International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Split Canny Edge Detection: Algorithm and its FPGA Implementation

Shamlee .V[1], Jeyamani[2]

[1, 2]Department of Electronics and Communication, United Institute of Technology, Coimbatore, India

Abstract: ***We present a Split Canny edge detection algorithm that results in significantly reduced memory requirements, decreased latency and increased throughput with no loss in edge detection performance as compared to the original Canny algorithm. The Canny edge detector is one of the most widely used edge detection algorithms due to its superior performance. Unfortunately, not only is it computationally more intensive as compared with other edge detection algorithms, but it also has a higher latency because it is based on frame-level statistics. In this paper, we propose a mechanism to implement the Canny algorithm at the block level without any loss in edge detection performance compared with the original frame-level Canny algorithm. Directly applying the original Canny algorithm at the block-level leads to excessive edges in smooth regions and to loss of significant edges in high-detailed regions since the original Canny computes the high and low thresholds based on the frame-level statistics. To solve this problem, we present a Split Canny edge detection algorithm that adaptively computes the edge detection thresholds based on the block type and the local distribution of the gradients in the image block. In addition, the new algorithm uses a non uniform gradient magnitude histogram to compute block-based hysteresis thresholds. The resulting block-based algorithm has a significantly reduced latency and can be easily integrated with other block-based image codecs. It is capable of supporting fast edge detection of images and videos with high resolutions, including full-HD since the latency is now a function of the block size instead of the frame size. In addition, quantitative conformance evaluations and subjective tests show that the edge detection performance of the proposed algorithm is better than the original frame-based algorithm, especially when noise is present in the images. Finally, this algorithm is implemented using a 32 computing engine architecture and is synthesized on the Xilinx Virtex-5 FPGA***

Keywords: Distributed image processing, Canny edge detector, high throughput, parallel processing, FPGA.

1. Introduction

Edge detection is a very important first step in many algorithms used for segmentation, tracking and image/video coding. The Canny edge detector is predominantly used due to its ability to extract significant edges. A lot of edge detection algorithms, such as Robert detector, Prewitt detector, Kirsch detector, Gauss -Laplace detector and Canny detector have been proposed. Among these algorithms, Canny algorithm has been used widely in the field of image processing because of its good performance].The Canny edge detector is predominantly used in many real -world applications due to its ability to extract significant edges with good detection and good localization performance. Unfortunately, the Canny edge detection algorithm contains extensive pre-processing and post-processing steps and is more computationally complex than other edge detection algorithms. Furthermore, it performs hysteresis thresholding which requires computing high and low thresholds based on the entire image statistics. This places heavy requirements on memory and results in large latency, hindering real-time implementation of the Canny edge detection algorithm [3]. Many implementations of the Canny algorithm have been proposed on a wide list of hardware platforms. The Canny-Deriche filter [1] is a network with four transputers that detect edges in a 256 × 256 image in 6s, far from the requirement for real- time applications. Although the design in [2] improved the Canny-Deriche filter implementation of [1] and was able to process 25 frames/s at 33 MHz, the used off-chip SRAM memories consist of Last-In First-Out (LIFO) stacks, which increased the area overhead compared to [1]. Demigny pro- posed a new organization of the Canny-Deriche filter in [3], which reduces the memory size and the computation cost by a factor of two. However, the number of clock cycles per pixel of the implementation [3] varies with the size of the processed image, resulting in variable clock-cycles/pixel from one image size to another with increasing processing time as the image size increases.

Our focus is on reducing the latency and increasing the throughput of the Canny edge detection algorithm so that it can be used in real-time processing applications. As a first step, the image can be partitioned into blocks and the Canny algorithm can be applied to each of the blocks in parallel. Unfortunately, directly applying the original Canny at a block-level would fail since it leads to excessive edges in smooth regions and loss of significant edges in high-detailed regions. In this paper, we propose an adaptive threshold selection algorithm which computes the high and low threshold for each block based on the type of block and the local distribution of pixel gradients in the block. Each block can be processed simultaneously, thus reducing the latency significantly. Furthermore, this allows the block-based Canny edge detector to be pipelined very easily with existing block-based codecs, thereby improving the timing performance of image/video processing systems. Most importantly, conducted conformance evaluations and subjective tests show that, compared with the frame-based Canny edge detector, the proposed algorithm yields better edge detection results for both clean and noisy images.

The block-based Canny edge detection algorithm is mapped onto an FPGA-based hardware architecture. The architecture is flexible enough to handle different image sizes, block sizes and gradient mask sizes. It consists of 32 computing engines configured into 8 groups with 4 engines per group. All 32 computing engines work in parallel lending to a 32-fold decrease in running time without any change in performance when compared with the frame-based algorithm. The architecture has been synthesized on the Xilinx Virtex-5

FPGA. It occupies 64% of the total number of slices and 87% of the local memory, and takes 0.721ms (including the SRAM read/write.

In this paper, FPGA synthesis results, including the resource utilization, execution time, and comparison with existing FPGA implementations are presented.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the original Canny algorithm. Section 3 presents the proposed Split Canny edge detection algorithm which includes the adaptive threshold selection algorithm and a non-uniform quantization method to compute the gradient magnitude histogram. Quantitative conformances as well as subjective testing results are presented in Section 4 in order to illustrate the edge detection performance of the proposed Split Canny algorithm as compared to the original Canny algorithm for clean as well as noisy images. In addition, the effects of the gradient mask size and the block size on the performance of the proposed Split Canny edge detection scheme are discussed and illustrated in Section 4. The proposed hardware architecture and the FPGA implementation of the proposed algorithm are described in Section 5. The FPGA synthesis results and comparisons with other implementations are presented in Section 6. Finally, conclusions are presented in Section 7.

