A NOVEL CANNY EDGE DETECTION ALGORITHM FOR REAL-TIME APPLICATIONS

Manoj Kumar Paramasivam and Anima Sahu

Department of Electrical Engineering, Arizona State University, Tempe, Arizona

ABSTRACT

Canny Edge Detection is a widely used image processing technique wherein the amount of image data used for computation is reduced while preserving the structural properties of the image. The algorithm has many use cases across software, hardware and real-time computation applications across Airplane tracking, biomedical imaging, stereo vision etc. In this paper, we approach the canny edge detection algorithm specifically for real time applications where throughput is important. The algorithm is implemented in software and hardware in MatLab and Synopsys Model Compiler and the performance is compared using the rubber band technique. Each stage of the algorithm is broken down and analyzed in simulation and emulation to reduce the maximum image data while achieving high throughput and reducing the processing time.

KEYWORDS

Edge detection, Canny algorithm, Real-time image processing, Hardware acceleration, Rubber band technique,

1. INTRODUCTION

Edge detection has numerous applications in the image processing domain. The primary purpose of edge detection is the reduction of image data while retaining the important information for further processing. This reduces the processing complexity of the following image processing algorithms. The edge detection algorithm finds the boundaries of objects within the image, based on the determination of discontinuities of brightness.

In this paper, real time applications of edge detection has been analyzed and studied. The software based edge detection techniques based on Fourier transforms, fuzzy logic, wavelet transform and derivative operators etc. [1-4] are very time-consuming and not suitable for real time applications. This makes the hardware acceleration of the edge detection algorithms necessary for real time image processing. Some of the many real time applications which use edge detection are the Runway Detection, Airplane tracking, biomedical real time image processing, stereo vision, obstacle detection and avoidance.

The project involves the hardware implementation of the modified canny algorithm with hardware acceleration using Synopsys Model Compiler and its comparison with the software implementation of the algorithm in MatLab.

2. CANNY EDGE DETECTION ALGORITHM DESIGN SPECIFICATION

Canny edge detection algorithm is known for its optimal criteria for edge detection. In the paper [6], John Canny has described an effective edge detection algorithm which constitutes of three

criteria. The first one corresponds to the low error rate. The second criterion is for the localization of the detected edges. The third criterion is elimination of multiple detected edges.

The canny edge detection algorithm is a step-by-step approach as shown in fig. 1. The first step of the algorithm is smoothening of the image to eliminate any noise present.

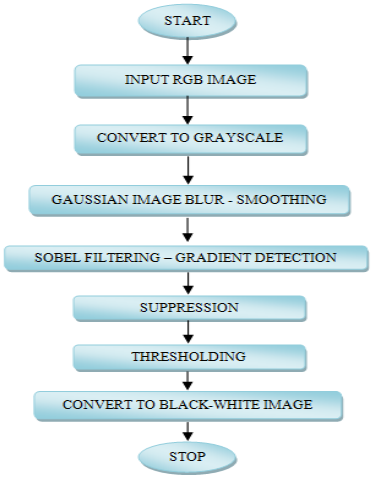


Figure 1. Canny Edge Detection Algorithm Flowchart

3. SOFTWARE IMPLEMENTATION

MatLab simulation is used for software implementation of the Canny edge algorithm. The purpose of software implementation was to compare the performance improvement achieved by the hardware implementation. The implementations discussed in this paper has been implemented on 515 x 512 image resolution.

3.1. Gaussian Smoothing

This step removes the noise from the grayscale image. This is done using a Gaussian filter. The filter operates on the image using a simple mask that is convoluted with the image pixels. The mask slides over the image and the smoothened pixel values are calculated. In our implementation a 5X5 Gaussian filter mask is used. The mask is shown in fig. 2a.

