Automatic Threshold Selection Based on Histogram Gaussian Estimation Method in FPGA

Deng-Yuan Huang[1], Ta-Wei Lin[1] and Wu-Chih Hu[2]

[1]Department of Electrical Engineering, Dayeh University,

168 University Rd., Dacun, Changhua 515, Taiwan

[2]Departement of Computer Science and Information Engineering, National Penghu University of Science and Technology,

300 Liu-Ho Rd., Makung, Penghu 880, Taiwan

{kevin@mail.dyu.edu.tw, daweimailbox@gmail.com, wchu@npu.edu.tw}

***Abstract*:** A fast and efficient algorithm called HGEM (Histogram-based Gaussian Estimation Method) based on an FPGA (Field Programmable Gate Array) is developed to automatically determine a threshold value for a Sobel edge detector. In comparison with Otsu's method based on a discriminant criterion, the proposed method is more efficient in computing performance. The proposed method is also simple to be implemented on the FPGA since it avoids the repetitious iterations and complex arithmetic operations in Otsu's thresholding procedures. The relative error (RE) of HGEM to Otsu's method is utilized to measure the closeness of the thresholds obtained by the two methods. The relative error is less than 1.50% for all the test images, indicating that the proposed method has the approximately same accuracy as that of Otsu's method. Timing simulations show that the FPGA circuits can run at a speed of up to 193.9 MHz, which is equivalent to a theoretical frame rate of 1,479 frame/s for a gray-level image of 256×256. This result confirms that the proposed hardware architecture can achieve the requirements for a real-time image processing system.

***Keywords*:** Otsu's method; binary thresholding; image segmentation; field programmable gate array (FPGA)**.**

1. Introduction

Automatic thresholding is a very straightforward and effective technique used in the fields of image processing, pattern recognition and computer vision. However, it requires an adequate threshold value to extract objects of interest from their background, since objects in an image have their own distinct gray-level distributions. Thresholding methods are widely used in many application domains, such as human action recognition [1], optical character recognition (OCR) [2],[3], automatic defect inspection [4],[5], video change detection [6]-[8], moving object segmentation [9]-[12], and medical image diagnoses [13],[14]. As a fundamental task for image preprocessing, many researchers pay much attention to the method of how to determine appropriate thresholds.

These applications demand real-time performance and hardware implementation, especially for an FPGA, is essential to increase the computational efficiency of thresholding procedures. Hence, the choice of a thresholding method for implementation on an FPGA board is important. In binary thresholding for image segmentation, Otsu's method [15] is a very popular global automatic thresholding technique; it selects an optimum threshold by maximizing the between-class variance in a gray-level image. However, the basic Otsu thresholding computations involve repetitious iterations of the zero- and first-order cumulative moments of a gray-level histogram, which requires a great number of complex arithmetic operations such as multiplications and divisions. The heavy computational resource makes Otsu's method unsuitable for a high-speed low-cost implementation in FPGA.

Otsu's method is simple to be implemented in software, but it is less efficient when implemented in FPGA circuits. Tian *et al.* [16] introduced a binary logarithmic conversion unit (LCU) to implement Otsu's method by eliminating the complex divisions and multiplications in the computations of between-class variances. The hardware was synthesized with Synplicity Synplify Pro 7.0.3 targeted at the Xilinx Virtex XCV800 HQ240-4 FPGA device. The results for implementations on the FPGA platform showed that their method is 2.75 times faster because it occupies only $1/6^{th}$ of the FPGA slices required by a direct implementation. The introduction of an LCU can avoid the complex computations of divisions and multiplications, but repetitious computations are still required to search for the maximum between-class variance to determine an optimum threshold.

To eliminate both the repetition and complex arithmetic operations in the computations of between-class variance, we present a fast algorithm called HGEM (Histogram-based Gaussian Estimation Method), which is based on the Gaussian distributions of a histogram to determine an optimal threshold for gray-level images. The proposed method is relatively simple and efficient for implementation on an FPGA platform when compared to the basic Otsu thresholding procedures. We also develop a Sobel-based edge detector in the FPGA circuits as a target platform for the HGEM. To detect the presence of an edge pixel in an image, an appropriate threshold value is required to compare it with the magnitude of the Sobel

gradient. Therefore, HGEM can be used to choose the optimum threshold for the Sobel-based edge detector.

Most algorithms for edge detection need to perform a convolution with an image in the spatial domain using a specific mask like a Sobel operator. Benkrid *et al*. [17] proposed a general framework which is built on a library of hardware skeletons for FPGA-based image processing. Two methods, online arithmetic and 2's complement LSBF (least significant bit first) with bit serial transfer, were presented to implement the Sobel-based edge detector on an FPGA board. Time simulations revealed that for a 256×256 gray-level image, the Sobel-based edge detector can run at a speed of 75 MHz, which leads to theoretical frame rates of 88 and 104 frame/s for online arithmetic and 2's complement LSBF methods, respectively. However, [17] did not explicitly describe the determination of the threshold required to establish the presence of an edge pixel. Other studies [18]-[20] have also omitted this description.

Rosas *et al*. [18] utilized a SIMD (single instruction multiple data) architecture based on an FPGA which was connected to two external RAMs modules. One of the RAMs modules was used to store the image captured by a CMOS sensor, and the other was used to store the image processed by the FPGA. In our study, instead of using external RAMs modules, the proposed hardware architecture uses built-in dual-port block RAMs to implement the HGEM algorithm targeted at a Sobel-based edge detector because it can store great amounts of data and access it quickly.

To evaluate the accuracy of the optimum threshold obtained for an image, Sezgin and Sankur [21] employed the following five methods to assess 40 existing thresholding algorithms: misclassification error (ME), edge mismatch (EMM), region non-uniformity (NU), relative foreground area error (RAE), and modified Hausdorff distance (MHD). In this paper, the method of ME is used to evaluate the accuracy of HGEM and the Otsu method.

This paper presents an efficient framework for threshold determination based on the FPGA using the HGEM algorithm. The rest of this paper is organized as follows: Section 2 briefly describes the proposed system architecture. Section 3 then gives a detailed description of the proposed HGEM method. The experimental results are discussed in Section 4, and Section 5 contains the concluding remarks of this work.

2. System Architecture

The proposed architecture for an FPGA-based image processing system is shown in Fig. 1. The hardware was implemented using a Xilinx ISE 8.1i IDE (Integrated Development Environment) tool on the ML401 Xilinx Virtex-4 (XC4VLX25) FPGA based board. This system consists of the following primary components: a data transmission unit, an image segmentation unit, a moving window generator, a three-stage Sobel pipeline unit, and a threshold estimation unit. The data transmission unit is designed to transfer original image pixels from a PC to the FPGA through a UART (universal asynchronous receiver and transmitter) module. To perform convolution, an input image with $m\times n$ pixels has to be convoluted with a $p\times q$ mask. The moving window generator is used to sequentially extract a $p\times q$ window of neighboring pixels from the input image. The three-stage Sobel pipeline unit is adopted to carry out the convolution of the Sobel gradient operator with the window of image pixels acquired by the moving window generator. The threshold estimation unit is an implementation of the proposed HGEM algorithm, which performs histogram statistics, threshold search, and threshold determination. The image segmentation unit is to binarize input image by a threshold that is determine by the threshold estimation unit.