2. Canny Edge Detection

Canny edge detector which is also well known as an optimal edge detector is one of the most efficient and successful edge detection methods. It operates on the gray-scale version of the image under consideration. He considered three criteria desired for any edge detector:

a) <u>*Good detection*</u>: The algorithm should mark as many real edges as possible in the image.
b) <u>*Good localization*</u>: Edges marked should be as close as possible to the edges in the real image.
c) <u>*Minimal response*</u>: *A given edge in the image should only be marked once and noise should not create false edges.*

The effect of the Canny operator is determined by three parameters:

- Width of the Gaussian kernel used in the smoothing stage, and the upper and lower thresholds used by the edge tracker.
- By increasing the width of the Gaussian kernel, reduces the edge detector's sensitivity to noise
- The localization error in the detected edges increases as the Gaussian width is increased. The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images, and it is regarded as a near optimal edge detection technique because it produces reliable, thin edges even in the presence of noise in the image. It is developed by John Canny in 1986.

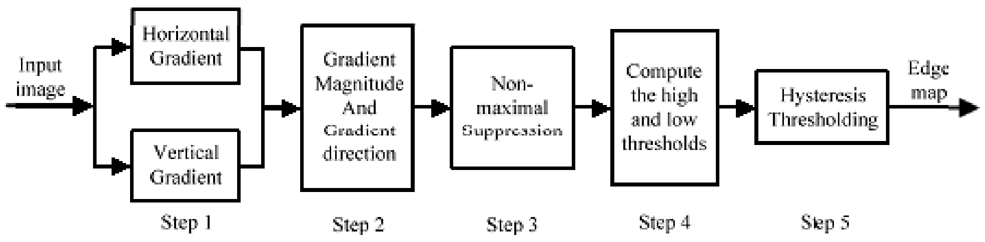


Figure 1: Block Diagram of Canny Edge Detection Algorithm

The Canny algorithm has 5 separate steps:

- *Smoothing*: removal of noise from image
- *Gradient magnitude and direction:* the image having larger magnitudes should be marked as an edge.

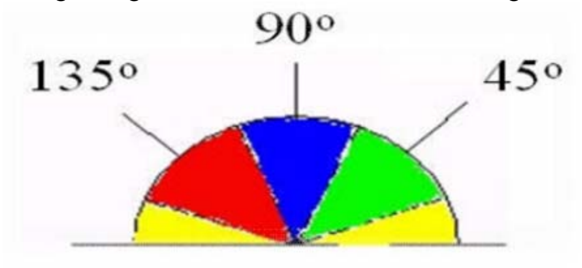


Figure 2: Gradient direction

- *Non-maximum suppression:* Only local maxima should be marked as edges.
- *Double thresholding*: strong edges are determined by thresholding.
- *Edge tracking by hysteresis*: Final edges are determined by suppressing all the edges that are not connected to strong edges.

As a result we miss some edges or detect some spurious edges when the threshold is not set to a proper value. the latency is increased, throughput is decreased and cost is increased. To overcome these limitations of a canny edge detector, a Split canny edge detector is proposed.

3. Proposed Split Canny Edge Detection Algorithm

The superior performance of the frame-based Canny algorithm is due to the fact that it computes the gradient thresholds by analyzing the histogram of the gradients at all the pixel locations of an image. Though it is purely based on the statistical distribution of the gradient values, it works well on natural images which consist of a mix of smooth regions, texture regions and high-detailed regions [1]. Direct ly applying the frame-based Canny at a block-level would fail because such a mix of regions may not be available locally in every block of the frame. This would lead to excessive edges in texture regions and loss of significant edges in high detailed regions. The Canny edge detection algorithm operates on the whole image and has a latency that is proportional to the size of the image. While performing the original canny algorithm at the block-level would speed up the operations, it would result in loss of significant edges in high -detailed regions and excessive edges in texture regions. Natural images consist of a mix of smooth regions, texture regions and high -detailed regions and such a mix of regions may not be available locally in every block of the entire image. In [1], it is proposed a Split Canny edge detection algorithm, which removes the inherent dependency between the various blocks so that the image can be divided into blocks and each block can be processed in parallel. In order to improve the performance of the edge detection at the block level and achieve the same performance as the original frame-based Canny edge detector when this latter one is applied to the entire image, a Split Canny edge detection algorithm is proposed.