3.2. Gradient Detection

This step determines the edge strength from the smoothened image using gradient detection. The Sobel operator is used for this operation. The Sobel operator is used for 2D spatial gradient measurement on the image. The Sobel operator is a 3X3 matrix as shown in fig. 2b and 2c, corresponding to the x-direction (0˚) and y-direction (90˚) respectively. Using the sobel operator the gradient magnitude (G_mag) and gradient direction (G_dir) are determined using the equations 1 and 2.

$$G\_mag = \sqrt{(Gx2 + Gy2)} \quad (1)$$

$$G\_dir = \arctan(Gy/Gx) \quad (2)$$

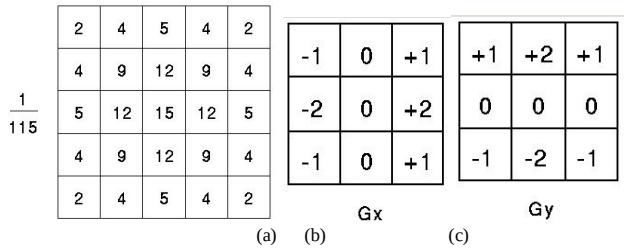


Figure 2. (a) Gaussian mask (b) Sobel 'x' operator (c) Sobel 'y' operator

In this step, the image edges detected in the previous step are suppressed using non-maximum suppression. This is accomplished by tracing the edges and suppressing any pixel value which aren't considered edge and is set to a zero value. This provides with a fine edge in the output image.

3.3. Thresholding

Hysteresis is used in this step by using two thresholds, high and low (T2 and T1). Any pixel in the image that has a value greater than T2 is presumed to be a pixel for the edge. Then, any pixels that are connected to this edge and that have a value greater than T1 are also selected as part of edge. Using this technique eliminates the weak edges, whereas detects the weak edges connected to the strong edges. Finally, the detected edges are converted into a black-white image for better viewing.

4. HARDWARE IMPLEMENTATION

The hardware implementation has been done using MatLab Simulink with the Synphony Model Compiler(SMC) blocks. This has been synthesized for 28nm ASIC target and found the performance comparisons. Different configurations have been implemented to understand the performance optimization. An image is considered as an ensemble of data pixels (R,G,B – 24 bits). In our implementation, the pixels have been processed in parallel in order to reach higher throughputs for real time applications. Each Parallel Element (PE) consists of the same blocks and operates on a 128 X 128 patch of the image, i.e. each PE processes 16384 pixels. Here, each block in a PE has been explained in further detail.

4.1. RGB to Gray Scale Conversion

The first block of the parallel element is the RGB to Grayscale conversion. This is required as the canny edge detection requires a grayscale image and also, it reduces each pixel to an 8-bit value, rather than the 24bit input. The RGB to Grayscale conversion is implemented using the widely accepted grayscale luminosity equation as in equation 3. The SMC model implementation for the block is shown in fig 3.

$$Grayscale = 0.21R + 0.72G + 0.07B \quad (3)$$

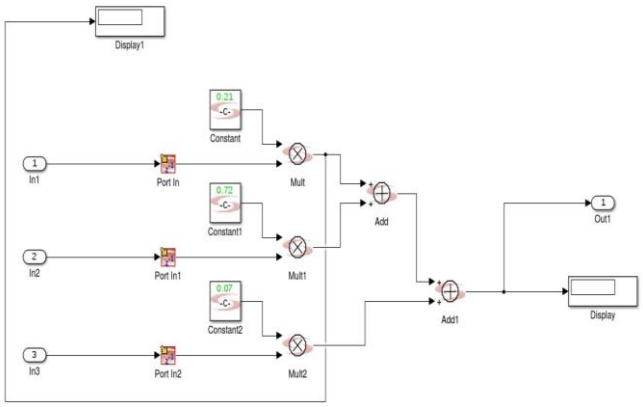


Figure 3. SMC model for RGB to Grayscale conversion subsystem

4.2. Gaussian Smoothing / Blur

The Gaussian smoothing is done using the mask operator shown in fig. 2a. However, in the hardware implementation, two 1D Gaussian masks has been used instead of the 2D mask. 1D mask implementation is less resource intensive than the 2D mask and thus, this technique was utilized. The 2D mask implementation requires the availability of the 25 elements (5X5) to perform the Gaussian operation. The data stream in our PEs is a 1D stream and requires extra memory elements to perform the 2D mask operation. 1D mask is used to accelerate the associated computation in this block.

As the Gaussian mask is circular symmetric, it can be individually split into the x-direction mask and y-direction masks, which are the first row and column of the 2D mask respectively. These two 1D masks were used to blur the image in the x and y direction and then combine the effects to achieve the final smoothened image. Fig. 4 shows the Gaussian smoothing subsystem. FIR filters were designed using the FDA Tool in SMC to implement the masks. FIR filter coefficients were calculated based on the masks and fed to the FDA Tool to design the required mask filters.

