IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 19, 2018, accepted August 1, 2018, date of publication August 3, 2018, date of current version August 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2863019

FPGA-Based Hardware Design for Scale-Invariant Feature Transform

SHIH-AN LI[1], WEI-YEN WANG[2], (Fellow, IEEE), WEI-ZHENG PAN[2], CHEN-CHIEN JAMES HSU[2], (Senior Member, IEEE), AND CHENG-KAI LU[3], (Senior Member, IEEE)

[1]Department of Electrical Engineering, Tamkang University, New Taipei City 25137, Taiwan
[2]Department of Electrical Engineering, National Taiwan Normal University, Taipei 10610, Taiwan
[3]Department of Electrical and Electronic Engineering, Universiti Teknologi PETRONAS, Bandar Seri Iskandar 32610, Malaysia

Corresponding author: Chen-Chien James Hsu (jhsu@ntnu.edu.tw)

This work was supported in part by the "Higher Education Sprout Project" of the National Taiwan Normal University, through the Ministry of Education, Taiwan, and in part by the Ministry of Science and Technology, Taiwan, under Grants MOST 107-2634-F-003-001 and MOST 107-2634-F-003-002.

ABSTRACT This paper proposes a novel hardware design method of scale-invariant feature transform (SIFT) algorithm for implementation on field-programmable gate array (FPGA). To reduce the computing costs, Gaussian kernels are calculated offline for use in Gaussian filters. To eliminate low-contrast points, the inverse of a Hessian matrix is required for hardware implementation, which results in poor performance because dividers are needed. To solve this problem, this paper presents a new mathematical derivation model to implement the low-contrast detection, avoiding the use of any dividers. For the implementation of the normalization module, a large number of dividers are required by traditional methods, which adversely affects the computational efficiency. This paper presents a new architecture using only one divider to implement the normalization function in hardware. Thanks to the parallel processing architecture proposed to design the image pyramid, SIFT detection, and SIFT descriptor, the computational efficiency of the SIFT algorithm is significantly improved. As a result of the proposed design method, the requirement of logic elements in the FPGA hardware is greatly reduced and system frequency is significantly increased. Experimental results show that the proposed hardware architecture outperforms existing techniques in terms of resource usage and computational efficiency for real-time image processing.

INDEX TERMS Scale-invariant feature transform, field-programmable gate array, parallel processing architecture.

I. INTRODUCTION

Feature extraction is a fundamental aspect of many problems in the field of machine vision, including image stitching, object recognition, 3D modeling from multiple images, image tracking, and robotic mapping and navigation. Although many feature-detection algorithms have been proposed over the past years, the scale-invariant feature transform (SIFT) [1] algorithm, which mainly converts image data into high-dimensional feature descriptors before matching two images, is the most stable method of all. Because of its stability and robustness, SIFT not only deals with change in brightness but also satisfactorily addresses the problem of image scaling and rotation. As a result, SIFT is able to maintain feature invariance under orientation or illumination changes, out-performing the commonly used Harris corner detector [2]. Although SIFT is powerful with excellent results, it suffers from heavy computation and large memory usage, which impose a serious constraint upon real-world applications. In order to overcome this drawback, several approaches, including Speed Up Robust Features (SURF) [3] and Principal Components Analysis (PCA) SIFT [4], have been proposed with the aim of reducing calculation time. As a stable image recognition and descriptor algorithm, SURF is widely applied to object-recognition problems. The biggest difference between SURF and SIFT is that the former uses integral images and 2D discrete wavelet transform; the latter uses an image pyramid and histogram of oriented gradients. PCA-SIFT, on the other hand, reduces the dimensionality of SIFT feature descriptors from 128 to 36 in order to minimize the dimension to speed up feature matching. In 2009, Juan and Gwun analyzed the performance of SIFT, SURF, and PCA-SIFT [5]. The result showed that SURF has the

shortest computation time while SIFT has the best matching accuracy, considering the factors of changing scale, image rotation, image blurring, and changing illumination. In particular, various variants of SIFT algorithms, for example affine SIFT [6], have been proposed in recent years with improved stability and robustness for use in feature matching. As a result, SIFT algorithms are widely used in various applications. Although deep learning approaches [7] have been adopted over the past years to extract region features with success, they inevitably require a lot of training examples for the model to learn properly. As complexities and uncertainties in the environment increase, feature extraction might not be successful if features in the scene are not well trained. Furthermore, network complexities of deep learning architecture also result in high costs particularly for hardware implementation [8], [9]. In view of the above, the SIFT algorithm would be the best choice of all if the problem of computation time can be overcome. To address this problem, much research has been conducted in recent years to implement the SIFT algorithm on various experimental platforms, including multi-core processors, graphics processing unit (GPU), and field-programmable gate array (FPGA), with a goal of speeding up the computational performance to achieve real-time image recognition. Among them, Bonate *et al.* presented a software and hardware co-design approach to implement SIFT on FPGA [10]. In addition, Yao *et al.* [11] presented a method of improving the SIFT algorithm for implementation on FPGA, where the dimensionality of SIFT feature descriptors is reduced from 128 to 72. As a result, its computation through hardware implementation of FPGA is nearly real-time. In 2014, Wang *et al.* [12] presented an embedded SOC architecture for detection and matching by hardware. The result showed that it can process 60 images per second. Jiang *et al.* [13] also presented a hardware structure to implement the detection and matching of SIFT by hardware. The experimental results demonstrated that about 150 images can be processed per second. All the above discussions suggested that SIFT can be successfully implemented on hardware platforms, not only maintaining the success rate of matching but also achieving the objective of real-time computation.

As an attempt to further improve the computational efficiency of hardware implementation on FPGA, we present a novel hardware design method for SIFT to accelerate the computational efficiency of its major modules, including image pyramid, SIFT detection, and SIFT descriptor. Because the hardware design is based on pipeline architecture, the execution speed is significantly improved. It is worth mentioning that Gaussian smoothing of an image requires an exponential function that it is hard to implement and requires a lot of logic elements in the hardware. With the use of offline calculation of the Gaussian kernel proposed in this paper, the number of logic elements required in the hardware can be greatly reduced to accelerate system frequency. To eliminate low-contrast points, inverse matrix operations are required in the hardware by traditional methods, which results in low performance because dividers are needed for the calculation. To solve this problem, this paper presents a new mathematical derivation model to implement low-contrast detection, avoiding the use of any dividers. Similarly, for the implementation of the normalization module, a large number of dividers are required by traditional methods. This paper presents a new architecture using only one divider, instead of 128, to implement the normalization function in hardware. As a result, computational efficiency is greatly improved. Thanks to the parallel processing architecture proposed in this paper to design the image pyramid, SIFT detection, and SIFT descriptor, computational efficiency of the entire hardware system is significantly improved. As a result of the proposed design method, utilization of logic elements required in the FPGA hardware is greatly reduced and system frequency is significantly increased.

The paper is organized as follows. A comprehensive review of hardware implementation methodologies of SIFT and their experimental results is presented in Section 2. Preliminaries of the SIFT algorithm are introduced in Section 3. The proposed hardware design methodology for SIFT is described in Section 4. Experimental results are presented in Section 5. Section 6 concludes this paper.

II. RELATED WORK

This section discusses existing techniques that implement SIFT algorithm and their experimental results.

Vourvoulankis *et al.* [14] proposed an FPGA-based pipeline architecture for implementing SIFT. At a resolution of 640 × 480, up to 70 fps can be processed, allowing the system to achieve real-time operation. Aniruddha *et al.* [15] also obtained a speed of around 55 fps for 640 × 480 images by implementing a parallel structure of SIFT on a GPU. Unfortunately, their works can only be used in situations in which the descriptor does not rotate or the system has a small change in angle. Yao *et al.* [11] proposed an optimized SIFT feature detection architecture for image matching. The number of descriptor vectors is reduced from 128 to 72, simplifying the matching operation. According to the experimental results, this approach successfully reduced the detection time to 31 ms, at the expense of very high resource utilization. Mizuno *et al.* [16], [17] proposed a SIFT implementation based on a FPGA architecture to quickly separate the regions of interest. This approach significantly reduced the required on-chip memory resources and supported two modes of operation. In the high-speed mode, a processing speed of 56 fps was achieved, while the high-precision mode could reach 32 fps; both of these results were achieved in VGA resolution. The main drawbacks of this architecture is the use of external Static Random-Access Memory (SRAM) to store input images and a large number of required Digital signal processing (DSP) blocks. Chang *et al.* [18] proposed a SIFT detection method that can reduce the usage of chip resources, where the SIFT keypoint detection time can be reduced to 11 ms. Unfortunately, it is applicable only to VGA resolution. Zhong *et al.* [19] used a combination of FPGA