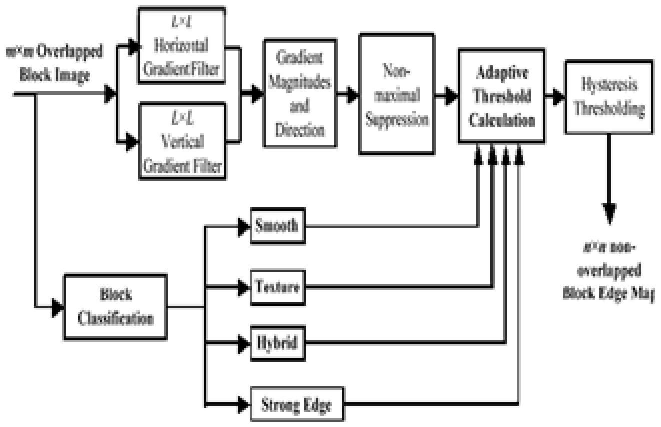

m×m Overlapped Block Image
L×L Horizontal Gradient Filter
L×L Vertical Gradient Filter
Gradient Magnitudes and Direction
Non-maximal Suppression
Adaptive Threshold Calculation
Hysteresis Thresholding
n×n non-overlapped Block Edge Map
Smooth
Texture
Hybrid
Strong Edge
Block Classification

Figure 3: Block Diagram of Proposed Split Canny Edge Detection algorithm

A diagram of the proposed algorithm is shown in Fig. 3. In the proposed Split version of the Canny algorithm, the input image is divided into m × m overlapping blocks and the blocks are processed independent of each other. For an L × L gradient mask, the m × m overlapping blocks are obtained by first dividing the input image into n × n non-overlapping blocks and then extending each block by (L + 1)/2 pixels along the left, right, top, and bottom boundaries, respectively. This results in m × m overlapping blocks, with m = n + L + 1. The non-overlapping n ×n blocks need to be extended in order to prevent edge artifacts and loss of edges at block boundaries while computing the gradients and due to the fact that the NMS operation at boundary pixels requires the gradient values of the neighboring pixels of the considered boundary pixels in a block.

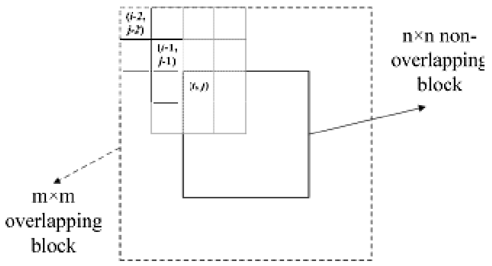


Figure 4: An example of the structure of an m × m overlapping block, where m = n + L + 1foran L × L(L = 3) gradient mask and when the image is initially divided into n × n non-overlapping blocks.

Fig. 4 shows an example of non-overlapping block and its extended overlapping block version in the case when a 3×3 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), (i, j−1), (i+1, j−1) are needed. In order to compute the gradient of the adjacent pixels (i −1, j −1), (i −1, j), (i −1, j +1), (i, j −1), (i +1, j −1)for the 3 × 3 gradient mask, the block has to be extended by2(where(L − 1)/2 + 1 = 2) pixels on all sides in order to generate a block of size (n + 4) × (n+4). Thus, m equals ton+4 for this example. Note that, for each block, only edges in the central n × n non-overlapping region are included in the final edge map, where n = m − L − 1. Steps 1 to 3and Step 5 of the Split Canny algorithm are the same as in the original Canny algorithm except that these are now applied at the block level. Step 4, which is the hysteresis high and low thresholds calculation, is modified to enable parallel block-level processing without degrading the edge detection performance.

Exploiting the local block statistics can significantly improve the edge detection performance for the distributed Canny edge detector. For this purpose, in order to learn the statistics of local blocks, we use a training database consisting of natural images. For each image in the database, we first divide the image into n×n non-overlapping blocks and classify the blocks into six types, uniform, uniform/texture, texture, edge/texture, medium edge, and strong edge block, by adopting the block classification method of [17]. This classification method utilized the local variance of each pixel using a 3 × 3 window that is centered around the considered pixel in order to label it as of type edge, texture, or uniform pixel. Then, each block is classified based on the total number of edge, texture, and uniform pixels in the considered block.

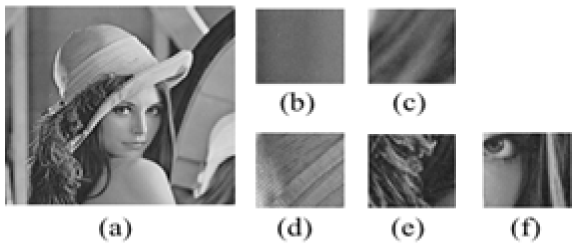


Figure 5: (a) Original 512 × 512 Lena image; b) uniform block; (c) uniform/texture block; (d) texture block; (e) edge/texture block; (f) medium edge block of the Lena image. Shown blocks are of size 64 × 64.

Fig. 5 shows an example where the 512 × 512 Lena image is divided into blocks, and each block is classified according to this classification technique. Fig. 5(b)–(f) provide, respectively, examples of uniform, uniform/texture, texture

In order not to degrade the performance of the original frame-based Canny algorithm when it is applied to a block, the high and low thresholds when computed at the block level should match thresholds that would have been obtained for the entire Image.

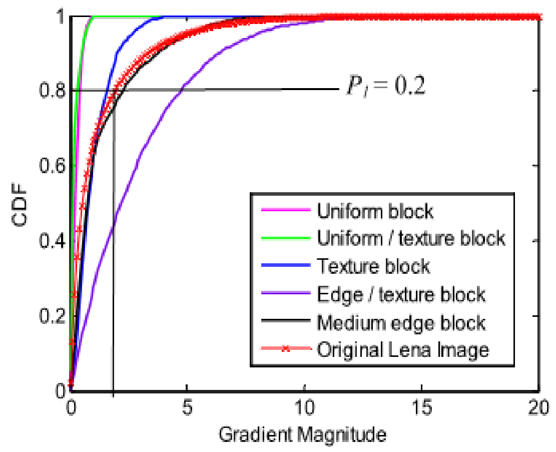


Figure 6: Normalized gradient magnitude CDFs for the 512 × 512 Lena image and CDFs for the 64 × 64 uniform block, uniform/texture block, texture block, edge/texture block, medium/edge block shown in Fig. 5.