A threshold value is required for segmenting a gray-level image. An algorithm called HGEM is proposed to replace the Otsu's method based on the feasibility of implementation on the FPGA device. The proposed algorithm HGEM, as implemented in the threshold estimation unit, involves the computing procedures including histogram statistics, threshold search, and threshold determination. As illustrated in Fig. 1, when the convolution operation is complete, the processed image (or output image) is stored in block RAMs and then sent back to the PC through the UART module for further verification. The details of the data transmission unit, the moving window generator, and the three-stage Sobel pipeline unit are described in the following sections.

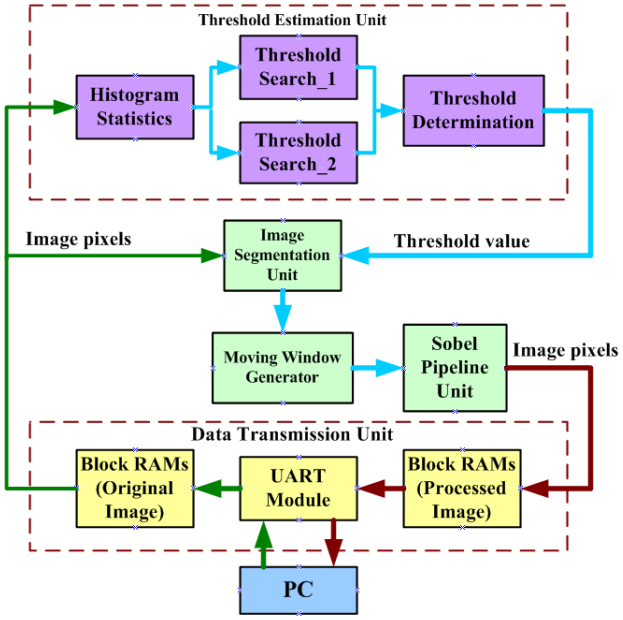


Figure 1. System architecture for threshold determination targeted at the Sobel edge detector

2.1 Data transmission unit

To verify the correctness of the proposed architecture shown in Fig.1, the output image stored in block RAMs, which can be transmitted to a PC by the UART module, is compared with that calculated by software. The architecture of data transmission unit is shown in Fig. 2. This unit primarily serves the following two functions (1) to transmit the original image from a PC to the FPGA, and (2) to send back the output image from the FPGA to the PC.

The byte data of the image is sent by a PC through the UART pin RXD using serial transmission to the UART_Rx module on the FPGA, as shown in Fig. 2. When the UART_Rx module has completely received one byte of image data, it

sends the RXD_data[7:0] and RXD_ready signals. RXD_ready is used to trigger the address controller to address the memory locations of block RAMs A to store the image data that is contained in register RXD_data[7:0]. When the image data has been fully transmitted, the address controller stops the action of writing the image data into block RAMs A to avoid writing error. The image can then be processed in the FPGA circuits.

When the output image has been produced, one can push the transmit button with debouncing capability to trigger the signal of TXD_start to initiate the transmission of the image data from the UART_Tx module to a PC. When one byte of the image data has been transmitted, the signal of TXD_busy is pulled down to a low level to trigger the address controller to acquire the next byte of the image data from block RAMs B into register TXD_data[7:0]. Then, the UART_Tx module sends the processed image data back to the PC until the total image has been completely transmitted. With the aid of this unit, the transmission of the original image and the verification of the output image can be easily achieved.

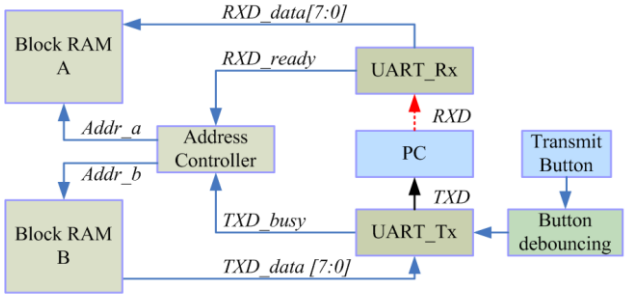


Figure 2. Architecture of the data transmission unit

2.2 Moving window generator

To compute the Sobel gradient, a 3×3 (i.e., p=q=3) window of neighboring pixels extracted from the input image is required for convoluting with a Sobel gradient operator. This neighborhood window then moves over the whole image until an output has been produced for all pixels. Generally, it is not practical to store the whole image in RAMs before starting computations due to the limited CLBs (Configurable Logic Blocks) in FPGA. A better way is to only store the image pixels required to perform the current convolution operation.

Figure 3 shows the architecture of the moving window generator which comprises nine flip-flops and two line buffers (or FIFOs; first in first out). The line buffers (i.e., FIFO A and B) and flip-flops are used to store one row of image data with a dimension of n, with each grayscale pixel represented by 8 bits. Generally, when a $p \times q$ convolution mask is applied, $[(p-1) \times n+q] \times 8$ registers are required. Line buffers can be implemented using either shift registers or block RAMs in FPGA. Generally, when an FPGA has no built-in block RAMs, the only way to implement the line buffer is to use shift registers [17],[20],[22],[23], which usually consume a large number of FPGA gates. For example, when a 3×3 convolution window is applied to a 256×256 (i.e., m=n=256) image with 8-bit pixels, (2×256+3)×8=4,120 flip-flops are required. That is about 19% (4,120/21,504*100%) of all the available flip-flops in the Virtex-4 (XC4VLX25) FPGA used in this case. Furthermore, if the size of the convolution mask or image becomes larger, the required gate counts or CLBs of the FPGA will increase accordingly. To reduce the consumption of FPGA CLBs, [24] utilized block RAMs to implement the line buffers.

This paper also adopts the block RAMs to implement the line buffers using a framework similar to that used in [24]. Using block RAMs to implement the line buffers not only reduces the consumption of FPGA gates, but also lowers the required routing of logic elements. Less routing of logic elements implies that a higher operation speed of the FPGA circuits can be achieved. This is verified by Figs. 4 and 5.

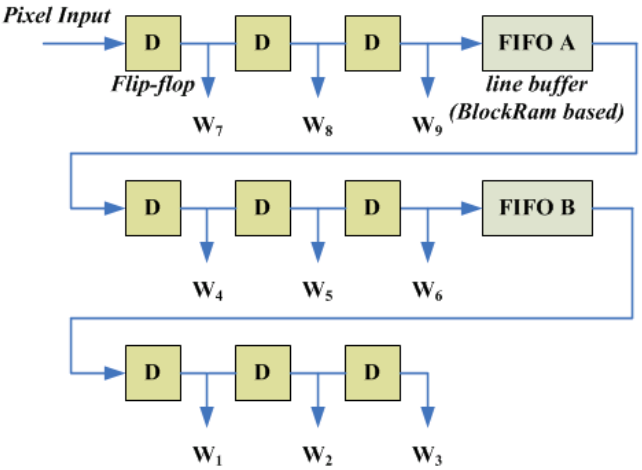


Figure 3. Architecture of the moving window generator

The FPGA gate counts for block RAM-based and shift register-based FIFOs under various image sizes are shown in Fig. 4. The results show that the required gate count increases from 4,000 to 38,000 when the image size grows from 32×32 to 512×512 pixels for the case of a shift register-based FIFO. However, when block RAMs is used, the required gate count remains approximately constant at about 2,000 with increasing image size. Figure 5 shows the effects of image size on the operational speed of the FPGA circuits. Generally, the total speed decreases when the image size becomes larger for both shift register-based and block RAM-based FIFOs. Sharp declines of speed can be observed when the image size increases from 128×128 to 512×512. In addition, the average speed for a block RAM-based FIFO is much higher than that of a shift register-based one. Figs. 4 and 5 show that the operational speed can be increased by 15.6% and that the logic elements of the FPGA can be reduced by 74.2% when block RAMs is used to replace shift registers to implement the FIFOs in the moving window generator.