and a DSP processor to implement SIFT, where the SIFT detector is implemented on FPGA, while the SIFT descriptor is implemented by DSP. This system takes 10 ms to detect features in an image, and each descriptor consumes 80 μs. This hybrid architecture, however, is limited to low-resolution input images and the calculation of the descriptor is too slow. Huang *et al.* [20] proposed an ASIC-based SIFT architecture, where parallel operations were used to transfer the input data to SRAM, which can reduce the number of transistors and chip area. According to the experimental results, they succeeded in detecting the feature points in VGA resolution in 3.4 ms. With the use of the finite-state machine (FSM), the calculation time for each descriptor is about 0.0331 ms. Suzuki and Ikenaga [21], [22] proposed a scheme in which the difference of Gaussian (DOG) is replaced by a Harris corner detector. The detected keypoints were used as a descriptor for SIFT. However, they failed to implement an image pyramid and therefore the difference in scale-space has not been considered in SIFT detection, which inevitably decreased the image matching performance. Kim and Lee [23] proposed a framework to maintain resource utilization at a lower level. They mainly split the input image and used an external SRAM to store the data to reduce the on-chip memory requirements. The calculation time of each descriptor is about 60 μs, which is the main bottleneck of the architecture. Chiu *et al.* [24] proposed a parallel layer of SIFT architecture. They used an integral image to reduce the computational complexity in Gaussian blurring. The calculation of descriptors, such as trigonometric functions and segmentation, used a custom multi-cycle component implemented on ASIC chips. When the number of feature points is less than 2000, it can handle 30-fps full-HD images. This means that it can compute the feature descriptor within 16 μs. Wang *et al.* [12] proposed an architecture that combines a detector, a descriptor, and a feature matcher, where SIFT is used for keypoint detection. The proposed system used an image with a resolution of 1280 $\times$ 720 pixels to achieve feature detection and matching at 60 fps. However, they sacrificed some of the robustness of the algorithm in order to speed up the execution. Jiang *et al.* [13] proposed an architecture for real-time SIFT extraction based on parallel and pipeline structure. They also used two ping-pong RAM buffers to preserve the location, gradient, and orientation of the features. The authors claimed that the proposed scheme has almost the same matching performance as the original algorithm. However, descriptor calculation is still the bottleneck of the entire algorithm.

III. PRELIMINARIES OF SIFT ALGORITHM

As a method of detecting and describing local features in images, the SIFT algorithm was proposed by David G. Lowe in 2004 [1]. Fig. 1 shows a flow chart of the SIFT algorithm, where three basic steps, namely Image Pyramid, SIFT Detection, and SIFT Descriptor, are required to implement the SIFT algorithm. Detailed descriptions for each function are given as follows:

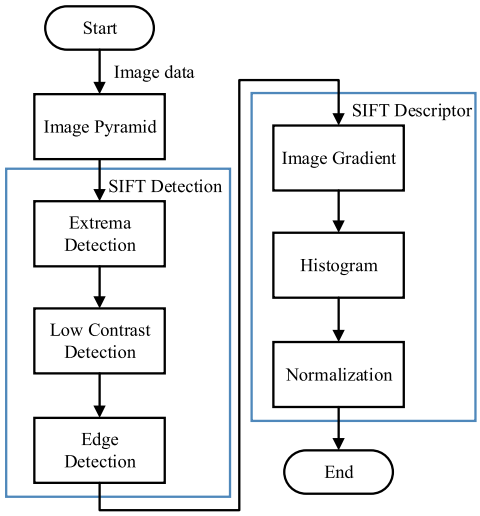


FIGURE 1. Flow chart of SIFT algorithm.

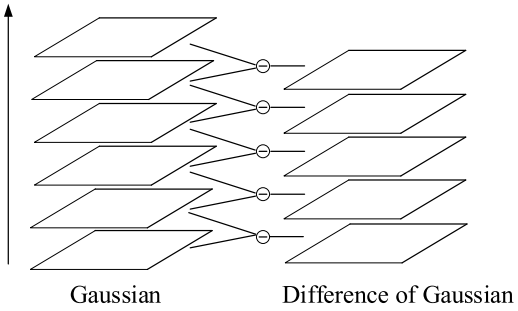


FIGURE 2. Architecture of Image Pyramid.

A. IMAGE PYRAMID

The image pyramid uses a cascade Gaussian filtering approach that creates continuous images at different scales. As shown in Fig. 2, six Gaussian images and five DOG (difference of Gaussian) images in every octave are generated to construct an image pyramid. First of all, each Gaussian filter uses a different sigma to calculate a Gaussian kernel in (1). Then a convolution operation is performed for the initial image and the Gaussian kernel to obtain a Gaussian image in (2). By subtracting the two continuous Gaussian images, a DOG image is obtained by (3).

$$K(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{1}$$

$$G(x, y, \sigma) = I(x, y) * K(x, y, \sigma) \tag{2}$$

$$D(x, y, \sigma) = G(x, y, k\sigma) - G(x, y, \sigma) \tag{3}$$

B. SIFT DETECTION

As soon as the image pyramid is built, we proceed to the step of keypoint detection, including extrema detection, eliminating low-contrast, and edge response keypoints. Extrema detection needs to use 3D DOG space, that is, three continuous DOG images of the same octave. As shown in Fig. 3, the red point is a target point and the blue points are the

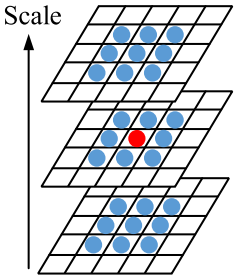


FIGURE 3. Extrema detection in SIFT Detection.

nearest neighbors of 3D DOG space. If the value of the red point is the maximum or minimum in the nearest neighbor region, it is a candidate point.

By doing so, candidate points can be determined in the previous step. However, some of them are unstable points. As a result, low-contrast keypoints and edge response points have to be eliminated in the next stage. First, the low-contrast detection uses a Taylor series to expand the 3D DOG space as shown in (4). Then we calculate the derivative of (4) to find the extrema value by making the derivative of (4) equal to zero. By proper derivations, we suppose the location of the extrema value according to (5). Substituting a of Equation (5) into x in Equation (4), we obtain (6). If $|D(a)|$ is smaller than 0.03, this candidate point has to be deleted.

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{4}$$

$$a = -\frac{\partial^2 D}{\partial x^2}^{-1}\frac{\partial D}{\partial x} \tag{5}$$

$$D(a) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}a \tag{6}$$

To eliminate the edge response, we first build a Hessian matrix in (7) and then follow the steps to calculate the trace (Tr(.)) and determinant (Det(.)) of the Hessian matrix by (8) and (9), respectively. Let r be the ratio between the biggest eigenvalue and the smallest one, so that $\alpha = r\beta$. We can use the solutions of (8) and (9) to derive (10). To detect an edge point, we only need to check (11), where r is usually 10. We can distinguish a non-edge point if the inequality in (11) is satisfied.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{7}$$

$$\mathrm{Tr}(H) = D_{xx} + D_{yy} = \alpha + \beta \tag{8}$$

$$\mathrm{Det}(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta \tag{9}$$

$$\frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{10}$$

$$\frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)} < \frac{(r+1)^2}{r} \tag{11}$$

Each keypoint has an orientation assignment to allow for rotation invariant. To calculate the correct orientation, we use the area adjacent to the gradient value of the direction of the statistics. The magnitude and direction calculations for the gradient are derived for every pixel in a neighboring region around the keypoint in the Gaussian-blurred image G. We count the magnitude of each direction in the entire Gaussian image and use the maximum magnitude of the direction as the direction of the keypoint, as shown by the red dot in Fig. 4. Equation (12) is used to calculate the gradient magnitude m around the red dot. Equation (13) is used to calculate the gradient direction $\theta(x, y)$. Each sample in the neighboring window added to a histogram bin is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a value of σ that is 1.5 times that of the scale of the keypoint. The peaks in this histogram correspond to dominant orientations, as shown in Fig. 4. In order to increase the success rate of matching, the second largest peak (about 15% of the feature points) has been chosen as the secondary direction. Thus, matching stability can be notably improved.

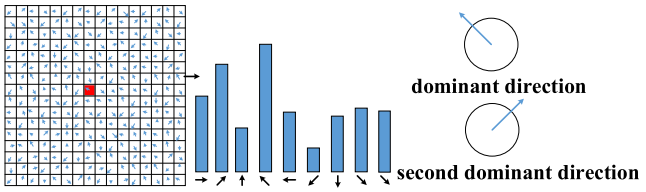


FIGURE 4. The statistics of the histogram of the feature gradient direction.

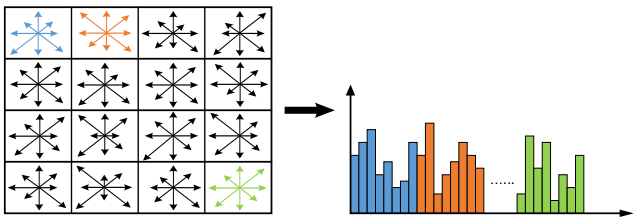


FIGURE 5. The graph of the result of feature descriptors.

$$m(x, y) = \sqrt{(G(x+1, y) - G(x-1, y))^2 + (G(x, y+1) - G(x, y-1))^2} \tag{12}$$

$$\theta(x, y) = \arctan\left(\frac{G(x, y+1) - G(x, y-1)}{G(x+1, y) - G(x-1, y)}\right) \tag{13}$$

C. SIFT DESCRIPTOR

In this paper, a gradient histogram is used to increase the matching success rate of feature descriptors. Before using the gradient histogram, we have to use (14) to rotate the main direction of the feature point mask to 0 degree. The 16 × 16 mask is divided into 4 × 4 regions, and the gradient of the eight directions in each region is counted, as shown in Fig. 5. Each region has a gradient value of eight directions, so each feature descriptor can be described as a 128-dimensions vector.