The gradient magnitude CDFs of each block in Fig. 5 are shown in Fig. 6,1 along with the gradient- magnitude CDF of the entire Lena image. According to the CDF of the entire Lena image, the high threshold should be selected as 1.8 corresponding to a P1 value (percentage of pixels that should be classified as strong edges) of 0.2. However, if P1 is still set to 0.2 for the medium edge block, the local high threshold for that block would be about 5.2, which is significantly different from the high threshold, 1.8, that is obtained for the entire image. Such a setting will result in a loss of significant edges in the considered strong edge block. On the other hand, the local high thresholds for the uniform and uniform/texture block would be about 0.7, if P1 is set to 0.2. Such a setting will lead to excessive edges in the uniform and uniform/texture blocks. From this analysis, it can be seen that, in order to keep a similar threshold for all block types, P1 should be selected differently for different types of blocks by adapting its value to the local block content. In order to determine, 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 the entire image, training set 2 of 200 images is formed from the Berkeley Segmentation Image Database [18]. For each image in the training database, the high threshold of the entire image is first calculated. 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 computed and the corresponding CDF used to compute the P1 value such that the local high threshold of the blocks entire image, a training set 2 of 200 images is formed from the Berkeley Segmentation Image Database [18]. For each image in the training database, the high threshold of the entire image is first calculated. 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 computed and the corresponding CDF used to compute the P1 value such that the local high threshold of the blocks in this class is the same as the one for the entire image.

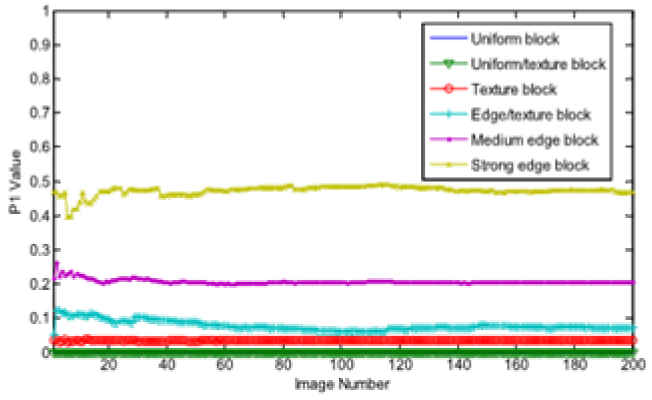


Figure 7: P1 values for each block type.

Fig. 7 shows the P1 values that are obtained for different block types, each of size 64 × 64, for 200 512 × 512 images with varied content. It illustrates that the P1 values of each block type are highly different from each other, except for the uniform block and uniform/texture block types. Also, for a given block type, the P1 values across all 200 images are quite similar for different block sizes. The final P1 value for a considered block type is computed as the average value of its corresponding set over all images and over all block sizes.

Table 1

Block type	Uniform	Uniform/ Texture	Texture	Edge/ Texture	Medium edge	Strong edge
Var. of P_l	0	0	9.37 E-4	0.0138	0.0071	0.013

Standard Deviations of P1 Values for Each Block type for 64 × 64 Block

For illustration, the standard deviations of the obtained P1 values for each block type are shown in Table I for 64 × 64 blocks. To evaluate the robustness of the obtained P1 values with respect to the block size, the 512×512 images are divided into fixed-size blocks, with the block size varying from 8 × 8 to 256×256. Table II 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, which indicates that the uniform and uniform/texture blocks can be combined into one block type, which we refer to as smooth block type. Also, this implies that there are no pixels that should be classified as edges in a smooth block. Therefore, there is no need to perform edge detection on smooth blocks, and this result in reduced computations, a feature that is exploited in the FPGA implementation.

In order to compute the high and low hysteresis thresholds, a finely quantized gradient magnitude histogram is usually needed. Here, we employ the non-uniform quantizer, which has been proposed by us in [13], to obtain the gradient magnitude histogram for each block such that the high threshold can be precisely calculated. Examples of gradient magnitude histograms for an edge/texture, medium edge and strong edge block are shown in Fig. 8.

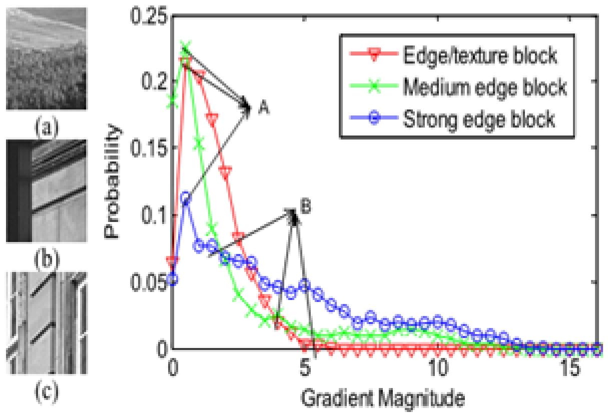


Figure 8: (a)-(c) Different types of 64 × 64 blocks ((a) Edge/texture,(b) medium edge, (c) strong edge) and (d)corresponding gradient magnitude histograms