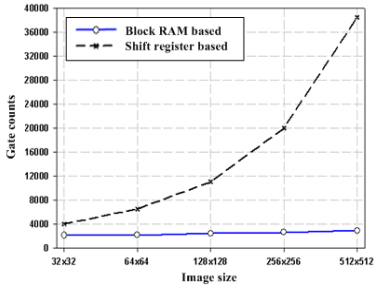


Figure 4. Effect of image size on the gates consumed in FPGA

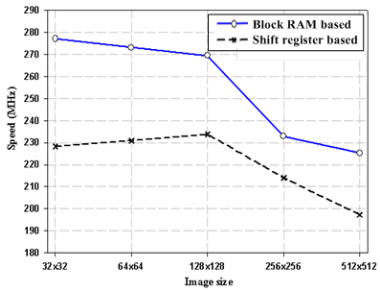


Figure 5. Effect of image size on the operational speed of the FPGA chip system

2.3 Three-stage Sobel pipeline unit

Many methods for edge detection have been implemented with convolution masks, and most are derived from the differential operators, which measure the rate of change in brightness of an image. Generally, a large change in brightness in an image over a short spatial distance (typically one pixel) reveals the existence of an edge. The most popular convolution mask used in edge detection is the Sobel gradient operator, which looks for edges in both the horizontal and vertical directions and then combines this information into a single metric, as shown in Fig. 6. The convolution can be carried out with the Sobel gradient operator as follows:

$$G_x = (w_7 + 2w_8 + w_9) - (w_1 + 2w_2 + w_3) \quad (1)$$

$$G_y = (w_3 + 2w_6 + w_9) - (w_1 + 2w_4 + w_7) \quad (2)$$

where Gx and Gy are called the "row mask" and the "column mask," respectively. Since both have the same computational complexity in performing the convolution operation, this paper only implements horizontal edge detection, i.e., Gx, to avoid detecting redundant edge information.

w1	w2	w3
w4	w5	w6
w7	w8	w9

Coefficient matrix

-1	-2	-1
0	0	0
1	2	1

Gx

-1	0	1
-2	0	2
-1	0	1

Gy

Figure 6. Sobel masks used to compute gradients Gx and Gy

Figure 7 shows the architecture of the three-stage Sobel pipeline unit. In this study, three pipelines are employed to improve the performance of the system. First, stage-1 pipelining deals with the additions and multiplications of the image window pixels with the Sobel gradient operator. In this stage, four additions and two multiplications are required, where multiplication is performed using one shift-left operation rather than using a multiplier to save logic elements. Stage-2 determines the absolute value for Gx, and stage-3 outputs an enhanced edge detection image. When synthesizing individually, this unit can run at a speed of up to 238.2 MHz.

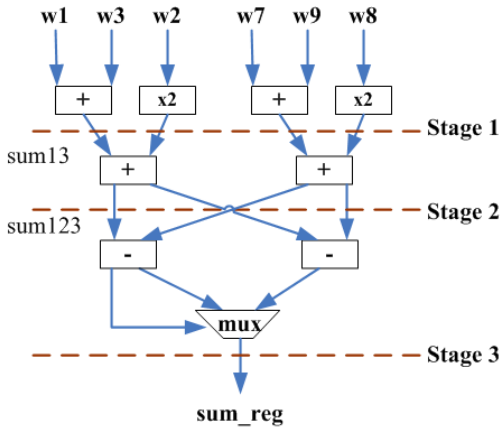


Figure 7. Architecture of the three-stage Sobel pipeline unit

3. Threshold selection algorithm

In an image processing system, the success of image segmentation highly depends on the capabilities of the thresholding method to determine an optimum threshold. Lee *et al.* [25] adopted the histogram concavity technique to locate the optimal threshold value. In their method, the slopes of all the line segments are calculated from the starting gray level. Then, the background peak, Bp, with the greatest slope can be obtained. Similarly, the object peak, Op, can be secured starting from the opposite direction. As a result, the optimal threshold can be found somewhere between Bp and Op. However, this method fails to find the optimal threshold for the special case when Bp meets Op due to extremely high histogram data in a gray level.

Some heuristic approaches have been presented to determine the optimal threshold. El-Khamy *et al.* [26] proposed a so called "Modified Fuzzy Sobel" method that uses a fuzzy reasoning-based algorithm to detect the edges of an image. They first divided an image into two fuzzy regions, i.e., the Fuzzy Smooth region and the Fuzzy Edge region, and then constructed a difference histogram from the input image. The four threshold values used to define the boundaries of the image fuzzy region were used to build a membership function to determine the optimal threshold.

The methods proposed by [25],[26] are relatively simple to implement in software, but they are quite difficult to implement in FPGA circuits due to the determination of varying slopes for [25], and the calculation of the difference histogram for [26]. To achieve much higher accuracy in thresholding estimation, a lot of studies [27]-[29] have used the entropic thresholding technique to find an optimal threshold instead of adopting histogram shape-based methods. However, this method needs to calculate the probability distributions for the edge and non-edge pixels, making it computationally expensive and hard to implement on an FPGA. To balance the computational cost and thresholding estimation accuracy, the present study proposes a histogram-based Gaussian estimation method (HGEM), which is not only easily implemented in FPGA circuits, but also provides a more reasonable threshold value.

3.1 Histogram-based Gaussian estimation method (HGEM)

HGEM is based on the analysis of the gray-level probability density function (pdf) for an image[30]. When the histogram is modeled as two different Gaussian functions, as shown in Fig. 8, with means and variances (μ_1 , σ_1^2) and (μ_2 , σ_2^2), respectively, the histogram function becomes:

$$p(z) = \frac{P_1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(z-\mu_1)^2}{2\sigma_1^2}} + \frac{P_2}{\sqrt{2\pi}\sigma_2} e^{-\frac{(z-\mu_2)^2}{2\sigma_2^2}} \quad (3)$$

and

$$P_1 + P_2 = 1 \quad (4)$$

where z denotes gray level values, and P_1 and P_2 are the probabilities of occurrence of the two classes of pixels, respectively. To find the optimal threshold value *T* in Fig. 8, erroneous classifications, which assign a background pixel to the object, and vice versa, should be minimized.

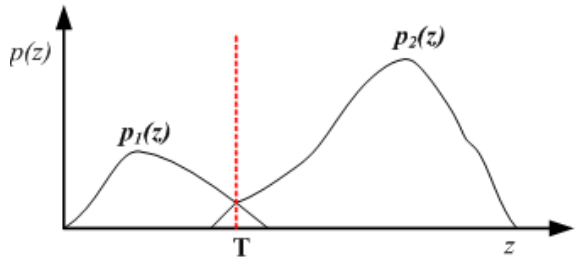
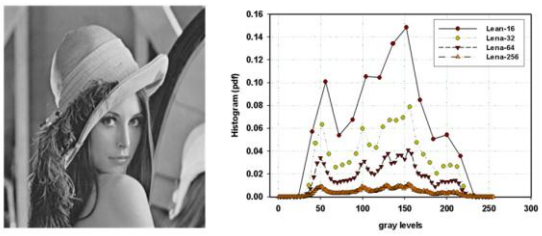


Figure 8. Graph of the probability density function (pdf) for gray-level distribution

To greatly reduce the hardware resources required to implement HGEM on the FPGA, we performed histogram binning by employing wider bin widths. In this study, 16 bin groups, which contain 16 gray levels in every group, are employed to compute the histogram of gray levels to find the optimal threshold. One may argue when the bin width is beyond a certain limit, it may destroy the modes or the valleys in between. However, if the bin width is constrained within a reasonable range, the fine characteristics of the histogram in an image can still be retained. Hence, the operation of "histogram binning" greatly decreases the computational complexity and significantly reduces the required logic elements in the FPGA. Figure 9 shows the histogram of various bin groupings, i.e., 16, 32, 64, and 256, for the test image "Lena". As shown in this figure, the fine characteristics (i.e., valleys in between) of the histogram are similar for all the cases even when the widest bin (i.e., Lean-16) is used. Consequently, the bin width adopted falls into a reasonable range without missing the fine characteristics of the histogram.