For the y-direction of smoothing (column pixels), a delay line (shift register) has been used which holds the pixels of five consecutive rows at any time and thus, the five column pixels can be extracted from the delay line. A multiplexer is used to extract the five column elements to be masked by the following FIR filter. After filtering the data is demultiplexed to obtain the smoothened image pixel data.

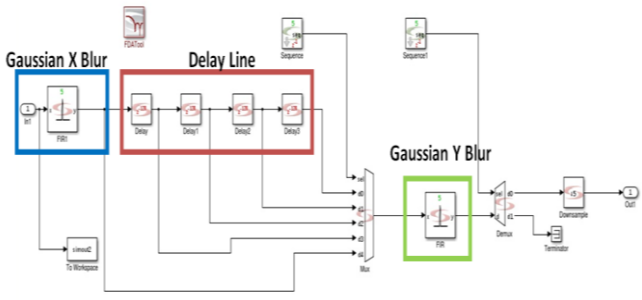


Figure 4. SMC model for Gaussian smoothing subsystem

4.3. Sobel Filtering – Gradient Detection

Gradient detection involves the computation of the gradient magnitude and the gradient direction as given by the equations 1 and 2 respectively. Sobel filter is a 3X3 asymmetric, directional, 2D filter mask, shown in fig. 2b and 2c, and thus, cannot be converted to a 1D mask. In order to perform the Sobel filtering, 9 elements are required to be convoluted with the mask. The Sobel filter uses the FIR filter in the SMC tool. The filter coefficients have been calculated based on the directional masks and provided as input to the FDATool.

Sliding window architecture is required to implement the 2D mask operation. In order to implement this, a delay line (shift register) was used which holds the three consecutive rows for processing of the 9 elements in the sliding window. A delay line provides with the advantage that every pixel data need not be recalled 9 times for mask convolution. While in the delay line, each data pixel gets convoluted with the filter mask to provide the necessary gradient in the direction of the mask.

4.3.1. Gradient Direction

Determining the Gradient direction is based on the equation 2, which is the formula for the vector direction. This involves the inclusion of trigonometric formula of arctan which increases the hardware complexity. In order to avoid using arctan, the design includes four directional sobel filters for 0˚, 45˚, 90˚ and 135˚. The directional filters convolute with the blurred image from the previous step, simultaneously. A comparison stage has been further used to determine which direction has the maximum intensity. This gives the direction gradient associated with each of the data pixel. This technique used is the Compass directional filtering. Fig. 5 shows the SMC model for the Sobel implementation and fig. 6 shows the SMC model for the Compass Sobel filter, G_mag and G_dir blocks.

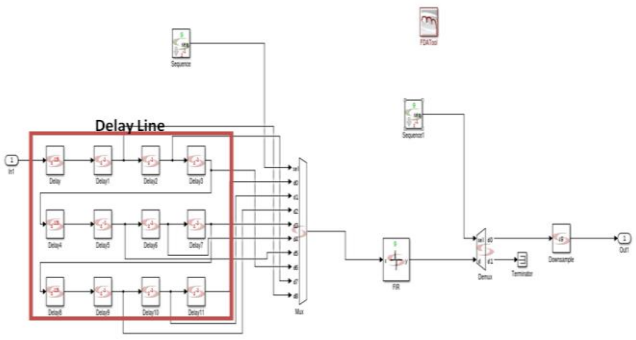


Figure 5. SMC model for the Sobel Filter implementation with Delay Line

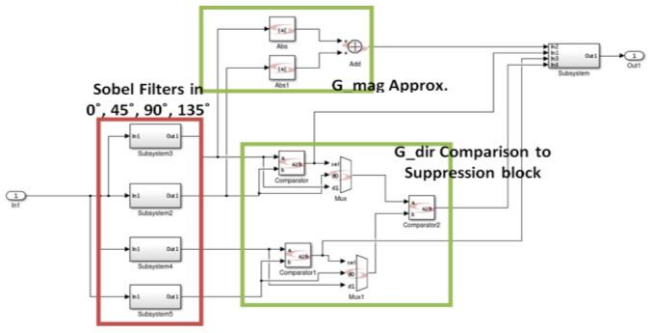


Figure 6. SMC model for the Compass filters and G_mag & G_dir blocks

4.3.2. Gradient Magnitude

The gradient magnitude computation is based on the vector magnitude calculation given in equation 1. The Sobel filter outputs in 0˚ and 90˚ are the Gx and Gy components respectively. This step involves square root calculation which is resource intensive for the hardware implementation. To reduce complexity, an approximated gradient magnitude is used as shown in equation 3. This approximation of Gradient magnitude calculation gives an error factor, which is corrected in the further step of non-maximal suppression.