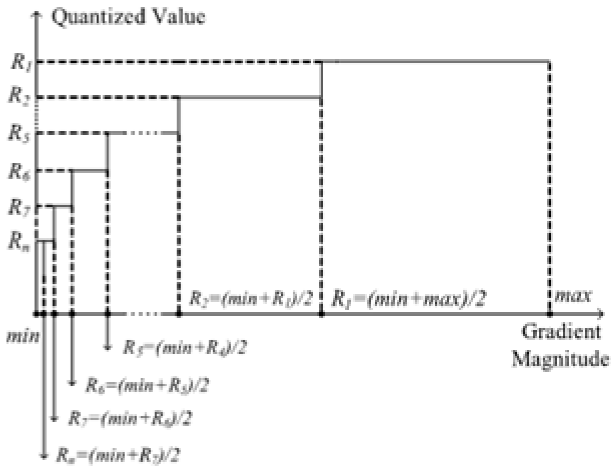
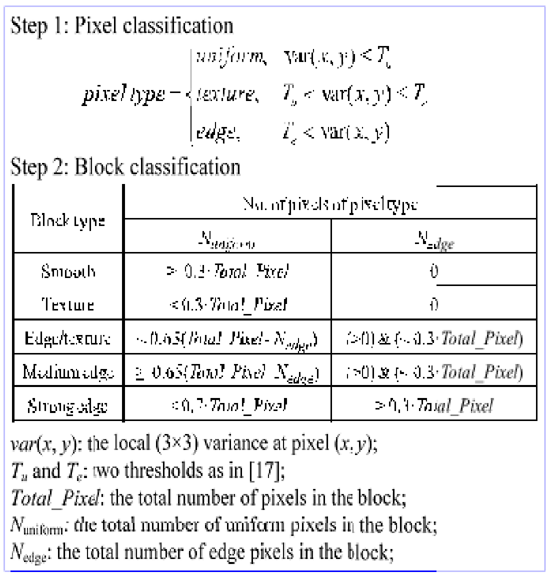


Figure 9: Reconstruction values and quantization levels

Fig. 9 shows the schematic diagram of the proposed non-uniform quantizer. The first reconstruction level (R1) is computed as the average of the maximum value and minimum value of the gradient magnitude in the considered block, and the second reconstruction level (R2) is the average of the minimum value of the gradient magnitude and the first reconstruction level. Accordingly, n reconstruction levels can be computed as shown in [13], R1 = (min+max)/2 and Ri+1 = (min+Ri)/2(i = 2, 3, ... n),where min and max represent the minimum and maximum values of the gradient magnitude, respectively, and Ri is the reconstruction level.


Step 1: Pixel classification

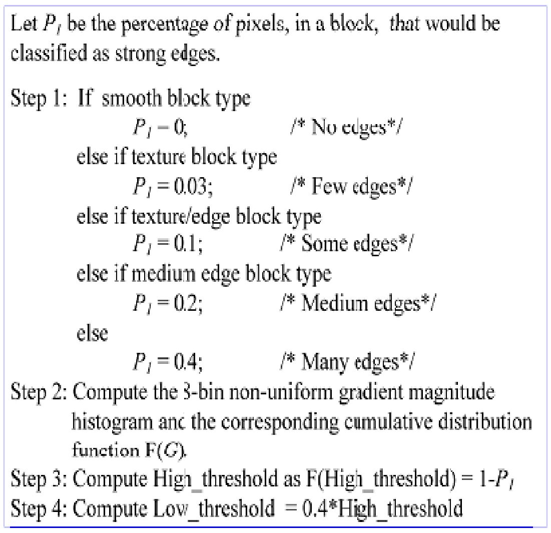
$$pixel\ type = \begin{cases} uniform, & \mathrm{var}(x,y) < T_u \\ texture, & T_u < \mathrm{var}(x,y) < T_e \\ edge, & T_e < \mathrm{var}(x,y) \end{cases}$$

Step 2: Block classification

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

$var(x, y)$: the local (3×3) variance at pixel (x, y);
T_u and T_e: two thresholds as in [17];
$Total\_Pixel$: 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)


Let P_l be the percentage of pixels, in a block, that would be classified as strong edges.

```
Step 1: If smooth block type
            $P_l = 0$;          /* No edges*/
        else if texture block type
            $P_l = 0.03$;       /* Few edges*/
        else if texture/edge block type
            $P_l = 0.1$;        /* Some edges*/
        else if medium edge block type
            $P_l = 0.2$;        /* Medium edges*/
        else
            $P_l = 0.4$;        /* Many edges*/
Step 2: Compute the 8-bin non-uniform gradient magnitude
        histogram and the corresponding cumulative distribution
        function F(G).
Step 3: Compute High_threshold as F(High_threshold) = 1-$P_l$
Step 4: Compute Low_threshold = 0.4*High_threshold
```

(b)

Figure 10: Pseudo-codes for the proposed (a) block classification and (b) adaptive threshold selection scheme

The pseudo-code of the block classification technique [17] and the proposed adaptive threshold selection algorithm is shown in Fig. 10(a) and (b), respectively.