(a) Lena (b) Histogram for various bin groupings

Figure 9. Histogram for various bin groupings for test image "Lean"

To efficiently determine the optimal threshold T shown in Fig. 8, the 16 bin grouping is employed. The index value (0 to 15) of the counter and count value of the histogram are used to estimate the Gaussian distribution of an image. Based on the index value and count value, the Gaussian distributions can be categorized into four types. The details of the HGEM algorithm used to determine the optimal threshold are described below.

To complete the histogram of gray levels in an image, the counters are first labeled C0 to C15. Hence, when the values of gray levels are in the ranges of 0 to 15, 16 to 31, …, and 240 to 255, they will be grouped into counters C0, C1, …, and C15, respectively. Then, the histogram can be modeled as two distinct Gaussian functions, divided into the left region, i.e., C0 to C7, and the right region, i.e., C8 to C15. Next, we can search for the index values, max1_1 and max1_2, corresponding to the first two largest count values in the direction from C7 to C0 in the left region. Similarly, the index values, max2_1 and max2_2, can be obtained from C8 to C15 in the right region. Typical search results are shown in Fig. 10. Two of the four index values, i.e., th1 and th2, can be selected based on the type of Gaussian distribution to which the shape of the image histogram belongs. Finally, the threshold value *T* can be calculated as 16×(th1+th2)/2. The HGEM algorithm for finding th1 and th2 is described in the style of the C-language for the following four cases.

$$\begin{aligned} &if\ (\max1\_1 \neq 7\ and\ \max2\_1 \neq 8) \\ &\quad th1 = \max1\_1\ and\ th2 = \max2\_1 \end{aligned} \quad (5)$$

$$\begin{aligned} &if\ (\max1\_1 = 7\ and\ \max2\_1 = 8) \\ &\quad th1 = \max1\_2\ and\ th2 = \max2\_2 \end{aligned} \quad (6)$$

$$\begin{aligned} &if\ (\max1\_1 \neq 7\ and\ \max2\_1 = 8) \\ &\quad th1 = \max1\_1 \\ &if\ (C[\max2\_1] \neq 0\ or\ \max1\_2 > \max1\_1) \\ &\quad th2 = \max2\_1 \\ &else \\ &\quad th2 = 1\ (\text{minimum}) \end{aligned} \quad (7)$$

$$
\begin{aligned}
&\textit{if}\ \ (\max 1\_1 = 7\ \textit{and}\ \ \max 2\_1 \neq 8) \\
&\quad th2 = \max 2\_1 \\
&\textit{if}\ \ (C[\max 1\_1] \neq 0\ \textit{or}\ \ \max 2\_2\ <\ \max 2\_1) \\
&\quad th1 = \max 1\_1 \\
&\textit{else} \\
&\quad th1 = 14\ \ (\text{maximum})
\end{aligned}
\tag{8}
$$

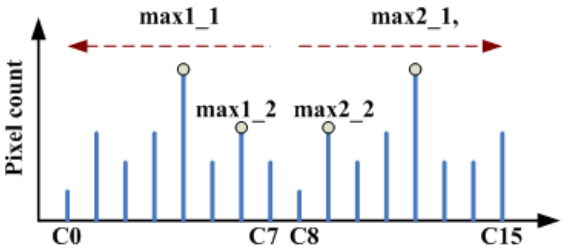
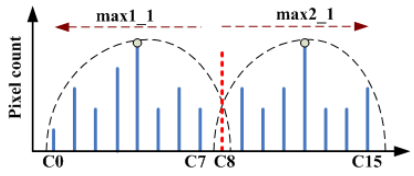
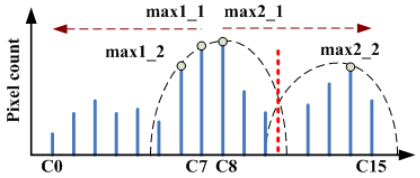


Figure 10. Typical distribution of the histogram with 16 counters

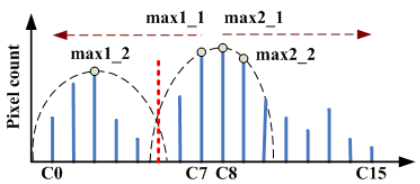
Equation (5) represents the first type of Gaussian distribution shown in Fig. 11(a). If the two largest count values with corresponding index values max1_1 and max2_1 are found on the opposite side of the histogram, the possible Gaussian distributions can be estimated around max1_1 and max2_1, as indicated in Fig. 11(a). Hence, the index values th1 and th2 can be selected as max1_1 and max2_1, respectively. Equation (6) denotes the second type of Gaussian distribution with the first two largest count values in the central region, i.e., corresponding to index values 7 and 8, as shown in Fig. 11(b) and (c). Thus, one possible Gaussian distribution can be estimated in the central region, but the other may be in the right region, as shown in Fig. 11(b), or in the left region, as shown in Fig. 11(c). Consequently, the index values of th1 and th2 should be determined as max1_2 and max2_2, respectively. However, if we choose th1 as max1_1 and th2 as max2_1, the threshold value, 16×(th1+th2)/2, must be in the central region of the histogram, implying that a greatly erroneous classification will be raised.



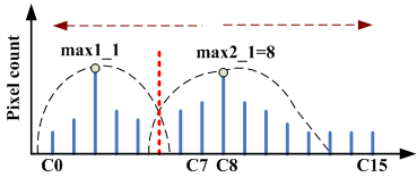
(a). First type of Gaussian distribution



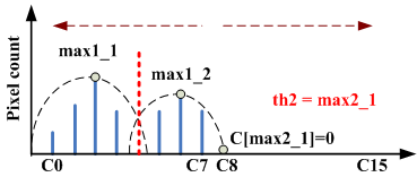
(b). Second type of Gaussian distribution in situation-1



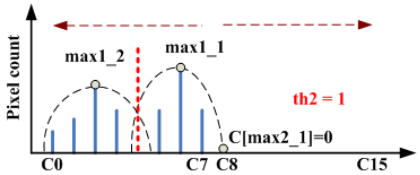
(c). Second type of Gaussian distribution in situation-2



(d). Third type of Gaussian distribution in situation-1



(e). Third type of Gaussian distribution in situation-2



(f). Third type of Gaussian distribution in situation-3

Figure 11. Analysis of Histogram-based Gaussian Estimation Method

The third type of Gaussian distribution is more complicated than the first and second ones, as shown in Fig. 11(d)-(f). If the first two largest count values are not next to each other in the central region, say max1_1≠7 but max2_1=8, one possible Gaussian distribution can be modeled around max1_1, as indicated in Fig. 11(d)-(f), and then th1 can be evaluated as max1_1. However, when the count value C[max2_1] is not equal to zero, the other possible Gaussian distribution can be estimated around max2_1 (see Fig. 11(d)), and th2 should be selected as max2_1. On the other hand, when the count value C[max2_1] is zero, no gray levels are larger than 128, as shown in Fig. 11(e) and (f). Hence, the other possible Gaussian distribution can be modeled around max1_2. As a result, th2 can be chosen as max2_1 when max1_2>max1_1, or 1 when max1_2<max1_1. As can be expected, the third type of Gaussian distribution often occurs in a darker image. The searching method of th1 and th2 (see Eq. (8)) for the fourth type of Gaussian distribution is similar to that of the third type but there are no gray levels smaller than 128. This always happens in a brighter image. Since the searching method is similar to that of the third type, its discussion is omitted here.