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \tag{14}$$

IV. PROPOSED METHODOLOGY

This section describes the proposed hardware design of SIFT. Major functional blocks of the SIFT hardware circuit

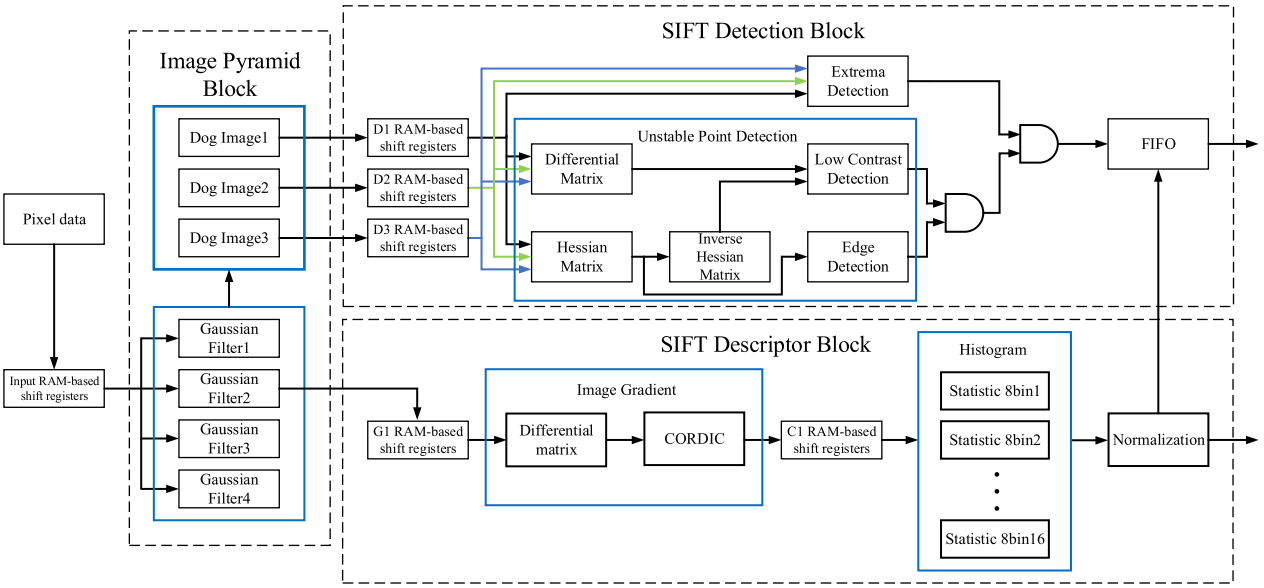


FIGURE 6. Block diagram of the proposed hardware design of SIFT.

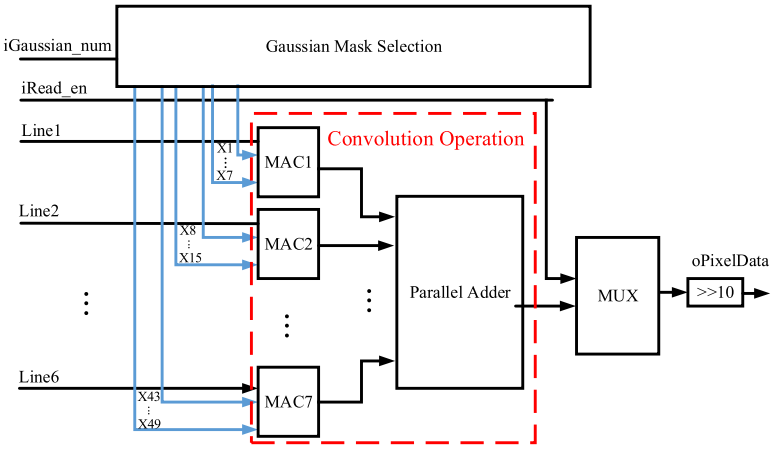


FIGURE 7. Gaussian Filter Module.

include Image Pyramid, SIFT Detection, and SIFT Descriptor, as shown in Fig. 6. The Image Pyramid Block uses parallel processing to calculate Gaussian filter images and DOG images. After the DOG images have been processed using hardware, the feature points are taken and stored in the first-in-first-out (FIFO). The feature points need to wait for the SIFT descriptor block to complete. The SIFT Descriptor then calculates the feature descriptor vector for each pixel. If the FIFO read value is 1, then the position and the descriptor vector of the feature point are stored. We use the pipeline architecture for the overall system to improve computational efficiency. Detailed descriptions for each hardware functional modules are given as follows.

A. IMAGE PYRAMID BLOCK

This paper designs a Gaussian filter module based on a parallel architecture to produce a Gaussian image, as shown in Fig. 7, where MAC# is a multiply and accumulate module. The Gaussian pyramid in this paper contains four consecutive Gaussian images, so four kinds of Gaussian mask parameters are used and stored in Gaussian Mask Selection, where iGaussian_num is used to select the corresponding Gaussian mask. To simplify the operations, we offline calculate four kinds Gaussian mask values and enlarge 1024 times of their values by shifting 10 bits left. After the operation is complete, the output value is then shifted right by 10 bits.

B. SIFT DETECTION BLOCK

Fig. 8 shows the pipeline hardware of the SIFT detection block. There are three main modules in this block, including 1) extrema detection, 2) unstable point detection, and 3) the pipeline hold circuit module. The following paragraphs describe the hardware design and implementation for each module.

1) EXTREMA DETECTION MODULE

The extrema detection module is used to determine whether the pixel is a maximum or a minimum of all neighboring

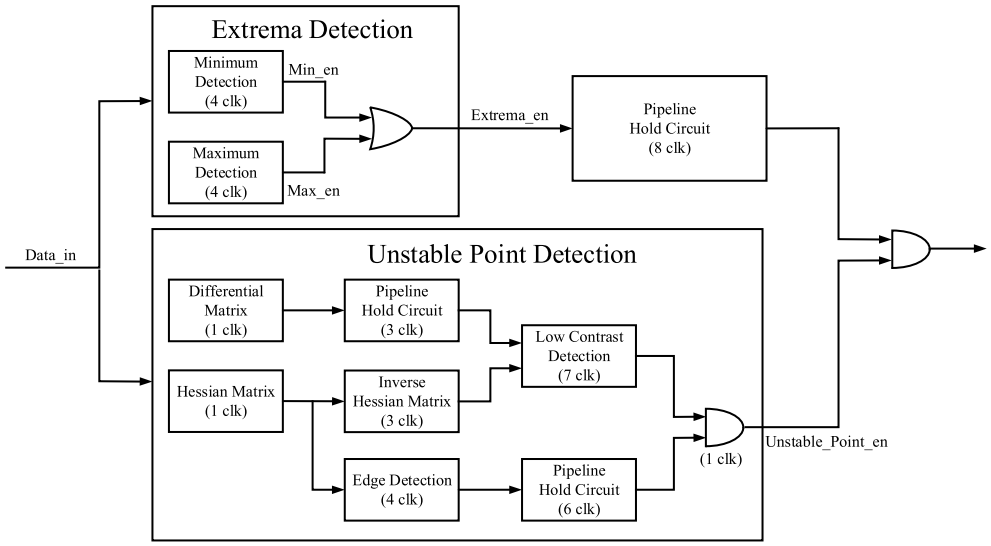


FIGURE 8. SIFT Detection Block.

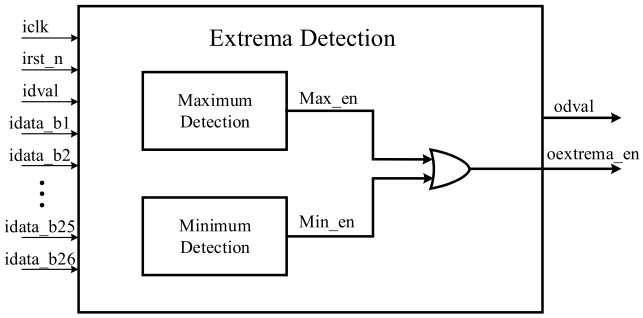


FIGURE 9. Extrema Detection Module.

points. This module performs both maximum detection and minimum detection with 26 neighboring points in parallel, as shown in Fig. 9.

2) HESSIAN MATRIX MODULE

Equation (15), as shown at the top of the next page, is a Hessian matrix and each variable in the matrix is described in (16) to (21), as shown at the top of the next page, where *Dxy*, *Dxs*, and *Dys* are divided by four by shifting 2 bits right. Fig. 10 illustrates the hardware realization of the Hessian matrix module.