4. MATLAB Experimental Results

A.Parametrical Analysis

The performance of the proposed algorithm is affected by two parameters, the mask size and the block size. the size of the gradient mask is a function of the standard

1) The Effect of Mask Size: deviation σ of the Gaussian filter, and the best choice of σ is based on the image characteristics. Canny has shown in [14] that the optimal operator for detecting step edges in the presence of noise is the first derivative of the Gaussian operator. As stated in Section 2, for the original Canny algorithm as well as the proposed algorithm, this standard deviation is a parameter that is typically set by the user based on the knowledge of sensor noise characteristics. It can also be set by a separate application that estimates the noise and/orblur in the image. A large value of σ results in smoothing and improves the edge detector's resilience to noise, but it undermines the detector's ability to detect the location of true edges. In contrast, a smaller mask size (corresponding to a lower σ) is better for detecting detailed textures and fine edges but it decreases the edge detector's resilience to noise.An L-point even-symmetric FIR Gaussian pulse-shaping filter design can be obtained by truncating a sampled version of the continuous-domain Gaussian filter of standard deviation σ.The size L of the FIR Gaussian filter depends on the standarddeviation σ and can be determined as follows:

$$L = 2 \cdot L_{side} + 1$$

$$L_{side} = \left\lfloor 2\sigma \sqrt{(-1)\log(C_T)} \right\rfloor$$

where CT represents the cut-off value in the spatial domain of the continuous-domain Gaussian function and determines the cut-off error.

2) Block Size: To find out the smallest block size for which the proposed Canny algorithm can detect all the psycho-visually important edges, the perceptual visual quality of the obtained edge maps was assessed using visual quality metrics.

B. Edge Detection Performance Analysis

The edge detection performance of the proposed split approach is analyzed by comparing the perceptual significance of its resulting edge map with the one produced by the original.

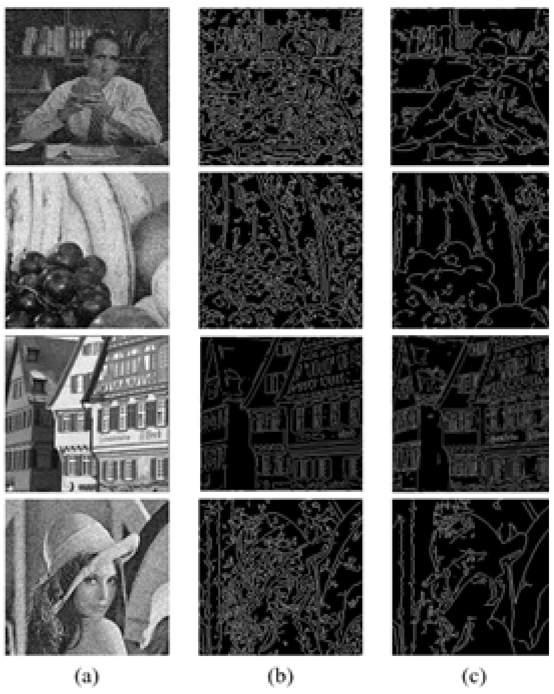


Figure 11: Comparison of the edge maps of noisy images by using the originalCanny edge detector and the proposed method: (a) images with Gaussian white noise (σn = 0.01); edge-maps of (b) the original Canny edge detector,and (c) the proposed algorithm with a non-overlapping block size of 64 × 64,using a 9 × 9 gradient mask to noise than the original frame-based Canny.

To further assess the performance of the proposed Split Canny algorithm, quantitative conformance evaluations and subjective tests are performed. The conformance evaluations aim to evaluate the similarity between edges detected by the original frame-based Canny algorithm and the proposed distributed Canny edge detection algorithm, while the subjective tests aim to validate whether the edge detection performance of the proposed distributed Canny is better, worse, or similar to the original frame-based Canny as perceived by subjects.

1) Conformance Evaluation: In order to quantify the similarity of two edge maps, three metrics, Pco (percentage of edge pixels detected by both implementations) Pnd (percentage of edge pixels detected by the original Canny edge detection2) Subjective Testing:
2) Subjective tests: were conducted by having human subjects evaluate the quality of the detected edge maps that are generated by the proposed algorithm and the original Canny for both clean and noisy images, without the subjects knowing which algorithm produced which edge maps, using images from the SIPI Database [20] and the Standard Test Image Database [21].

5. FPGA Implementation Of The Proposed Distributed Canny Edge Detection Algorithm

In order to demonstrate the parallel efficiency of the proposed split Canny edge detection algorithm, we describe an FPGA-based hardware implementation of the proposed algorithm it gives a bird's eye view of the embedded system for implementing the distributed Canny edge detection algorithm based on an FPGA platform. It is composed of several components, including an embedded micro-controller, a system bus, peripherals & peripheral controllers, external Static RAMs (SRAM) & memory controllers, and an intellectual property (IP) design for the proposed distributed Canny detection algorithm. The embedded micro-controller coordinates the transfer of the image data from the host computer (through the PCI e (or USB) controller, system local bus, and memory controller) to the SRAM; then from the SRAM to the local memory in the FGPA for processing and finally storing back to the SRAM. Xilinx and Alter a offer extensive libraries of intellectual property (IP) in the form of embedded micro-controllers and peripherals controller [23], [24]. Therefore, in our design, we focused only on the implementation of the proposed algorithm on the Xilinx Virtex-5 FPGA and the data communication with external SRAMs.