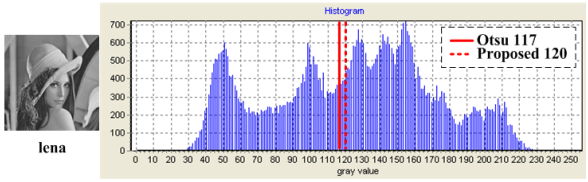
3.2 Comparison of HGEM and Otsu's method

Sezgin and Sankur *et al.* [21] conducted an exhaustive survey of 40 selected image thresholding methods. The results confirm that the thresholding evaluation rank of 40 NDT (nondestructive testing) images according to the overall average quality score for Otsu's method is relatively high, with a rank of 6 and an average score of 0.318. This indicates that Otsu's method can provide a reasonable threshold value for image segmentation. Here, the threshold estimations of HGEM are compared with those of the Otsu method.

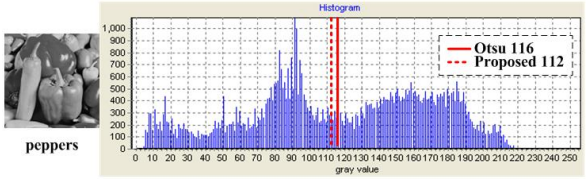
The testing images used in this study consists of natural images (see Fig. 12(a)-(c)) and artificial images (see Fig.

12(d)-(g)), where Fig. 12(d) and (e) are adopted from [31]. To evaluate the accuracy of threshold estimations by HGEM, an artificial image is much better than a real-world one. The threshold values estimated by HGEM for images Lena and Peppers, as shown in Fig. 12(a) and (b), respectively, are very close to those evaluated by Otsu's method. For the image Twins, as shown in Fig. 12(c), HGEM provides a more reasonable threshold estimation, which is much closer to the deeper valley than that of Otsu's method, implying that HGEM can find a satisfactory threshold value even for an image histogram with a wide flat valley. A similar result was obtained for Fig. 12(d). That is, HGEM provided a threshold value exactly in the valley between the two peaks of Gaussian distributions. However, the threshold value estimated by Otsu's method was shifted to the edge part of the right Gaussian distribution. Furthermore, when three objects appear in one image, as shown in Fig. 12(e), an appropriate threshold value was obtained by both methods.

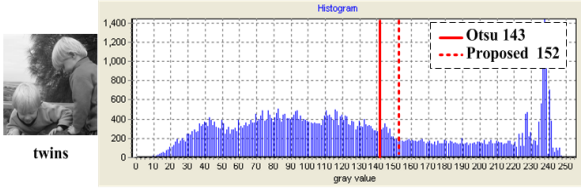
Images with different luminance levels were also examined by HGEM to verify the robustness of the threshold estimations. Images with low luminance, as shown in Fig. 12(f), and with high luminance, as shown in Fig. 12(g), were tested using HGEM and Otsu's method. The results indicate that a reasonable threshold value can be obtained using either method, even for images with large variations in luminance. However, HGEM is computationally efficient; it avoids both repetitious iterations and complex arithmetic operations that are required to compute the between-class variance when using Otsu's method.



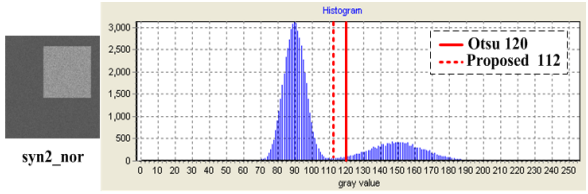
(a). Lena



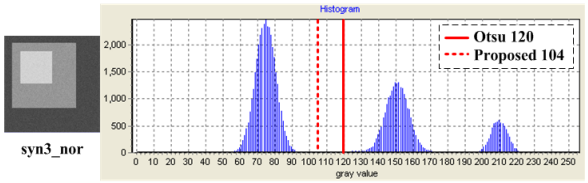
(b). Peppers



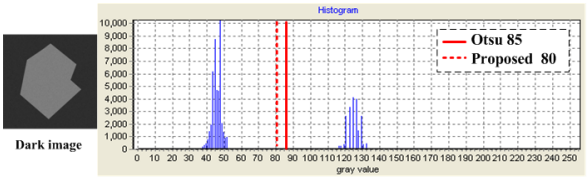
(c). Twins



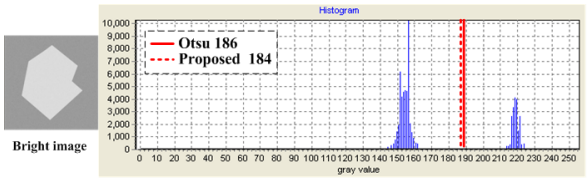
(d). Two Objects



(e). Three Objects



(f). Low luminance



(g). High luminance

Figure 12. Testing images for threshold estimation

To visually compare the segmented results obtained by HGEM and Otsu's method, three 256×256 test images (i.e., Lena, Peppers, and Twins), with each pixel represented by 8 bits, were used. The segmented images with their corresponding thresholds are shown in Fig. 13. The figure shows that the thresholds evaluated by HGEM are very close to those of Otsu's method, indicating that a closely visual perception between them can be achieved.

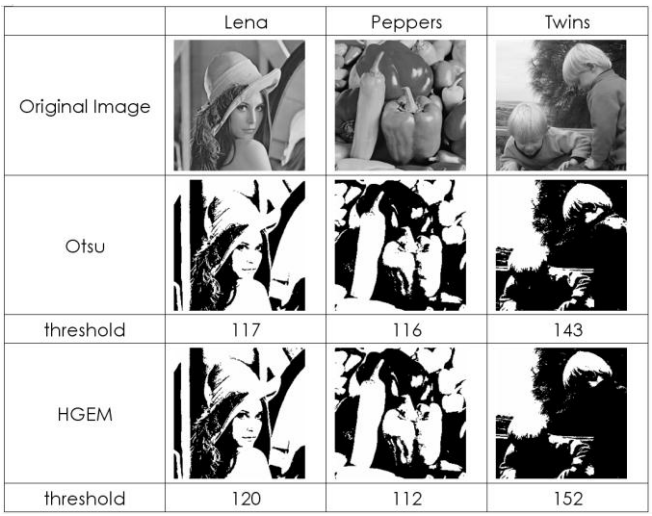


Figure 13. Segmented images of binary thresholding for HGEM and Otsu's method

A comparison of accuracy using the ME method [21] for HGEM and Otsu's method was performed. Gray-level images extracted from the CEDAR database of handwritten words with 29 test images [32], and from the FVC2000 database of fingerprints with 70 test images [33], were employed. Some typical samples are shown in Figs. 14 and 15 with the corresponding ground-truth images. The ground-truth images can be obtained by visually determining the valley of the histogram of the test images. The average results of ME for 29

test images of the CEDAR database and 70 test images of the FVC2000 database were used to evaluate the accuracy of bi-level thresholding for the two methods.

