a)

Block type	No. of pixels of pixel type	
	$N_{uniform}$	N_{edge}
Smooth	$\geq 0.3*Total\_Pixel$	0
Texture	$< 0.3*Total\_Pixel$	0
Edge/texture	$< 0.65(Total\_Pixel - N_{edge})$	$(>0)\&(< 0.3*Total\_Pixel)$
Medium edge	$\geq 0.65(Total\_Pixel - N_{edge})$	$(>0)\&(< 0.3*Total\_Pixel)$
Strong edge	$\leq 0.7*Total\_Pixel$	$\geq 0.3*Total\_Pixel$

```
Let P1 be the percentage of pixels, in a block, that would be classified as
strong edges [1].
Step 1: If smooth block type
                P1 = 0;                         /*No edges*/
        else if texture block
                P1 = 0.03;                      /*Few edges*/
        else if texture/edge block type
                P1 = 0.1;                       /*Some edges*/
        else if medium edge block type
                P1 = 0.2;                       /*Medium
edges*/
        else
                P1 = 0.4;                       /*Many edges*/
Step 2: Compute High_threshold = 1-P1
Step 3: Compute Low_threshold = 0.4*High_threshold
```

(b)

Figure 6. Pseudo-code for the proposed (a) block classification and (b) adaptive threshold selection scheme.

First, the high threshold of the entire image is calculated, for each image in the practice database. Then, the image is divided into blocks and the blocks are classified into six block types as discussed previously. Then, for each block type, the gradient magnitude CDF is measured and the corresponding CDF used to measure the *P1* value such that the local high threshold of the blocks in this class is the same as the one for the whole image. The resulting *P1* value for a considered block type is measured as the average value of its corresponding set over all images and over all block sizes. The 512×512 images are divided into fixed-size blocks, with the block size varying from 8 × 8 to 256×256, to evaluate the robustness of the obtained *P1* values with respect to the block size. Table I shows the *P1* values that are obtained for each block type and for each block size. It should be noted that the *P1* values for uniform and uniform/texture blocks are equal to 0 for all block sizes. This illustrates that the uniform and uniform/texture blocks can be united into one block type, which we refer to as smooth block type. Moreover, this signifies that there are no pixels that should be classified as edges in a smooth block.

High thresholds and low thresholds are required for thresholding with hysteresis. The high threshold corresponds to the point at which the value of the gradient magnitude cumulative distribution function (CDF) equals to 1-P1. The low threshold is computed as a percentage P2 of the high threshold [1].Then hysteresis thresholding is applied to each block in an image to determine the edge image, instead of applying to the whole image as in original Canny algorithm.

4. MATLAB RESULTS

4.1 Performance Analysis

The proposed Distributed Canny edge detection algorithm is implemented in MATLAB. All images taken are gray scale images. Gaussian filter is used to filter out any noises in the image and the size of the mask is chosen as 5 x 5.Sobel operators is used to find horizontal and vertical gradient images. Figure 7 shows a comparison made between proposed Canny with overlapping and non-overlapping blocks. Block size of 64 x 64 is taken for non-overlapping and since Sobel operator is used the overlapping block size will become 68 x 68(since size of the gradient mask is 3 x3) Both the proposed methods have same edge detection performance. In Figure 8, Distributed Canny edge detection is implemented with the conventional gradient and proposed gradient direction step. The proposed step has better edge detection performance than the conventional one. Figure 9, 10 and 11 depict that the proposed Canny has better edge detection than the existing Canny and all other edge detectors.

4.2 Computation Time Analysis

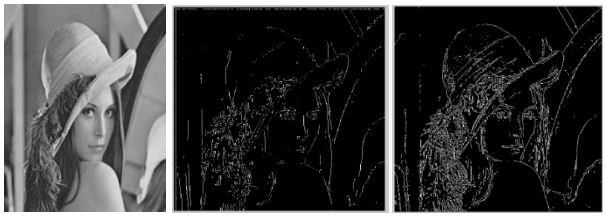
The Table 2 shows that the Distributed Canny edge detection algorithm takes more time to compute than original existing Canny algorithm. It also shows that there is not much difference in computation time between overlapped and non-overlapped Distributed Canny edge detectors.

Table 2. Comparison of computation times

Input Image Size	Existing Canny	Proposed Canny Non-overlapping	Proposed Canny overlapping
512 x 512 (lena.jpg)	1.230s	25.503s	26.772s
256 x 256 (rice.png)	0.660s	6.6982s	7.091s

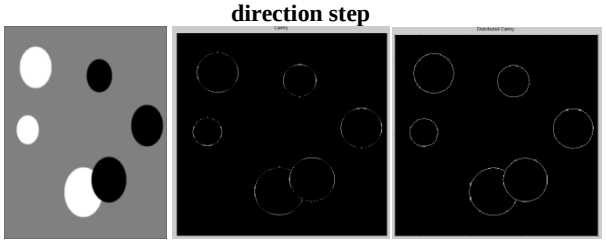
(a) (b) (c)

Figure 7. (a) 512 x 512 "Charlie Chaplin" image; final edge map of proposed distributed Canny with (b) non-overlapping blocks (c) overlapping blocks