3) DIFFERENTIAL MATRIX MODULE

Equation (22) is the partial differential matrix of the DOG images, and Fig. 11 shows the hardware realization of the Differential Matrix Module.

$$Diff\_m = \begin{bmatrix} D_x \\ D_y \\ D_s \end{bmatrix} = \begin{bmatrix} \dfrac{f_s(x+1,y) - f_s(x-1,y)}{2} \\ \dfrac{f_s(x,y+1) - f_s(x,y-1)}{2} \\ \dfrac{f_{s+1}(x,y) - f_{s-1}(x,y)}{2} \end{bmatrix} \quad (22)$$

4) INVERSE HESSIAN MATRIX MODULE

To obtain the inverse of the Hessian matrix, we use (23) to calculate the adjoint matrix. The determinant of the matrix is

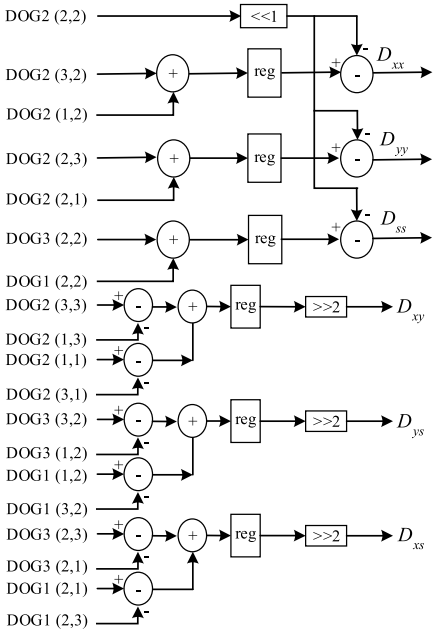


FIGURE 10. Hessian Matrix Module.

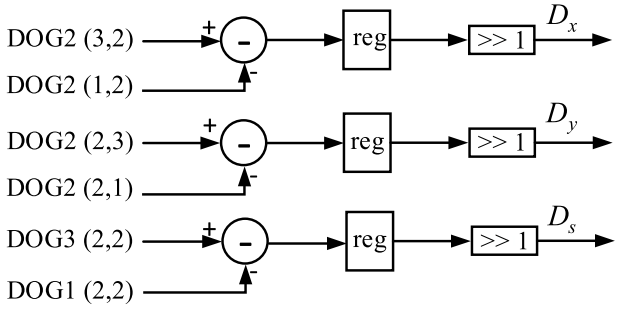


FIGURE 11. Differential Matrix Module.

calculated based on (24), where d_1 to d_6 can be calculated in parallel. Since calculating the inverse matrix using the divider in the hardware design significantly slows down the system

$$H = \begin{bmatrix} D_{xx} & D_{xy} & D_{xs} \\ D_{xy} & D_{yy} & D_{ys} \\ D_{xs} & D_{ys} & D_{ss} \end{bmatrix} \tag{15}$$

$$D_{xx} = f_s(x+1,y) + f_s(x-1,y) - 2f_s(x,y) \tag{16}$$

$$D_{yy} = f_s(x,y+1) + f_s(x,y-1) - 2f_s(x,y) \tag{17}$$

$$D_{ss} = f_{s+1}(x,y) + f_{s-1}(x,y) - 2f_s(x,y) \tag{18}$$

$$D_{xy} = \frac{f_s(x+1,y+1) - f_s(x+1,y-1) + f_s(x-1,y-1) - f_s(x-1,y+1)}{4} \tag{19}$$

$$D_{ys} = \frac{f_{s+1}(x+1,y) - f_{s+1}(x-1,y) + f_{s-1}(x-1,y) - f_{s-1}(x+1,y)}{4} \tag{20}$$

$$D_{xs} = \frac{f_{s+1}(x,y+1) - f_{s+1}(x,y-1) + f_{s-1}(x,y-1) - f_{s-1}(x,y+1)}{4} \tag{21}$$

performance, we calculate and output the adjoint matrix and the determinant to the next module.

$$Inv\_adj(H) = \begin{bmatrix} +\begin{vmatrix} h_{22} & h_{23} \\ h_{32} & h_{33} \end{vmatrix} & -\begin{vmatrix} h_{12} & h_{13} \\ h_{32} & h_{33} \end{vmatrix} & +\begin{vmatrix} h_{12} & h_{13} \\ h_{22} & h_{23} \end{vmatrix} \\ -\begin{vmatrix} h_{21} & h_{23} \\ h_{31} & h_{33} \end{vmatrix} & +\begin{vmatrix} h_{11} & h_{13} \\ h_{31} & h_{33} \end{vmatrix} & -\begin{vmatrix} h_{11} & h_{13} \\ h_{21} & h_{23} \end{vmatrix} \\ +\begin{vmatrix} h_{21} & h_{22} \\ h_{31} & h_{32} \end{vmatrix} & -\begin{vmatrix} h_{11} & h_{12} \\ h_{31} & h_{32} \end{vmatrix} & +\begin{vmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{vmatrix} \end{bmatrix} \tag{23}$$

$$Inv\_\det(H) = (d_1 - d_4) + (d_2 - d_5) + (d_3 - d_6)$$

$$\begin{cases} d_1 = h_{11} \cdot h_{22} \cdot h_{33} \\ d_2 = h_{13} \cdot h_{21} \cdot h_{32} \\ d_3 = h_{12} \cdot h_{23} \cdot h_{31} \\ d_4 = -h_{13} \cdot h_{22} \cdot h_{31} \\ d_5 = -h_{11} \cdot h_{23} \cdot h_{32} \\ d_6 = -h_{12} \cdot h_{21} \cdot h_{33} \end{cases} \tag{24}$$

5) LOW CONTRAST DETECTION MODULE

Equations (25) and (26) are the original formula to determine the low contrast keypoints, where the inverse of matrix is required. Taking square of (25), we obtain (27). To compute the inverse matrix, it is necessary to use the divider, which inevitably consumes a lot of hardware resources and reduces the processing speed. To avoid this problem, we firstly output $adj(A)$ and $\det(A)$ of the Inverse Hessian Matrix and then we express the inverse matrix in (26) using the adjoint matrix and determinant to obtain Eq. (28).

$$|D(x)| = \left| D(0) + \frac{1}{2}\frac{\partial D^T(0)}{\partial X}x \right| \le 0.03 \tag{25}$$

$$x = -\left(\frac{\partial^2 D(0)}{\partial X^2}\right)^{-1}\frac{\partial D(0)}{\partial X} \tag{26}$$

$$D(x)^2 = D(0)^2 + D(0)\frac{\partial D^T(0)}{\partial X}x + \frac{1}{4}\left(\frac{\partial D^T(0)}{\partial X}x\right)^2 \le 0.0009 \tag{27}$$

$$x = -\frac{adj(\frac{\partial^2 D(0)}{\partial X^2})}{\det(\frac{\partial^2 D(0)}{\partial X^2})}\frac{\partial D(0)}{\partial X} \tag{28}$$

Substituting Equation (28) into (27), we obtain an approximate inequality (29). Finally, we can easily implement (29) by hardware to determine the success or failure of low-contrast detection. The hardware module is shown in Fig. 12.

$$1024 \times (a - b + c) \le \det(\frac{\partial^2 D(0)}{\partial X^2})^2 \tag{29}$$

$$a = D(0)^2 \det(\frac{\partial^2 D(0)}{\partial X^2})^2 \tag{30}$$

$$b = D(0)\frac{\partial D^T(0)}{\partial X}adj(\frac{\partial^2 D(0)}{\partial X^2})\frac{\partial D(0)}{\partial X}\det(\frac{\partial^2 D(0)}{\partial X^2}) \tag{31}$$

$$c = \frac{1}{4}\left(\frac{\partial D^T(0)}{\partial X}adj(\frac{\partial^2 D(0)}{\partial X^2})\frac{\partial D(0)}{\partial X}\right)^2 \tag{32}$$

6) EDGE-DETECTION MODULE

The function of this module is to determine whether a feature point is an edge response keypoint. We use (33) and (34) to calculate the trace and determinant of the Hessian matrix, so that (35) can be used to decide whether it is an edge feature point, where the pixel is not an edge feature point if (35) is true. Fig. 13 shows the hardware diagram of the Edge Detection Module, where r is equal to 10.

$$\mathrm{tr}(\mathrm{H}) = D_{xx} + D_{yy} = \lambda_1 + \lambda_2 \tag{33}$$

$$\det(\mathrm{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \lambda_1\lambda_2 \tag{34}$$

$$\mathrm{tr}(\mathrm{H})^2 r < (r+1)^2 \det(\mathrm{H}) \tag{35}$$

C. SIFT DESCRIPTOR BLOCK

This paragraph introduces the hardware architecture of the SIFT feature descriptor as shown in Fig. 14, including three functional modules: (1) Image Gradient, (2) Histogram, and (3) Normalization.