Figure 14. Typical images in the CEDAR database of handwritten words

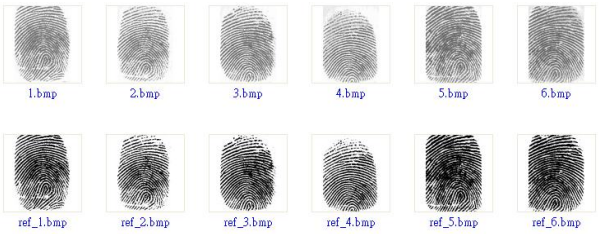


Figure 15. Typical images in the FVC2000 database of fingerprints

The index of ME is quite useful for quantifying the percentage of background pixels wrongly assigned to the foreground, and vice versa. For bi-level segmentation, ME can be simply represented as

$$ME = 1 - \frac{|B_O \cap B_T| + |F_O \cap F_T|}{|B_O| + |F_O|} \quad (9)$$

where B_O and F_O denote the background and foreground of the original (ground-truth) image, respectively, and B_T and F_T denote the background and foreground pixels in the test image, respectively. Note that the value of ME varies from 0 for a totally well classified image to 1 for a completely wrongly binarized image.

Table 1 shows the results of ME of bi-level thresholds for the two methods using the test images of handwritten words taken from CEDAR and fingerprints taken from FVC2000. As indicated in Table 1, the values of ME are very close for fingerprints images, but there are small differences for handwritten words images under the cases of no noise. The results of ME after adding Gaussian noise with standard deviations of and were also examined. Approximate ME values were obtained by the two methods although there was some noise in the test images.

The relative error (RE) of HGEM to Otsu's method, defined in Eq. (10), can be used to measure the closeness of the threshold values obtained by the two methods, where (1-ME) means the percentage of the correct classification of image pixels. As indicated in Table 1, the maximum RE with cases of no noise is 1.494% for handwritten words images. However, when Gaussian noise was added to the test images, the maximum RE is only 1.104% in the CEDAR database. Consequently, the relative errors of HGEM to Otsu's method in all cases are less than 1.50%.

$$RE = \frac{|(1 - ME_{Otsu}) - (1 - ME_{TSMO})|}{1 - ME_{Otsu}} = \frac{|ME_{Otsu} - ME_{TSMO}|}{1 - ME_{Otsu}} \quad (10)$$

Table 1. Comparisons of ME and RE with cases of no noise, σ=10, and σ=20

Methods	Handwritten words			Fingerprints		
	No noise	σ=10	σ=20	No noise	σ=10	σ=20
ME(Otsu)	0.0162	0.0079	0.0221	0.0298	0.0225	0.0442
ME(HGEM)	0.0309	0.0163	0.0113	0.0295	0.0315	0.0444
RE(%)	1.494	0.847	1.104	0.031	0.921	0.021

3.3 Comparison of HGEM and Otsu's method

The corresponding FPGA circuits for the proposed HGEM method were designed based on the architecture of the threshold estimation unit shown in Fig. 16. This unit comprises the following three modules: histogram statistics, threshold searching, and threshold determination. The histogram statistics module is used to divide the number of gray levels into 16 counters (C0 to C15) as described earlier; it then outputs the resulting histogram to the threshold searching module. As indicated in Fig. 16, the threshold searching module consists of two sub-modules, namely, threshold search1 used to find max1_1 and max1_2, and threshold search2 used to find max2_1 and max2_2. Then, the threshold determination module determines th1 and th2 based on the proposed HGEM method (see Eqs. (5) – (8)). Finally, the threshold value can be evaluated as 16×(th1+th2)/2 in this module.

Figure 16. Architecture of the threshold estimation unit

Some issues, such as latency in the design of the FPGA, should be considered carefully. Theoretically, when a 256×256 gray-level image is used, the width of the counters must be 16 bits to avoid overflow. Therefore, 16-bit wide comparators are required to find the first two largest count values in the submodules: threshold search1 and search2. However, the latency of the synthesized circuit in the FPGA is very serious due to the larger number of bits, i.e., 16 bits, used in the comparators. To improve the latency of the threshold searching module, the number of bits was reduced to 10 in the comparators. By only comparing the results in higher bits of the counters and discarding lower bits with allowable losses in accuracy, the number of bits used in the comparator can be reduced from 16 to 10, which increases the performance of this system from 184.6 MHz to 193.9 MHz.

The total cycles required for threshold estimation in the FPGA includes the time consumed by the three modules mentioned above, where the time required by the histogram statistics module highly depends on the input image size; for example, 65,536 cycles are needed to complete the histogram when the input image is 256×256 pixels. Moreover, eight

cycles are needed for the threshold searching module to determine the index values corresponding to the first two largest count values for both regions; max1_1 and max1_2 for the left region, and max2_1 and max2_2 for the right region. Four cycles are required for the threshold determination module to determine th1 and th2, and to complete the calculation of the threshold value, 8×(th1+th2). In this study, the threshold searching and determination modules were designed using state machines. Consequently, the total time needed by the threshold estimation unit for a 256×256 image is 65,536+8+4=65,548 cycles.

4. Experimental results

The proposed system architecture consists of the moving window generator, the three-stage Sobel pipeline unit, and the threshold estimation unit. The components were integrated and implemented into a chip system with a Xilinx Virtex-4 (XC4VLX25) FPGA. The total execution time can be evaluated as the cycles consumed by the moving window generator (=65,536 cycles), the three-stage Sobel pipeline unit (=4 cycles), and the threshold estimation unit (=65,548 cycles), a total of 65,536+4+65,548=131,088 cycles. The synthesized results of this FPGA chip system including the UART data transmission unit reveal that the required gate count is only 17,101, and that the operation speed can reach up to 193.9 MHz, which is equivalent to the processing rate of 1,479 (=193.9MHz • 106/131,088) frame/s for a 256 • 256 image, as indicated in Tables 2 and 3.

Table 2. Comparison of performance of Sobel-based edge detector on FPGA

Architecture	Image size	Operation speed	Frame/s
Proposed system	256x256	193.9 MHz	1,479
K. Benkrid, 2002[17]	256x256	75 MHz	104
X. Li, 2003[19]	256x256	40 MHz	610
R.L. Rosas, 2005[18]	640x480	13.2 MHz	43

Table 3. List of synthesized resources for individual components on FPGA

Module	Slice flip flops	4 input LUTs	Occupied slices	BRAM/ FIFOs	Speed (MHz)	Gate Count
UART	55	121	64	0	249.6	1,203
Histogram statistic	257	49	146	0	324.9	5,846
Threshold search 1/2	77	194	116	0	266.6	1,816
Threshold segment	10	50	31	0	260.2	804
Moving window generator	140	189	131	2	237.9	2,382
Sobel pipeline unit	51	71	44	0	238.2	1,305
Total system	995 (4%)	1,116 (5%)	930 (8%)	38 (52%)	193.9 (%)	17,101 (%)
Virtex-4XC4VLX25	21,504	21,504	10,752	72	500	-

Table 2 compares the performance of the Sobel-based edge detectors implemented by [17],[18],[19] for some fixed image sizes. However, the edge detectors [17]-[19] were implemented on a pre-specified threshold value. The operational speed of the proposed architecture, i.e., 193.9 MHz, is about 2.5 to 4.8 times greater than those of [17],[19] for 256×256 images. The processing rate, i.e., 1,479 frame/s, far exceeds the requirement of a real time image processing system. This implies that highly efficient image processing can be achieved using the proposed architecture.