$$G\_mag = |Gx| + |Gy| \qquad (3)$$

4.4. Non-maximal Suppression

The purpose of non-maximal suppression is to thin the detected edges from the gradient stage. This requires a series of comparison to determine the maximum contrast between the neighborhood pixels in the gradient direction detected. This requires the following steps: 1) Compare the edge strengths of the positive and negative pixels in the gradient direction and 2) Keep the edge pixel only if it has the highest intensity as compared to neighborhood pixels, else make it zero. This requires a 3X3 mask to extract the positive and negative pixels and thus, a delay line has been implemented. A set of comparison operations are carried out in all the four directions and the G_dir calculated direction is considered. Fig. 7 shows the different blocks in this stage.

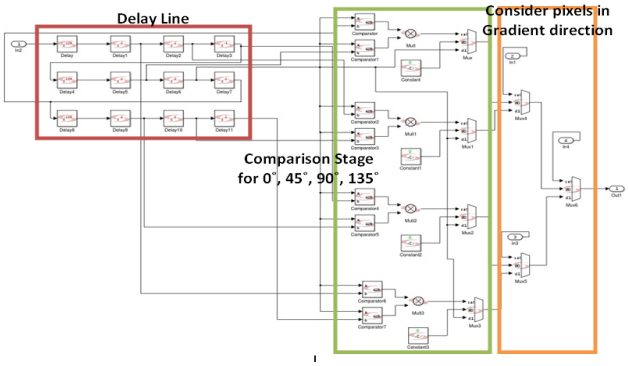


Figure 7. SMC model for Non maximal suppression block

4.5. Two level Hysteresis Threshold

Hysteresis utilizes two threshold values to track the remaining pixels that have not been suppressed from the previous stage. All the values below the lower threshold is set to zero (i.e. discarded) and all the values above the high threshold are set to 1 (edge). After finding the strong edges the values connected to the strong edges and above the low threshold are also set to 1. Any other weak edges i.e. above low threshold, with no path connected to strong edges are set to zero. This technique of using two threshold values to determine edges helps in eliminating weaker edges and gives a finer edge precision at the output.

The threshold comparison occurs in parallel for the low threshold (weak) and high threshold (strong). Determining association between strong and weak edge pixel requires a 3X3 window of pixels. A delay line has been implemented after the strong edge comparison to hold three consecutive rows at any time. For any weak edge pixel, if there is any strong edge pixel in the 3X3 vicinity, the weak edge pixel is also considered as an edge. A series of OR (adder) and a AND (multiplier) operation is done in the edge association step to get the final output in edge assignment step. Fig. 8 shows the SMC model for the Hysteresis Threshold block.

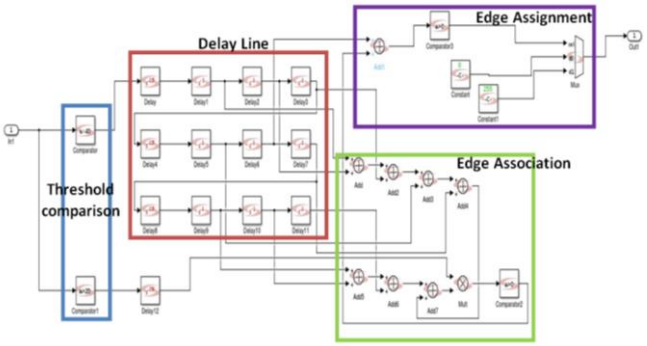


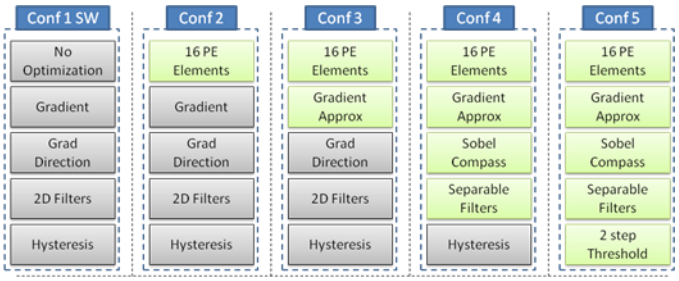
Figure 8. SMC model for Hysteresis Threshold block