1) IMAGE GRADIENT MODULE

To calculate the gradient and direction, we need the square root and $\tan^{-1}$ functions. Therefore, we design a CORDIC

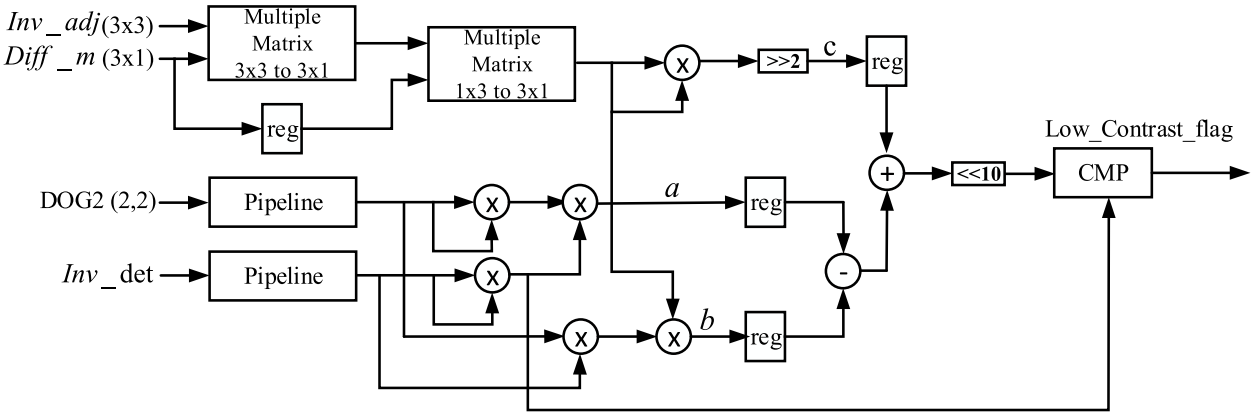


FIGURE 12. Low-Contrast Detection Module.

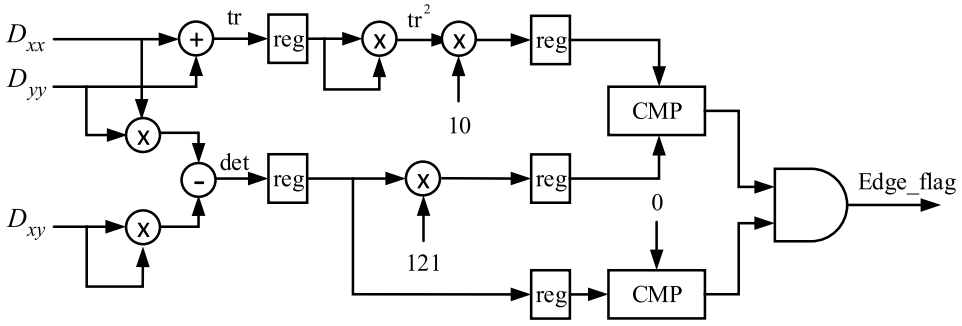


FIGURE 13. Edge Detection Module.

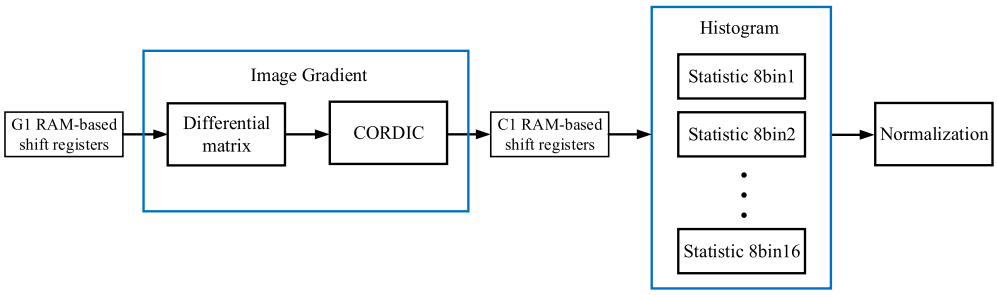


FIGURE 14. SIFT Descriptor Block.

(Coordinate Rotation Digital Computer) circuit to deal with them, as shown in Fig. 15, where K is a constant value

$$(K = \prod_{i=0}^{n} \cos\left(\tan^{-1}\left(\frac{1}{2^n}\right)\right) \approx 0.607253).$$

2) HISTOGRAM MODULE

For feature point matching, we use the histogram of oriented gradient (HOG) to calculate the dominant direction and descriptor. To implement the gradient statistical histogram module, we divide the 16×16 mask size into 16 sub-regions and calculate the gradient histogram in the each sub-region, as shown in Fig. 16. Sixteen statistic eight-bin modules are therefore to simultaneously compute the gradient in eight directions, as shown in Fig 17. Each Statistic_bin module determines which sub-region the angle belongs to and accumulates the gradient value, as shown in Fig. 18.

3) NORMALIZATION MODULE

The normalization operations are performed in (36) and (37), where W is the feature point vector, and $W = (w_1, w_2, \ldots, w_{128})$. L is a normalization vector and $L = (l_1, l_2, \ldots, l_{128})$.

$$s = \|W\| \tag{36}$$

$$L = \frac{W}{s} \tag{37}$$

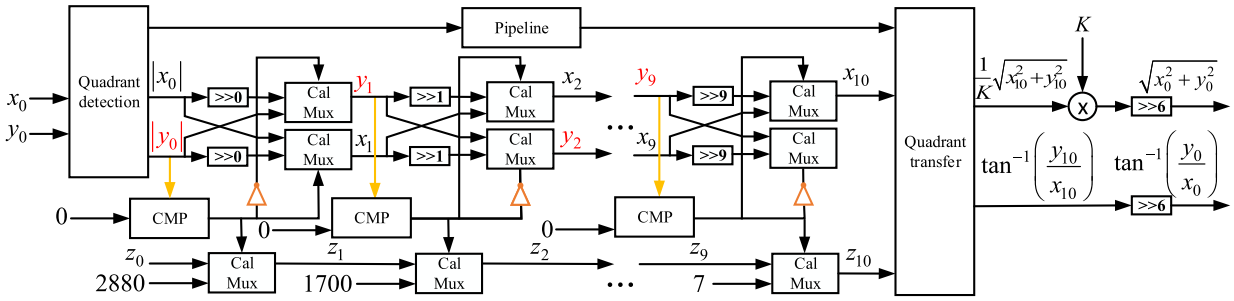


FIGURE 15. CORDIC Module of Image Gradient.

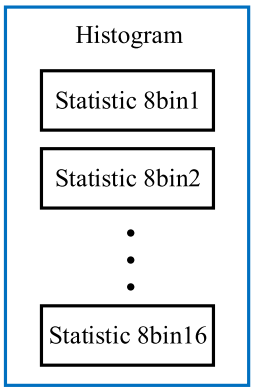


FIGURE 16. Histogram Module.

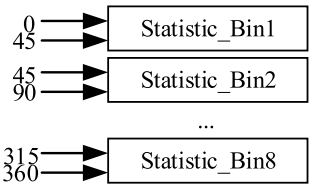


FIGURE 17. Statistic 8-bin# Module.

For the implementation of the hardware normalization, a large number of multipliers and dividers are required. However, the use of a large number of dividers undermines system performance and demands a large number of logic elements. Thus, we rewrite (39) as (40), where we can only use one divider in addition to multipliers and shifters to implement the normalization hardware. If s is 270, the value of n ($n = 8$ in this case) can be found according to (38) and m value can be calculated from (41) afterwards. The architecture of the normalization hardware module is shown in Fig. 19.

$$2^n \leq s < 2^{n+1} \tag{38}$$

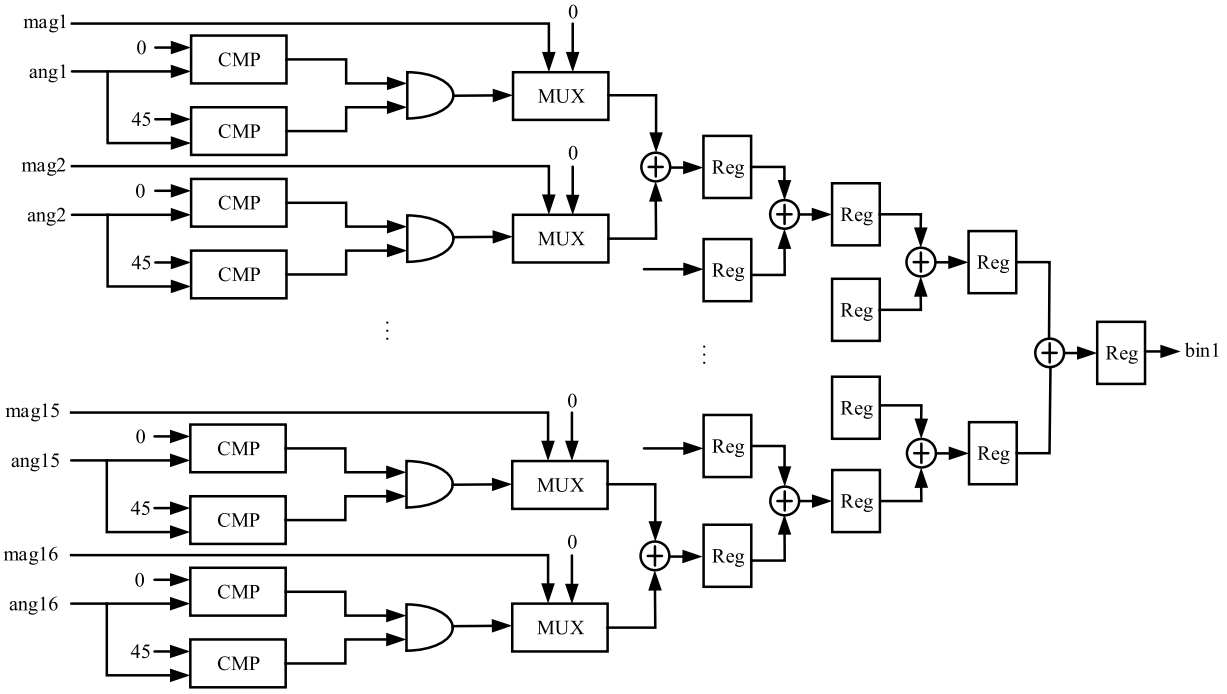


FIGURE 18. Statistic bin # Module.