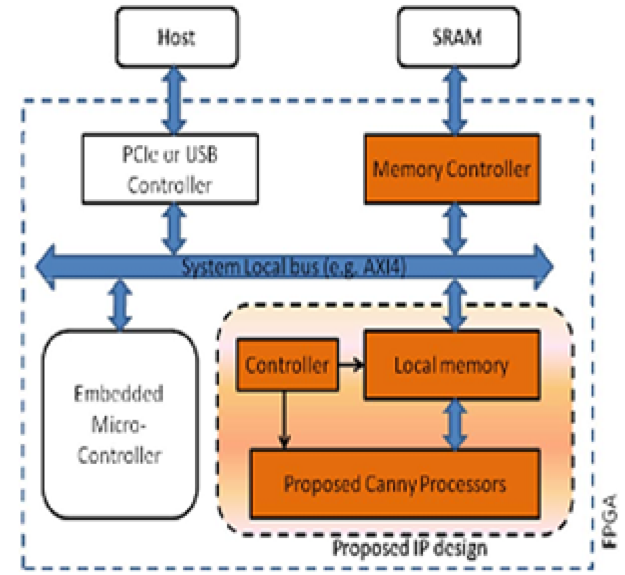


Figure 12: Block diagram of the embedded system for the proposed algorithm.

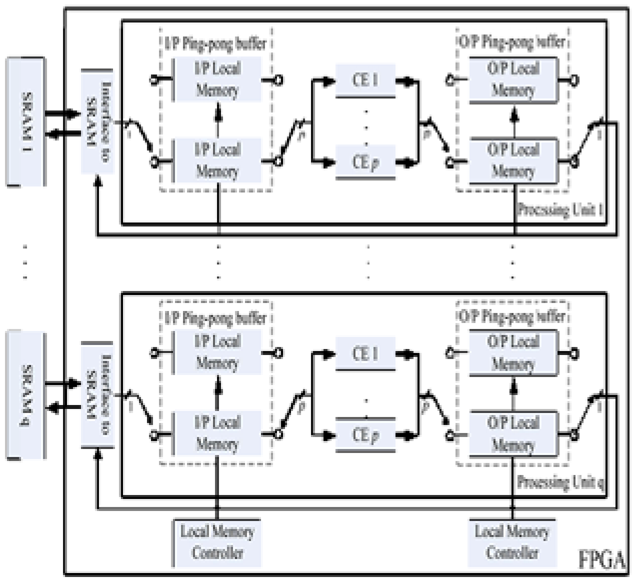


Figure 13: The architecture of the proposed split Canny algorithm.

6. Synthesis Results

The proposed FPGA-based architecture can support multiple image sizes and block sizes. To demonstrate the performance of the proposed system, a Xilinx Virtex-5 FPGA [26] was used to process grayscale images with a block size of 64 × 64. The data width is 16 bits (Q8.7) with 8 bits to represent the integer part since the maximum gray value of the image data is 255, and 7 bits to represent the fractional part since the Gaussian filter parameters are decimals. Ouranalysis shows that 7 bits are sufficient to meet the accuracy requirement of the Gaussian filter parameters, which is typically in the order of 0.001. To store grayscale images, we used the SRAM (CY7C0832BV) [27].

Table 2

Block Size	Number of CE	Occupied slices	Slice Reg.	Slice LUTs	DSP48Es	Total used Memory
64×64	1	747 (2%)	1,270 (1%)	2,578 (2%)	7 (1%)	217(KB) (1%)

Resource Utilization on Xc5vsx240t For 1CE

Table 3

Block Size	Number of CE	Occupied slices	Slice Reg.	Slice LUTs	DSP48Es	Total used Memory
64×64	4	2,988 (8%)	5,080 (4%)	10,312 (8%)	28 (3%)	2023(KB) (10%)

Resource Utilization on Xc5vsx240t For 1PU

Table 4

Block Size	Number of CE	Occupied slices	Slice Reg.	Slice LUTs	DSP48 Es	Total used Memory
64×64	32	23,904 (64%)	40,640 (32%)	82,496 (65%)	224 (25%)	16,184 (KB) (87%)

Resource Utilization on Xc5vsx240t for an 8-PU Architecture

Table 5

	T_{GRAD}	T_{FIR}	T_{NMS}	T_{TC}	T_{TH}
Clock Cycles	9248	16	20	4630	4634

Clock Cycles for Each Unit

This is a dual ported SRAM with 110 pins. The Xilinx Virtex-5 FPGA (XC5VSX240T) has 960 I/O pins and so, to satisfy the I/O pin constraint, the maximum number of PUs is 8 (q = 8). The local memory on the FPGA for a block size of 64 × 64, which is needed to support 8 PUs, is equal to 7 pqm2b = 4046 p Kbits (see Section 5.1), for q = 8, m = 68 (for a 64 × 64 block size and A 3 × 3 gradient mask size), and b = 16. Since the available memory resource on the FPGA is 18,576 Kbits, the p value using the memory constraint is determined to be 4. The p value could have also been constrained by the number of available slices. Since the number of slices for the considered FPGA is very large (37 440) and since each CE only utilizes a small slice percentage (as shown later in Section 5.3.2),the local memory resource in each PU constrains p, the number of CEs in each PU, and not the numbers of slices. Taking all this into consideration, our design has q = 8Pus and each PU has p = 4 CEs. This design is coded in Verilog and synthesized on a Xilinx Virtex-5 device (XC5VSX240T) using the Xilinx's ISE software and verified using Modelsim. According to the 'Place and Route' synthesis report, our implementation can achieve an operating frequency of 250 MHz. But we choose 100 MHz to support a pipelined implementation of SRAM read/write and CE processing as described later in Section 6.2