5. CONFIGURATIONS & RESULTS

The hardware design for the Canny edge detection algorithm has been implemented on a 28nm Synopsys PDK ASIC target to understand the Latency, Energy and Area performance parameters. Several hardware configurations were implemented to obtain high throughput without affecting the energy and area for the hardware which was the primary objective of this work.

The first step was parallelization using PE elements. In this project, 16 PEs have been implemented. Each PE handles a 128X128 image block. Parallelizing the hardware provided us with a high throughput of 84MPPS. The algorithm was further modified to reduce the data path latency and hardware intensive requirement (i.e. memory, complex math operations etc.). These modifications, as have been explained in Section IV earlier, helped us achieve a throughput of 89MPPS. The hardware design operates at a frequency of 100MHz and has been simulated in the Design Compiler.

Fig. 9 gives the various configurations used and the performance achieved in each case. Configuration 1 is the software implementation which gives a throughput of 0.3MPPS, which might work for real time application of a 512X512 image. However, for a higher resolution image this software implementation will not be able to perform for real time requirement. Configurations 2, 3, 4 and 5 are hardware implementations with added optimizations which have been indicated in green in the fig. 9.



	Conf 1 SW	Conf 2	Conf 3	Conf 4	Conf 5
Area (um²)	-	272584	258421	263392	242422
Power (mW)	-	68.45	61.23	70.28	59.05
Clock Freq. (MHz)	-	100	100	100	100
Throughput (PPS)	0.3 M pixels	84 M pixels	85 M pixels	88 M pixels	89 M pixels

Figure 9. Different Configurations tested and performance achieved

The performance parameters of latency, power and area are governed based on the sensitivity theory for any hardware implementation. They act like the three poles with a rubber band wound around them as shown in fig. 10. The rubber band acts like constraints and parameters adjust themselves based on the other two. For example, to reduce latency and pull the 1/Latency knob further, we increase either the area or power or both. Using the rubber band technique we analyze each parameter's sensitivity with respect to the other two.

The rubber band technique and sensitivity theory has been used to achieve the high throughput while not negatively affecting the area and power for the system design. As can be seen from the results shown in fig. 9, the desired throughput was achieved by the 16 PE implementation (Configuration2). However, the further configurations were implemented to modify the algorithm to reduce the energy and area considerations without negatively affecting the already achieved throughput. The 3-axix performance graph is shown in fig. 11.

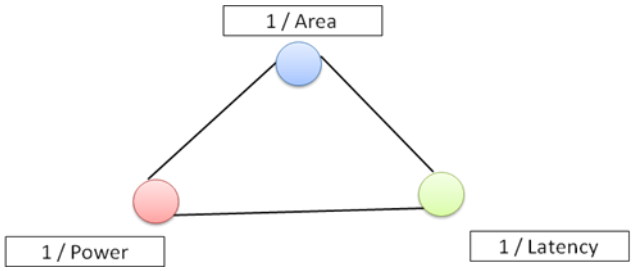


Figure 10. Rubber Band Technique

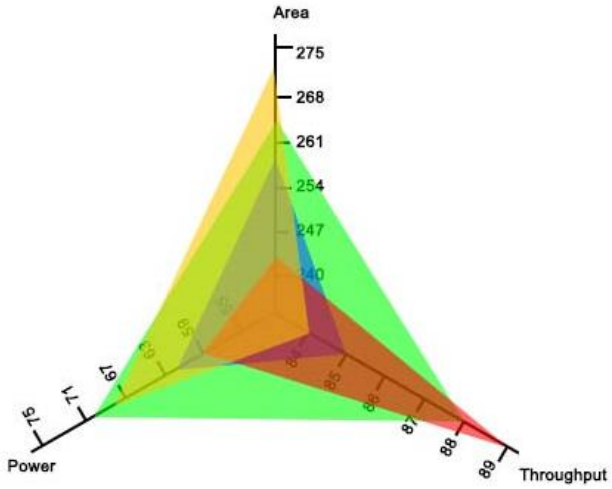


Figure 11. 3-axis Performance Graph for the canny edge detection algorithm

6. CONCLUSION

The canny edge detection algorithm implemented here was able to achieve a high throughput of 89MPPS. Through a series of algorithm improvements, the area and power for the hardware could also be optimized for the design. A sample 512 x 512 image resolution has been used for all the simulations. The system can be used effectively for high resolution images as well. 89MPPS should be able to cater to HD and 4K images as well. However, if there is a requirement of higher throughput, the number of parallel elements can be increased to the

desired level. The total power and area for the system will increase with the number of PE elements due to the increase in hardware devices.