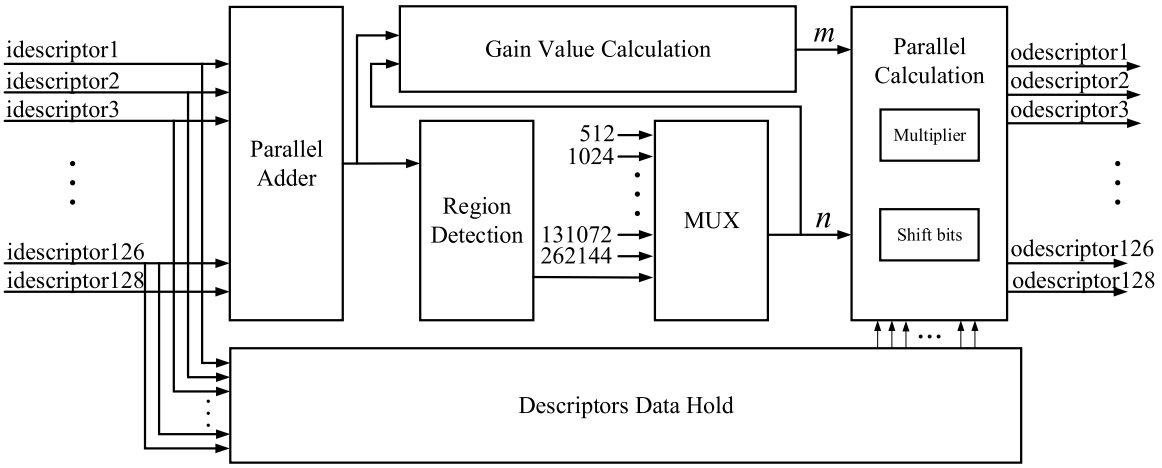


FIGURE 19. Normalization module.

$$l_j = \frac{w_j}{s} \quad j = 1, 2, \ldots, 128 \tag{39}$$

$$l_j = \frac{w_j}{2^n} \times m \quad j = 1, 2, \ldots, 128 \tag{40}$$

$$m = \frac{2^n \times 1024}{s} \tag{41}$$

V. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of all modules in the proposed hardware and compare the computation time achieved by the proposed FPGA hardware, NIOS II, and a PC. The computation platform is a personal computer with an Intel Core i7-3770, a 3.4-GHz CPU, and 8 GB of RAM. The simulation environment is Nios II and Visual Studio 2010 Express with OpenCV library under Win 7. The hardware used is an Altera DE2i-150 board of FPGA with Cyclone IV GX: EP4CGX150DF31C7, and the system frequency is 50 MHz.

Table 1 shows the system performances of all modules in the proposed hardware system, including logic elements (LE), memory bits, and nine-bit multiplier.

TABLE 1. The hardware resources utilized for each system modules of proposed method.

Module	Total LE	Registers	Total memory bits	Multiplier 9×9 bits
Image pyramid	5746	2852	47880	72
SIFT detection	1482	968	59135	58
SIFT descriptor	58475	35805	260256	512

A. COMPUTATION TIME OF GAUSSIAN IMAGE

Table 2 shows the computation time of the Gaussian filter for two different image sizes on different platforms, including FPGA, NIOS II, and PC. The mask size of the Gaussian filter is 7 × 7, and the Gaussian sigma is 1.6. We also calculate the differences in grey value of 0, 1, and 2 between the original image and the image processed by the proposed Gaussian filter, as shown in Table 3. The size of the given images is 800 × 480. Note that three pixels around the edges are ignored in Table 3. The results indicate that the proposed method has achieved almost the same performance level as that by software. The reason for the image error of the Gaussian filter is that we only use a finite number of 10 bits for shifting to solve the floating point problem. Based on Table 3, the Peak Signal-to-Noise Ratio (PSNR) can be calculated as 43.6694 db. The original image is shown in Fig. 20, while Fig. 21 shows the Gaussian image using a Gaussian operation to process the original image by FPGA.

TABLE 2. Computation time of gaussian filter.

Gaussian Filter	FPGA	NIOS II	PC
Device	Cyclone IV GX: EP4CGX150DF31C7		Intel Core i7-3770
Frequency	50M Hz		3.4G Hz
Image size : 256 × 256			
Time	0.00133 s	4.32266 s	0.008 s
multiplying factor	×1	×3250	×6
Image size : 800 × 480			
Time	0.00773 s	27.19163 s	0.048 s
multiplying factor	×1	×3517	×6

TABLE 3. Results of gaussian filter error.

Difference in grey value	0	1	2	$2^{\uparrow}$
Quantity of pixels	151827	224253	276	0

B. COMPUTATION TIME OF IMAGE PYRAMID

Table 4 shows the computation time of the image pyramid executed on three platforms. We can see that the Gaussian

FIGURE 20. Original image.

FIGURE 21. Gaussian image by FPGA.

TABLE 4. Computation time of image pyramid executed on three different platforms.

IMAGE PYRAMID	FPGA	NIOS II	PC
FREQUENCY	50 MHZ		3.4 GHZ
IMAGE SIZE : 256×256			
TIME	0.00133 S	25.69534 S	0.048 S
MULTIPLYING FACTOR	×1	×19320	×36
IMAGE SIZE : 800×480			
TIME	0.00773 S	163.16930 S	0.288 S
MULTIPLYING FACTOR	×1	×21109	×37

filter (Table 2) and the image pyramid (Table 4) have the same computation time by hardware, because all Gaussian images and DOG images can be processed in parallel.

Note that the computation time for the image pyramid is not affected by the proposed FPGA design, in comparison to the Gaussian filter, thanks to parallel computation of the proposed hardware architecture, while the computation time of the image pyramid by PC is six times longer than that by the Gaussian filter.

C. COMPUTATION TIME OF SIFT DETECTION

The result of SIFT detection by FPGA hardware is shown in Fig. 22, in which the red points represent the keypoints. Table 5 shows the computation time of SIFT detection for two different image sizes. From Tables 4 and 5, we can find that the larger the image size, the more processing time the proposed FPGA hardware can save.

D. COMPUTATION TIME OF SIFT DETECTION

TABLE 6. compares hardware resource usage and maximum operation frequency between the traditional inverse matrix method and the proposed method. The traditional method, which uses an inverse matrix to implement the hardware, requires nine dividers to perform the operations. In the proposed method, we only use the inequality to determine the

FIGURE 22. Image showing the result of the SIFT feature detection by FPGA hardware.

TABLE 5. Computation time of SIFT detection.

SIFT DETECTION	FPGA	NIOS II	PC
FREQUENCY	50M HZ		3.4G HZ
IMAGE SIZE : 256×256 (NUMBER OF KEYPOINTS : 1363)			
TIME	0.0013 S	27.0917 S	0.396 S
MULTIPLYING FACTOR	×1	×20840	×305
IMAGE SIZE : 800×480 (NUMBER OF KEYPOINTS : 3874)			
TIME	0.0077 S	171.7987 S	2.5220 S
MULTIPLYING FACTOR	×1	×22312	×328

TABLE 6. Hardware resources used by traditional inverse method and proposed method and their mximum operation frequency.

MODULE / METHOD	TOTAL LE	TOTAL MEMORY BITS	MULT 9×9 BITS	FMAX (MHZ)
TRADITIONAL INVERSE METHOD	8057	0	30	9.69
PROPOSED	409	0	30	135.21

low-contrast keypoints. Thus, we can achieve a satisfactory performance, as shown in Table 6.

Note that the total LE required is much smaller and FMAX is significantly increased, because no dividers are used.

E. IMAGE GRADIENT ERROR CALCULATION

We enter four quadrants of X and Y values into the CORDIC module implemented by hardware and obtain an approximation value for $\sqrt{X^2+Y^2}$ and $\tan^{-1}\left(\frac{Y}{X}\right)$ in Eq. (12) and (13). Note that the error is satisfactorily small for practical usage. Table 7 shows the calculation results of CORDIC, and Table 8 shows the hardware resource usage of CORDIC.

To evaluate the benefits and drawbacks of the proposed method, we compare twelve state-of-the-art hardware implementations of SIFT in terms of their performance, namely detection time, descriptor calculation time and frame rate, and their hardware resource usage (e.g. gates, registers, multiplexers and RAM). The results are shown in Table 9 and Table 10, respectively.

TABLE 7. Calculation results of CORDIC hardware.