7. Conclusion

The original Canny algorithm relies on frame-level statistics to predict the high and low thresholds and thus has latency proportional to the frame size. In order to reduce the large latency and meet real-time requirements, we presented a novel Split Canny edge detection algorithm which has the ability to compute edges of multiple blocks at the same time. To support this, an adaptive threshold selection method is proposed that predicts the high and low thresholds of the entire image while only processing the pixels of an individual block. This results in three benefits: 1) a significant reduction in the latency; 2) better edge detection performance; 3) the possibility of pipelining the Canny edge detector with other block-based image codecs. In addition, a low complexity non-uniform quantized histogram calculation method is proposed to compute the block hysteresis thresholds. The proposed algorithm is scalable and has very high detection performance.We show that our algorithm can detect all psycho-visually important edges in the image for various block sizes. Finally, the algorithm is mapped onto a Xilinx Virtex-5 FPGA platform and tested using ModelSim. The synthesized results show 64% slice utilization and 87% BRAM memory utilization. The proposed FPGA implementation takes only 0.721ms (including the SRAM read/write time and the computation time) to detect edges of 512 × 512 images in the USC SIPI database when clocked at 100 MHz. Thus the proposed implementation is capable of supporting fast real-time edge detection of images and videos including those with full-HD content.

References

[1] R. Deriche, "Using canny criteria to derive a recursively implemented optimal edge detector," Int. J. Comput. Vis., vol. 1, no. 2, pp. 167–187, 1987.
[2] L. Torres, M. Robert, E. Bourennane, and M. Paindavoine, "Implementation of a recursive real time edge detector using retiming technique," in Proc. Asia South Pacific IFIP Int. Conf. Very Large Scale Integr.,1995, pp. 811–816.
[3] F. G. Lorca, L. Kessal, and D. Demigny, "Efficient ASIC and FPGA implementation of IIR filters for real time edge detection," in Proc. IEEE ICIP, vol. 2. Oct. 1997, pp. 406–409.
[4] D. V. Rao and M. Venkatesan, "An efficient reconfigurable architecture and implementation of edge detection algorithm using handle-C," inProc. IEEE Conf. ITCC, vol. 2. Apr. 2004, pp. 843–847.
[5] H. Neoh and A. Hazanchuck, "Adaptive edge detection for real-time video processing using FPGAs," Altera Corp., San Jose, CA, USA, Application Note, 2005.
[6] C. Gentsos, C. Sotiropoulou, S. Nikolaidis, and N. Vassiliadis, "Real-time canny edge detection parallel implementation for FPGAs," in Proc. IEEE ICECS, Dec. 2010, pp. 499–502
[7] W. He and K. Yuan, "An improved canny edge detector and its realization on FPGA," in Proc. IEEE 7th WCICA, Jun. 2008,pp. 6561–6564.
[8] J.D.Owens et al., "A survey of general-purpose computation on graphics hardware," Comput. Graph. Forum, vol. 26, no. 1, pp. 80–113, 2007.
[9] Y. Luo and R. Duraiswami, "Canny edge detection on NVIDIA CUDA,"in Proc. IEEE CVPRW, Jun. 2008, pp. 1–8.[11] R. Palomar, J. M. Palomares, J. M. Castillo, J. Olivares, and
[10] J. Gómez-Luna, "Parallelizing and optimizing lip-canny using NVIDIACUDA," in Proc. IEA/AIE, Berlin, Germany, 2010, pp. 389–398.
[11] L. H. A. Lourenco, "Efficient implementation of canny edge detection filter for ITK using CUDA," in Proc. 13th Symp. Comput. Syst., 2012,pp. 33–40.
[12] Q. Xu, C. Chakrabarti, and L. J. Karam, "A distributed Cannyedge detector and its implementation on FPGA," in Proc. DSP/SPE),Jan. 2011, pp. 500–505.
[13] J. F. Canny, "A computation approach to edge detection," IEEE Trans.Pattern Anal. Mach. Intell., vol. 8, no. 6, pp. 769–798, Nov. 1986.
[14] S. Nercessian, "A new class of edge detection algorithms with performance measure," M.S. thesis, Dept. Electr. Eng., Tufts Univ., Medford,MA, USA, May 2009.
[15] P. Bao, L. Zhang, and X. Wu, "Canny edge detection enhancement by scale multiplication," IEEE Trans. Pattern Anal. Mach. Intell., vol. 27,no. 9, pp. 1485–1490, Sep. 2005.
[16] 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.
[17] 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/
[18] D. V. Rao and M. Venkatesan, "An efficient reconfigurable Architecture and implementation of edge detection algorithm Using Handle-C," ITCC, vol. 2, pp. 843 – 847, Apr. 2004.
[19] Shengxiao Niu, Jingjing Yang, Sheng Wang, Gengsheng Chen,"Improvement and Parallel Implementation of Canny Edge Detection Algorithm Based on GPU".
[20] W. He and K. Yuan, "An improved Canny edge detector and its Realization on FPGA," WCICA, pp. 6561 –6564, Jun. 2008.
[21] J. Canny, "A computational approach to edge detection," IEEE Trans. PAMI, vol. 8, no. 6, pp. 679 – 698, Nov. 1986.

Author Profile

V. Shamlee received Bachelor of Engineering degree in Electronics and Communication Engineering from CSI College of Engineering, Ketti .Currently she is doing Master of Engineering degree in VLSI Design at United Institute of Technology, Coimbatore. Her area of interest lies in the field of Image Processing and VLSI Design.

J. Jeyamani is an Assistant Professor at United Institute of Technology, Her area of interest lies in the field of Wireless networks, Image Processing and VLSI Design.