REFERENCES

[1] L. A. Zadeh, "Fuzzy sets," Information and control, pp. 338–353, 1965.

[2] S. Mallat and S. Zhong, "Characterization of signals from multiscale edges," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 14, pp. 710–732, Jul 1992.

[3] A. Safari, C. Niras, and Y. Kong, "Vlsi architecture of multiplier-less dwt image processor," in TENCON Spring Conference, 2013 IEEE, pp. 280–284, April 2013.

[4] T. M. Khan, M. A. Khan, and Y. Kong, "Fingerprint image enhancement using multi-scale DDFB based diffusion filters and modified hong filters," Optik - International Journal for Light and Electron Optics, vol. 125, no. 16, pp. 4206 – 4214, 2014.

[5] T. M. Khan, D. Bailey, M. A. Khan, and Y. Kong, "Real-time edge detection and range finding using FPGA," Optik – International Journal for Light and Electron Optics, vol. 126, no. 17, pp. 1545 – 1550, 2015.

[6] J. Canny, "A computational approach to edge detection," Pattern Analysis and Machine Intelligence, IEEE transactions on, vol. PAMI-8, pp. 679–698, Nov 1986.

[7] Wojciech Mokrzycki and Marek Samko, "Canny Edge Detection Algorithm Modification," Faculty of Mathematics and InformaticsUniversity of Warmia and Mazury, Olsztyn, Poland

[8] A. Amaricai, O. Boncalo, M. Iordate, and B. Marinescu, "A Moving Window Architecture for a HW/SW Codesign Based Canny Edge Detection for FPGA," Proc. 28th International Conference on Microelectronics (MIEL 2012), pp. 393-396, 13-16 May, 2012

[9] Tai Kuang Qing-Xin Zhu Yue Sun, (2011),"Edge detection for highly distorted images suffering Gaussian noise based on improve Canny algorithm", Kybernetes, Vol. 40 Iss 5/6 pp. 883 – 893

[10] Wojciech Mokrzycki and Marek Samko, "Canny Edge Detection Algorithm Modification," ICCVG 2012, LNCS 7594, pp. 533–540, 2012.

[11] Raman Maini and Dr. Himanshu Aggarwal, "Study and Comparison of Various Image Edge Detection Techniques," International Journal of Image Processing (IJIP), Volume (3) : Issue (1)

[12] P.Kalyan chakravathi, K. Vijaya Kumar, P.Devi Pradeep and D. Suresh, "FPGA based architecture for realtime Edge detection," Proceedings of 2015 Global Conference on Communication Technologies(GCCT 2015), pp 12-17, 2015

[13] Xiangjian He, Wenjing Jia and Qiang Wu, "An Approach of Canny Edge Detection with Virtual Hexagonal Image Structure," Proceedings of 10th International Conference on Control, Automation, Robotics and Vision (ICARCV2008, Tier A), pp.879-882, 2008.

[14] D.G. Bariamis, D.K. Iakovidis, and D.E. Maroulis, "An FPGA-based Architecture for Real Time Image Feature Extraction," Proceedings of the 17th International Conference on Pattern Recognition, p 801–804, ICPR2004.

[15] Qian Xu, Chaitali Chakrabarti and Lina J. Karam, "A Distributed Canny Edge Detector And Its Implementation on FPGA," IEEE Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE), 2011.

[16] Zhengdong Xu, Kui Yuan and Wenhao He, "An implementation method of Canny edge detection algorithm on FPGA," IEEE Electric Information and Control Engineering (ICEICE), 2011

[17] Wenhao He, Kui Yuan, "An improved Canny edge detector and its realization on FPGA," Intelligent Control and Automation, 2008. WCICA 2008.