X	100	–100	100	–100
Y	100	100	-100	–100
$\sqrt{X^2+Y^2}$	141.4326	141.4326	141.4326	141.4326
$\tan^{-1}\left(\frac{Y}{X}\right)$	44.9990	134.9990	224.9990	314.9990

Most works (e.g. Vourvoulakis *et al.* [14], Chiu *et al.* [24], Wang *et al.* [12] and Jiang *et al.* [30]) reported in the literature use sequential structures to implement the descriptor.

TABLE 8. The hardware resource usage of CORDIC.

Module	Total LE	Registers	Total Memory bits	Multiplier 9×9 bits
CORDIC	934	562	30	0

We can only compare performance and hardware resource usage with [14], not only because their work achieved an exceptional performance reported in the literature, but also because the hardware platform and image size used are the same. As shown in Table 9, detection time and descriptor calculation time required by the proposed method are less than those by [14] and therefore the frame rate of the proposed architecture can be as high as 150 fps. As far as hardware resource usage is concerned, the proposed architecture greatly reduces usage of LUTs/ gates, which leaves more

TABLE 9. Performance of the proposed method and twevle sate-of-the-art implementaion.

	Image size	Detection time	Descriptor calculation time	Frame rate
Vourvoulakis[14]	640 × 480	6.8 ms	46 ns/feature	≈ 70 fps
Yao [11]	640 × 480	31 ms	–	≈ 32 fps
Mizuno [16]	640 × 480	17.8 ms	–	≈ 56 fps
Mizuno [17]	1920 × 1080	N/A	N/A	30 fps
Chang [18]	320 × 240	1.1 ms	–	≈900 fps
Zhong [19]	320 × 256	10 ms	80 µs/feature	100 fps
Huang [20]	640 × 480	3.4 ms	33.1 µs/feature	≈30 fps
Suzuki [21][22]	1920 × 1080	N/A	N/A	60 fps
Kim [23]	320 × 240	N/A	60 µs/feature	≈ 30 fps
Chiu [24]	640 × 480	N/A	5.5 µs/feature	≈ 30 fps
Wang [12]	1280 × 720	8 ms	8.3 µs/feature	60 fps
Jiang [13]	512 × 512	6.55 ms	2.23 µs/feature	≈150 fps
Proposed	640 × 480	6.145 ms	20 ns/feature	150 fps

TABLE 10. Hardware resource usage of the proposed method and twevle sate-of-the-art implementaions.

	LUTs/ gates	Registers/FF	DSP/Mult.	RAM
Vourvoulakis [14]	125,644	8372	77	406 Kbits
Yao [11]	35,889	19,529	97	3240 Kbits
Mizuno [16]	32,592	23,247	258	0.67 Mbits
Mizuno [17]	1.1M gates	–	–	1.38 Mbits
Chang [18]	5554	5676	N/A	1944 Kbits
Zhong [19]	18,195	11,821	56	2808 Kbits
Huang [20]	1320 gates	–	–	5729 Mbits
Suzuki [21][22]	108,322	55,407	3	3204 Kbits
Kim [23]	16,832	5729	8	75,240 bits
Chiu [24]	57,598	24,988	8	1206 Kbits
Wang [12]	18,437	13,007	52	4932 Kbits
Jiang [13]	26,398	10,310	89	7.8 Mbits
Proposed	65,560	39,482	642	319248 bits

FIGURE 23. Experiment illustrating a successful matching rate = 99.29 %, where 141 out of 142 features are successfully matched.

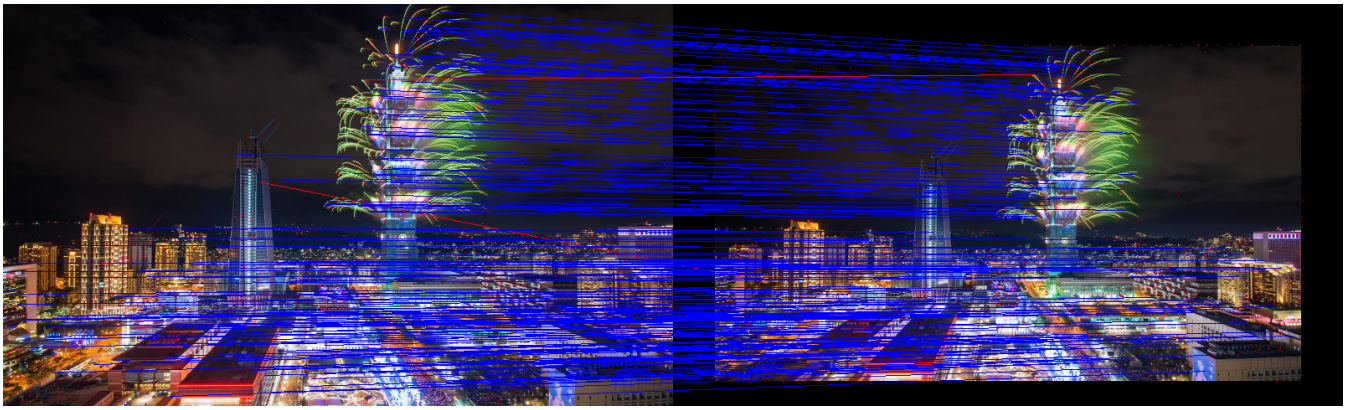


FIGURE 24. Experiment illustrating a successful matching rate = 99.03 %, where 205 out of 207 features are successfully matched.

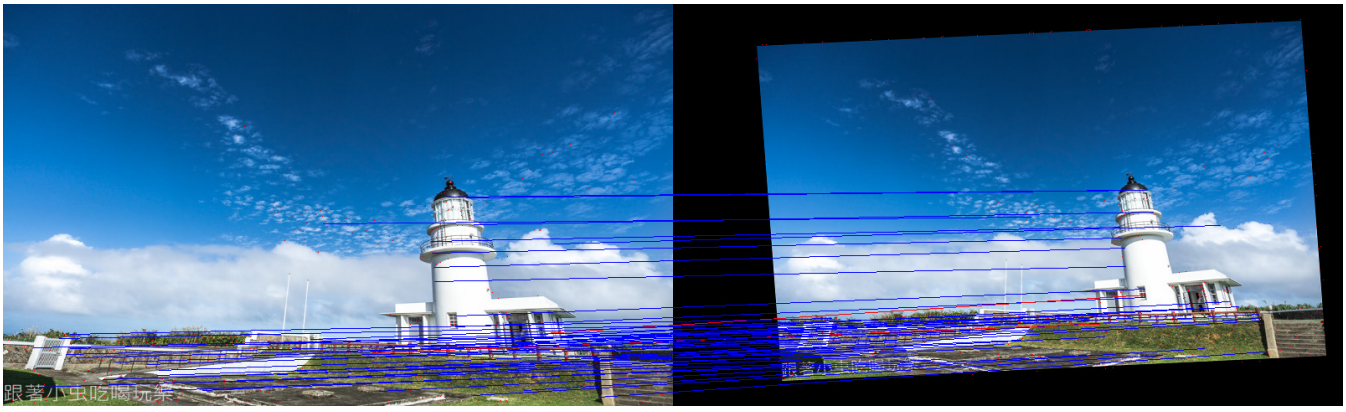


FIGURE 25. Experiment illustrating a successful matching rate = 96.15 %, where 50 out of 52 features are successfully matched.

room to make the SIFT hardware more applicable for real-world applications, as shown in Table 10.

F. MATCHING ACCURACY

As soon as the hardware modules are constructed, experiments can be conducted to validate the matching accuracy based on the obtained SIFT descriptors by the proposed hardware design method. Taking image sets in 10 different scenes, where the images bear different degrees of scaling, rotation, and view angles in each image set, we obtain a satisfactory average matching accuracy of about 97.84% for the image sets. This suffices to show the robustness and effectiveness of the obtained SIFT descriptors. As illustrative examples, Figs. 23-25 show the matching results of 3 image sets, where matching accuracy is 99.29%, 99.03%, and 96.15%, respectively.

VI. CONCLUSION

The SIFT algorithm uses scale-space images to achieve feature point detection and description with good results to changes in scale or rotation. Thus, matching performance of SIFT is more stable and robust compared with other image

recognition algorithms. However, the steps to implement the SIFT algorithm are complex and the processing time is much too long. It is therefore a non-trivial task to achieve a real-time realization for SIFT algorithm using only a single PC. In this paper, a hardware design method for SIFT on FPGA is proposed, using a pipeline and parallel process to speed up the SIFT algorithm. Several new methods have also been proposed in this paper to raise the system frequency of the SIFT hardware. For example, we offline calculate the Gaussian mask values to reduce computing costs. The use of dividers to calculate the inverse matrix in the low-contrast detection module is avoided. In the normalization module, the use of dividers is also avoided to save resources. As a result, utilization of logic elements required in the FPGA hardware is greatly reduced and system frequency is significantly increased. From the experimental results, the proposed FPGA-based architecture for SIFT compares favorably with the state-of-the-art implementations to achieve real-time image processing.

REFERENCES

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[2] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. 4th Alvey Vis. Conf.*, Manchester, U.K., 1988, pp. 147–151.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Comput. Vis. Image Understand.*, vol. 110, no. 3, pp. 404–417, Jun. 2008.

[4] Y. Ke and R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, Jul. 2004, pp. 506–513.