Table 3 lists the synthesized resources of the FPGA chip system for individual components. As indicated in Table 3, the histogram statistics module, implemented using only 16 counters with a width of 16 bits, consumes the most resources of the FPGA, with about 5,846 gates (about 34% of the total system), to complete a histogram of a gray-level image. That is why this work does not use 256 counters, corresponding to 256 gray levels, to implement this module. Furthermore, since the function of this module is relatively simple, the operational speed, i.e., 324.9 MHz, is the highest among the modules. Although all components can perform individually with an operation speed higher than 235 MHz, the total speed of the integrated system is only 193.9 MHz when all components have been interconnected. Moreover, to readily access the image data, 38 blocks of RAMs were used in the FPGA chip system, where two of the blocks were used by the moving window generator, and the others were employed for temporarily storing the image data.

Figure 17 shows the software used for interfacing a PC and the FPGA chip system, developed using Borland C++ Builder 6.0. This software can be used to verify the correctness of the edge detection obtained by the proposed architecture. As indicated in Fig. 18, the edge detection of the image "camera man" by the software (Fig. 18(b)) and FPGA (Fig. 18(c)) is the same, implying that a highly accurate threshold value can be obtained using the proposed architecture.

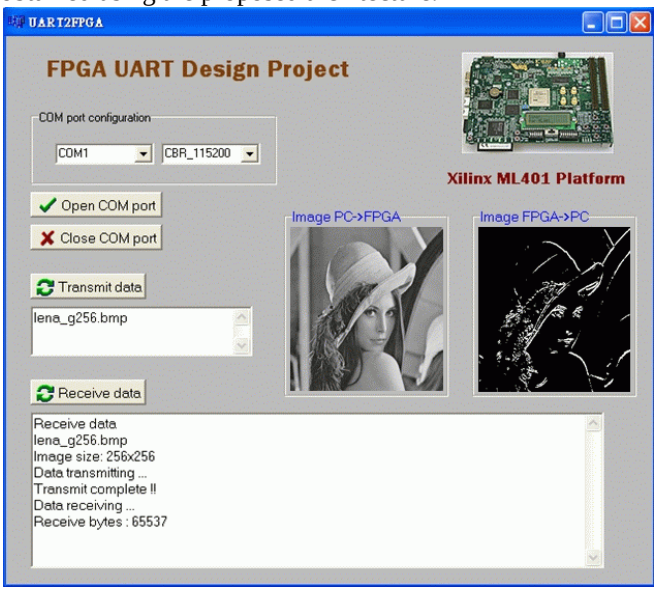


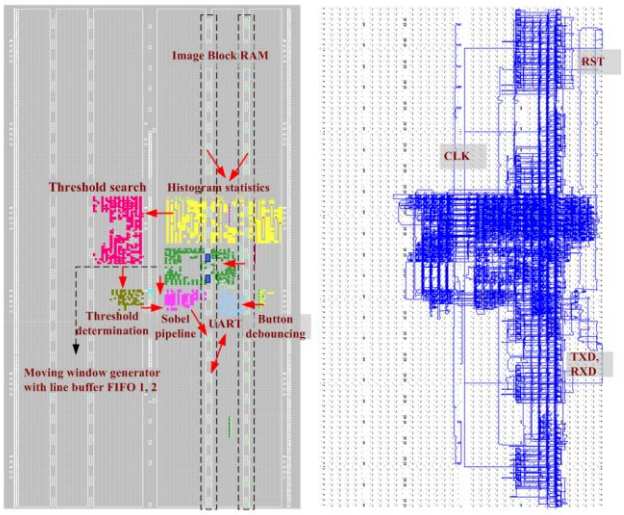
Figure 17. Developed software for threshold estimation by HGEM

(a). Original image **(b).** Software **(c).** FPGA

Figure 18. Comparison of edge detection by software and FPGA

Figure 19 shows the placement and routing for all the components in the FPGA after floorplanning. As indicated in

Fig. 19(a), the components in FPGA can be arranged compactly by specifying the positions of CLBs (Configurable Logic Blocks) to reduce the required areas. The block RAMs, indicated by two rectangles with dashed lines (see Fig. 19(a)), was used to store the input and output images temporarily. The histogram statistics module accesses the block RAMs to acquire the image data and then groups it into 16 counters to complete the histogram of an input gray-level image. The placement of this module should be as close as possible to the block RAMs to save routing resources. However, since the synthesized logic of this module is relatively large, its placement may occupy the positions of some block RAMs, making this memory space unavailable for other modules. To reduce the number of occupied blocks of RAMs, the height of the floorplanning area of this module was shortened by area constraints, and the width was adjusted to cross over three CLB columns so that the input and output images could be completely stored in neighboring two columns of the block RAMs, greatly reducing the routing paths between this module and the block RAMs. Additionally, because the line buffers of the moving window generator need two blocks of RAMs, the placement of the moving window generator also needs to cross over two CLB columns to reduce routing paths.



(a). Placement **(b).** Routing

Figure 19. Placement and routing in the FPGA after floorplanning

5. Conclusion

A fast and efficient algorithm called HGEM, which is based on the Gaussian distributions of a histogram, was developed to determine a threshold for a gray-level image of which value is close to that of Otsu's method. The proposed method is simple and efficient for implementation on an FPGA, since it avoids the repetitious iterations and complex arithmetic operations, such as multiplication and division, when compared to the basic Otsu thresholding procedures. To use hardware resource more effectively, the block RAMs was used to implement the line buffers (FIFO A and B) of the moving window generator. The synthesized results indicate that the operation speed can be increased by 15.6%; the logic elements of the FPGA were reduced by 74.2%, when using the block RAMs to replace the shift registers.

Misclassification error (ME) was used in the evaluations of the accuracy for the proposed method. The maximum ME for HGEM in all test cases with or without noise (σ=10 and σ=20) was only 0.044, which is very close to the value obtained by Otsu's method. The relative errors (REs) were all less than 1.50% for the test images, indicating that a comparable threshold can be obtained by HGEM when compared to Otsu's method. Therefore, the proposed method is very efficient with an accuracy equivalent to that of Otsu's method.

The hardware architecture of the Sobel-based edge detector with an optimal threshold determined by HGEM comprises four major components: the UART transmission unit, the moving window generator, the three-stage Sobel pipeline unit, and the threshold estimation unit. The components were integrated and implemented into a single chip system with a Xilinx Virtex-4 (XC4VLX25) FPGA. The synthesized results reveal that the total required gates amount to 17,101, and that the total operation speed can run at up to 193.9 MHz, which is equivalent to a theoretical processing rate of 1,479 frame/s for 256×256 images. This result confirms that the proposed architecture on FPGA can easily achieve the requirements for a real-time image processing system.

References

[1] S. Arseneau, J.R. Cooperstock, "Real-Time Image Segmentation for Action Recognition," in: Proc. IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, Victoria, B.C., Canada, pp. 86-89, 1999.

[2] Hammouche K, Diaf M, Siarry P, "A multilevel automatic thresholding method based on a genetic algorithm for a fast image segmentation," Comput. Vis. Image Und., Vol.109, No. 2, pp. 163-175, 2008.

[3] D.Y. Huang, C.H. Wang, "Optimal multi-level thresholding using a two-stage Otsu optimization approach," Pattern Recogn. Lett., Vol. 30, No. 3, pp. 275-284, 2009.

[4] D. Aiteanu, D. Ristic, A. Graser, "Content based threshold adaptation for image processing in industrial application," in: Int. Conf. Control and Automation, Budapest, Hungary, pp. 1022-1027, 2005.