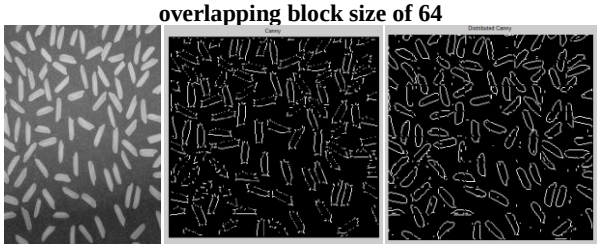
(a) (b) (c)

Figure 8. (a) 512 x 512 "Lena" image; final edge map of proposed non-overlapping distributed Canny with (b) conventional gradient direction step (c) proposed gradient direction step



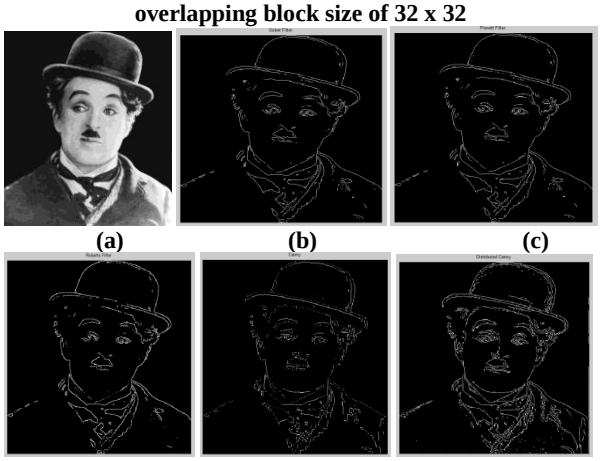
(a) (b) (c)

Figure 9. (a) 512 x 512 "Circles" image; final edge map of (b) original Canny (c) proposed Canny with non-overlapping block size of 64



(a) (b) (c)

Figure 10. (a) 256 x 256 "Rice" image; final edge map of (b) original Canny (c) proposed Canny with non-overlapping block size of 32 x 32



(a) (b) (c)

(d) (e) (f)

Figure 11. (a) 512 x 512 "Charlie Chaplin" image; final edge map of (b) Sobel (c) Prewitt (d) Roberts (e) original Canny (f) proposed distributed non- overlapping Canny

5. CONCLUSION

In this paper, Canny edge detection algorithm is implemented at block level in MATLAB. In the proposed Distributed Canny edge detection algorithm, the high and low threshold values are calculated for each block in an image. To support this adaptive threshold selection scheme, Block Classification Scheme is proposed in order to classify type of each block in an input image as Smooth, Texture, Edge/texture, Medium edge and Strong edge blocks. For classifying the blocks, Pixel Classification Method is proposed. With these steps for selecting the high and low thresholds, the proposed method shows enhanced edge detection performance than original Canny algorithm. It also has been shown in this project that the proposed method has better edge detection performance than many existing algorithms. Also, by the replacement of conventional gradient direction computation step with the proposed step results in better edge detection. But the limitation with the proposed method is increased computation time. The proposed algorithm can be implemented in FPGA.

6. REFERENCES

[1] Qian Xu, Srenivas Varadarajan, Chaitali Ckakrabarti, and Lina J. Karan, "A Distributed Canny Edge Detector: Algorithm and FPGA Implementation", IEEE Trans Image Processing, Vol.23, No.7, July 2014, pp. 2944-2960.Ding, W. and Marchionini, G. 1997 A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[2] Indrajeet Kumar, Jyoti Rawat, Dr. H.S. Bhadauria, "A conventional study of edge Detection technique in Digital image processing", International Journal of Computer Science and Mobile Computing, Vol.3 Issue.4, April 2014, pp. 328-334.Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[3] Chinu and Amit Chhabra, "Overview and Comparative Analysis of Edge Detection Techniques in Digital Image Processing", International Journal of Information & Computation Technology. ISSN 0974-2239 Volume 4, Number 10, 2014, pp. 973-980Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3 (Mar. 2003), 1289-1305.

[4] Q. Xu, C. Chakrabarti, and L. J. Karam, "A distributed Canny edge detector and its implementation on FPGA," in *Proc. DSP/SPE)*, Jan. 2011, pp. 500–505.

[5] Daggu Venkateshwar Rao*, Shruti Patil, Naveen Anne Babu and V Muthukumar, "Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture using C-based Hardware Descriptive Languages", International Journal of Theoretical and Applied Computer Sciences, Volume 1 Number 1,2006, pp. 9–34.

[6] Mitra Basu, "Gaussian Based Edge-Detection Methods A Survey", IEEE Transactions on System, man, and cybernetics part c: Application and Reviews, Vol. 32, No. 3, August 2002, pp. 252-260.

[7] J. K. Su and R. M. Mersereau, "Post-processing for artifact reduction in JPEG-compressed images," in *Proc. IEEE ICASSP*, vol. 3, May 1995, pp. 2363–2366.

[8] J. F. Canny , "A computational approach to edge detection", IEEE Trans. Pattern Anal. Machine Intell. vol. PAMI-8, no. 6, 1986, pp. 679-697.

[9] W. E. Grimson and E. C. Hildreth , "Comments on Digital step edges from zero crossings of second Directional derivatives'', IEEE Trans. Pattern Anal. Machine Intell., vol. PAMI-7, no. 1, 1985, pp. 121-129.

[10] P. Arbelaez, C. Fowlkes, and D. Martin. 2013, The Berkeley Segmentation Dataset and Benchmark [Online]. Available: http://www.eecs.berkeley.edu/Research/ Projects/CS/vision/bsds/.