[5] L. Juan and O. Gwun, "A comparison of SIFT, PCA-SIFT and SURF," *Int. J. Image Process.*, vol. 3, no. 4, pp. 143–152, 2009.

[6] G. Yu and J.-M. Morel, "ASIFT: An algorithm for fully affine invariant comparison," *Image Process. Line*, vol. 1, pp. 11–38, Feb. 2011.

[7] H. Altwaijry, A. Veit, and S. Belongie, "Learning to detect and match keypoints with deep architectures," in *Proc. Brit. Mach. Vis. Conf.*, Sep. 2016, pp. 1–12.

[8] J. Wang, J. Lin, and Z. Wang, "Efficient hardware architectures for deep convolutional neural network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 6, pp. 1941–1953, Jun. 2018.

[9] K. Guo *et al.*, "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[10] V. Bonato, E. Marques, and G. A. Constantinides, "A parallel hardware architecture for scale and rotation invariant feature detection," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 12, pp. 1703–1712, Dec. 2008.

[11] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao, and W. Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *Proc. Int. Conf. Field-Program. Technol.*, Sydney, NSW, Australia, Dec. 2009, pp. 30–37.

[12] J. Wang, S. Zhong, L. Yan, and Z. Cao, "An embedded system-on-chip architecture for real-time visual detection and matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 3, pp. 525–538, Mar. 2014.

[13] J. Jiang, X. Li, and G. Zhang, "SIFT hardware implementation for real-time image feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 7, pp. 1209–1220, Jul. 2014.

[14] J. Vourvoulankis, J. Kalomiros, and J. Lygouras, "Fully pipelined FPGA-based architecture for real-time SIFT extraction," *Microprocess. Microsyst.*, vol. 40, pp. 53–73, Feb. 2016.

[15] K. A. Acharya, R. V. Babu, and S. S. Vadhiyar, "A real-time implementation of SIFT using GPU," *J. Real-Time Image Process.*, vol. 14, no. 2, pp. 267–277, 2018.

[16] K. Mizuno *et al.*, "Fast and low-memory-bandwidth architecture of sift descriptor generation with scalability on speed and accuracy for VGA video," in *Proc. Int. Conf. Field Program. Logic Appl.*, Milan, Italy, Aug./Sep. 2010, pp. 608–611.

[17] K. Mizuno *et al.*, "A low-power real-time SIFT descriptor generation engine for full-HDTV video recognition," *IEICE Trans. Electron.*, vol. E94.C, no. 4, pp. 448–457, 2011.

[18] L. Chang, J. Hernández-Palancar, L. E. Sucar, and M. Arias-Estrada, "FPGA-based detection of SIFT interest keypoints," *Mach. Vis. Appl.*, vol. 24, no. 2, pp. 371–392, 2012.

[19] S. Zhong, J. Wang, L. Yan, L. Kang, and Z. Cao, "A real-time embedded architecture for SIFT," *J. Syst. Archit.*, vol. 59, no. 1, pp. 16–29, 2013.

[20] F.-C. Huang, S.-Y. Huang, J.-W. Ker, and Y.-C. Chen, "High-performance SIFT hardware accelerator for real-time image feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 3, pp. 340–351, Mar. 2012.

[21] T. Suzuki and T. Ikenaga, "SIFT-based low complexity keypoint extraction and its real-time hardware implementation for full-HD video," in *Proc. Asia–Pacific Signal Inf. Process. Assoc. Annu. Summit Conf.*, Hollywood, CA, USA, Dec. 2012, pp. 1–6.

[22] T. Suzuki and T. Ikenaga, "Low complexity keypoint extraction based on SIFT descriptor and its hardware implementation for full-HD 60 fps video," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E96.A, no. 6, pp. 1376–1383, 2013.

[23] E. S. Kim and H.-J. Lee, "A novel hardware design for SIFT generation with reduced memory requirement," *J. Semicond. Technol. Sci.*, vol. 13, no. 2, pp. 157–169, Apr. 2013.

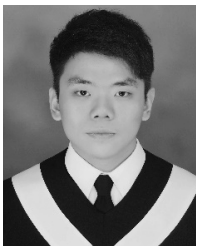
[24] L.-C. Chiu, T.-S. Chang, J.-Y. Chen, and N. Y.-C. Chang, "Fast SIFT design for real-time visual feature extraction," *IEEE Trans. Image Process.*, vol. 22, no. 8, pp. 3158–3167, Aug. 2013.

SHIH-AN LI received the B.S. degree in electrical engineering from the National Taipei University of Technology, Taipei, Taiwan, in 1997, and the M.S. and Ph.D. degrees in electrical engineering from Tamkang University, Taipei, Taiwan, in 2004 and 2008, respectively. In 2010, he joined the Department of Electrical Engineering and Computer Engineering, Tamkang University, where he is currently an Assistant Professor. His research interests include fuzzy system, intelligent control, SOPC design, FPGA design, and genetic algorithms.

WEI-YEN WANG (F'13) received the Diploma degree in electrical engineering from the National Taipei Institute of Technology in 1984, and the M.S. and Ph.D. degrees in electrical engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 1990 and 1994, respectively.

From 1990 to 2006, he was a Patent Screening Member with the National Intellectual Property Office, Ministry of Economic Affairs, Taiwan. Since 2003, he has been certified as a the Patent Attorney in Taiwan. In 1994, he was an Associate Professor with the Department of Electronic Engineering, St. John's and St. Mary's Institute of Technology, Taiwan. From 1998 to 2000, he was with the Department of Business Mathematics, Soochow University, Taiwan. From 2000 to 2004, he was with the Department of Electronic Engineering, Fu-Jen Catholic University, Taiwan. In 2004, he was a Full Professor with the Department of Electronic Engineering, Fu-Jen Catholic University. In 2006, he was a Professor and the Director of the Computer Center, National Taipei University of Technology, Taiwan. From 2007 to 2014, he was a Professor with the Department of Applied Electronics Technology, National Taiwan Normal University, Taiwan. From 2011 to 2013, he was the Director of the Information Technology Center, National Taiwan Normal University, Taiwan. He is currently a Professor with the Department of Electrical Engineering, National Taiwan Normal University, Taiwan. His current research interests and publications are in the areas of fuzzy logic control, robust adaptive control, neural networks, computer-aided design, digital control, and CCD camera based sensors. He has authored or coauthored over 200 refereed conference and journal papers in the above areas.

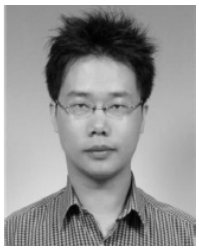
Dr. Wang is also an IET Fellow and CACS Fellow. He is a recipient of the Best Associate Editor Award of the IEEE TRANSACTIONS ON CYBERNETICS. He is currently an Associate Editor of the IEEE Transactions on CYBERNETICS, and the *International Journal of Fuzzy Systems*.



WEI-ZHENG PAN received the B.S. degree from Tamkang University in 2014 and the M.S. degree from National Taiwan Normal University, Taipei, Taiwan, in 2016, all in electrical engineering. He is currently a Research Assistant with the Computational Intelligence Lab, National Taiwan Normal University, Taiwan, where he is involved in the area of facial expression recognition and action recognition using artificial intelligence approaches. He has been studying the problems of image recognition for over 3 years. His expertise includes image recognition and SLAM algorithms.

CHEN-CHIEN JAMES HSU (SM'14) was born in Hsinchu, Taiwan. He received the B.S. degree in electronic engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 1987, the M.S. degree in control engineering from National Chiao-Tung University, Hsinchu, in 1989, and the Ph.D. degree from the School of Microelectronic Engineering, Griffith University, Brisbane, Australia, in 1997.

He was a Systems Engineer with IBM Corporation, Taipei, for three years, where he was responsible for information systems planning and application development, before commencing his Ph.D. studies. He joined the Department of Electronic Engineering, St. Johns University, Taipei, as an Assistant Professor, in 1997, and was appointed as an Associate Professor in 2004. From 2006 to 2009, he was with the Department of Electrical Engineering, Tamkang University, Taipei. He is currently a Professor with the Department of Electrical Engineering, National Taiwan Normal University, Taipei. He has authored or co-authored over 180 refereed journal and conference papers. His current research interests include digital control systems, evolutionary computation, vision-based measuring systems, sensor applications, and mobile robot navigation. He is an IET Fellow.



CHENG-KAI LU received the B.S. and M.S. degrees in electronics engineering from Fu Jen Catholic University, Taipei, Taiwan, in 2001 and 2003, respectively, and the Ph.D. degree in engineering from The University of Edinburgh, Edinburgh, U.K., in 2012. After graduation, he was the Director of the Research and Development Division, Chyao Shiunn Electronic Industrial Co., Shanghai. Apart from academic experience, he has over eight years industrial work experience. He is currently a Faculty Member with the Electrical and Electronic Engineering Department, Universiti Teknologi PETRONAS, Malaysia. He has published his research works on peer-reviewed papers (book chapters, journal papers, conferences, reports). He has also has filed a couple of patents. His research interests focus on medical imaging, embedded systems, artificial intelligence and their applications and clinical decision support systems. He has served as an Executive Member for the IEEE EMBS Malaysia Chapter from 2017 to 2018.