[5] H.F. Ng, "Automatic thresholding for defect detection," Pattern Recogn. Lett., Vol. 27, No. 14, pp. 1644-1649, 2006.

[6] G. Jing, D. Rajan, C.E. Siong, "Motion Detection with Adaptive Background and Dynamic Thresholds," in: IEEE Int. Conf. Information, Communications and Signal Processing, Bangkok, Thailand, pp. 41-45, 2005.

[7] E.P. Ong, B.J. Tye, W. S. Lin, M. Etoh, "An efficient video object segmentation scheme," in: IEEE Proc. Int. Conf. Acoustics, Speech, and Signal Processing, Orlando, Florida, USA, pp. 3361-3364, 2002.

[8] C. Su, A. Amer, "A Real-Time Adaptive Thresholding for Video Change Detection," in: IEEE Int. Conf. on Image Processing(ICIP), Atlanta, Georgia, USA, pp. 157-160, 2006.

[9] A. Amer, "Memory-based spatio-temporal real-time object segmentation for video surveillance," in: Proc.

SPIE Int. Symposium on Electronic Imaging, Conf. on Real-Time Imaging VII, Santa Clara, CA, USA, pp. 10-21, 2003.
[10] S.Y. Chien, Y.W. Huang, B.Y. Hsieh, S.Y. Ma, L.G. Chen, "Fast video segmentation algorithm with shadow cancellation, global motion compensation, and adaptive threshold techniques," IEEE Trans. Multimedia, Vol. 6, No. 5, pp. 732-748, 2004.
[11] O. Sukmarg, K.R. Rao, "Fast object detection and segmentation in MPEG compressed domain," in: IEEE Proc. TENCON, Kuala Lumpur, Malaysia, pp. 364-368, 2000.
[12] D. Zhang, G. Lu, "Segmentation of Moving Objects in Image Sequence: A Review. Circuits Syst," Signal Process, Vol. 20, No. 2, pp. 143-183, 2001.
[13] M.S. Atkins, B.T. Mackiewich, "Fully Automatic Segmentation of the Brain in MRI," IEEE Trans. Med. Imaging, Vol. 17, No. 1, pp. 98-107, 1998.
[14] P.K. Saha, J.K. Udupa, "Optimum Image Thresholding via Class Uncertainty and Region Homogeneity," IEEE Trans. Pattern Anal. Mach. Intell., Vol. 23, No. 7, pp. 689-706, 2001.
[15] N. Otsu, "A threshold selection method from gray-level histogram," IEEE Trans. Syst. Man Cybern., Vol. 9, No. 1, pp. 62-66, 1979.
[16] H. Tian, S. K. Lam, T. Srikanthan, "Implementing Otsu's thresholding process approximation unit using area-time efficient logarithmic," in: Proc. Int. Symposium Circuits and Systems (ISCAS), Bangkok, Thailand, pp. IV-21-IV-24, 2003.
[17] K. Benkrid, D. Crookes, A. Benkrid, "Towards a general framework for FPGA based image processing using hardware skeletons," Journal of Parallel Computing, Vol. 28, No.7-8, pp. 1141-1154, 2002.
[18] R.L. Rosas, A. de Luca, F.B. Santillan, "SIMD architecture for image segmentation using Sobel operators implemented in FPGA technology," The 2nd International Conference on Electrical and Electronics Engineering (ICEEE) and XI Conference on Electrical Engineering, Mexico City, Mexico, pp. 77-80, 2005.
[19] Xue Li, Rongchun Zhao, Qing Wang, "FPGA based Sobel algorithm as vehicle edge detector in VCAS," in: Proc. IEEE International Conference on Neural Networks and Signal Processing, Nanjing, China, pp. 1139-1142, 2003.
[20] P.Y. Hsiao, C.H. Chen, H. Wen, S.J. Chen, "Real-time realisation of noise-immune gradient-based edge detector," IEEE Proceedings-Computers and Digital Techniques. Vol. 153, No. 4, pp. 261-269, 2006.
[21] M. Sezgin, B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," J. Electron. Imaging, Vol. 13, No. 1, pp. 146-165, 2004.
[22] A. Benedetti, A. Prati, N. Scarabottolo, "Image convolution on FPGAs: the implementation of a multi-FPGA FIFO structure," Proceedings of the 24th Euromicro Conference, Vasteras, Sweden, pp. 123-130, 1998.
[23] B. Bosi, G. Bois, Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., Vol. 7, No. 3, pp. 299-308, 1999.
[24] V. Muthukumar, D.V. Rao, "Image processing algorithms on reconfigurable architecture using HandelC," Proceedings of the Euromicro Symposium on Digital System Design, Rennes, France, pp. 218-226, 2004.
[25] C.K. Lee, F.W. Choy, H.C. Lam, "Real-time thresholding using histogram concavity," Proceedings of the IEEE International Symposium on Industrial Electronics, Xian, China, pp. 500-503, 1992.
[26] S.E. El-Khamy, M. Lotfy, N. El-Yamany, "A modified Fuzzy Sobel edge detector," 7th National Radio Science Conference, Minufiya University, Egypt, pp. C32 1-9, 2000.
[27] Jianping Fan, Walid G. Aref, Mohand-Said Hacid, Ahmed K. Elmagarmid, "An improved automatic isotropic color edge detection technique," Pattern Recogn. Lett., Vol. 22, No. 13, pp. 1419-1429, 2001.
[28] C.H. Li, P.K.S. Tam, "An iterative algorithm for minimum cross-entropy thresholding," Pattern Recogn. Lett., Vol. 19, No. 8, pp. 771-776, 1998.
[29] Shu Yang, Ying Han, Cai-Rong Wang, Xiao-Wei Wang, "Fast selecting threshold algorithm based on one-dimensional entropy," Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, China, pp. 4554-4557, 2005.
[30] R.C. Gonzalez and R.E. Woods, Digital image processing, 2nd edition, Prentice Hall, Upper Saddle River, New Jersey, 2002.
[31] URL: http://www.csse.monash.edu.au/hons/projects/2002/Laura.Frost/index.html
[32] N.S. Sargur, Center of Excellence for Document Analysis and Recognition (CEDAR), 1991. Available from:< http://www.cedar.buffalo.edu>
[33] D. Maltoni, D. Maio, A.K. Jain, S. Prabhakar, FVC2000 Fingerprint Database, 2000. Available from: http://bias.csr.unibo.it/fvc2000/default.asp

Author Biographies

Deng-Yuan Huang received his Ph.D. degree in Aeronautic and Astronautic Engineering from National Cheng Kung University, Tainan, Taiwan, in 1994. He was with the steel and alumina R&D department at CSC Inc. in Taiwan for several years as an associate scientist specializing in process control in steel-making. He joined the Department of Electrical Engineering at Dayeh University in 2002 and is currently an assistant professor. He has published over 40 papers in journals and conference proceedings since 2002. His major research interests include FPGA chip design, image processing, pattern recognition, and computer vision.

Da-Wei Lin received his M.S. degree in computer science and information engineering in 2009 from Dayeh University, Changhua, Taiwan. Currently, he is working toward the Ph.D. degree in electrical engineering at the same university. His current research interests include image processing and computer vision.

Wu-Chih Hu received his Ph.D. degree in electrical engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 1998. From 1998, he worked at the National Penghu University of Science and Technology for 12 years. He is currently an associate professor in the Department of Computer Science and Information Engineering. He has published more than 70 papers in journals and conference proceedings since 1998. His current research interests include computer vision, image processing, pattern recognition, digital watermarking, visual surveillance, and video processing.