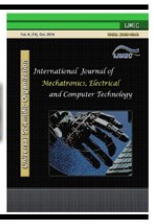
Vol. 8(27), Jan. 2018, PP. 3745-3752

Contents list available at IJMEC

International Journal of Mechatronics, Electrical and Computer Technology (IJMEC)

Journal Homepage: www.aeuso.org

PISSN: 2411-6173 - EISSN: 2305-0543



An Efficient FPGA Implementation of DAISY Descriptor based on Pipeline and Multicycle Architectures

Ensieh Iranmehr[1*] and Shohreh Kasaei[2]

[1]*Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran*

[2]*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

***Corresponding Author's E-mail:** eiranmehr@ee.sharif.edu*

Abstract

The DAISY descriptor is inspired of earlier descriptors such as SIFT and Gradient Location and Orientation Histogram (GLOH) but it can be computed much faster. Those earlier descriptors require a significant amount of computational power which makes them to be inappropriate for real-time applications. In this work, the Daisy descriptor is implemented on the Field Programming Gate Array (FPGA) in order to be appropriate for real-time applications. Because of high accuracy and low computational power of DAISY descriptor, a special structure for implementing this descriptor on FPGA is designed. Implementation results show that the DAISY descriptor can be computed on FPGA much faster than on CPU. Required time of dense computation of the DAISY descriptor on an image with size of 300×400 pixels is 4 milliseconds on FPGA whereas the required time on CPU is 1.2 seconds. The proposed architecture is implemented using Verilog and achieves real-time descriptor calculation on approximately 220 fps 300×400 video frames on 133 MHz clock.

Keywords: *DAISY Descriptor, FPGA Implementation, Pipeline, Multicycle, Real-time.*

1. Introduction

Descriptors are used in a various field of applications such as: object recognition, dense and sparse matching, and 3D reconstruction. There are a lot of descriptors with different capabilities that are used in different applications. In the following, some popular descriptors are considered. The Scale Invariant Feature Transform (SIFT) [2] is one of the most representative methods. But, the high computational complexity often makes it infeasible for real-time applications. Therefore, some researchers have explored to implement the SIFT descriptor with Graphic Processing Unit (GPU) [8], Field Programming Gate Array (FPGA) [13], or Application Specific Integrated Circuit (ASIC) [7] to accelerate the computation. Another useful descriptor is the Speeded-Up Robust Features (SURF) [5], which outperforms the implementations of SIFT on CPUs. Most significant improvement in calculation speed is achieved by using the "Integral Image". The SURF algorithm is a scale and rotation invariant robust feature detector and descriptor. It was firstly presented by Herbert Bay et al. [5]. The SURF descriptor is widely used for object tracking, traffic monitoring in image sequences, 3D reconstruction, augmented reality applications, and so on. All these applications require real-time image sequence processing. In order to solve the lack of the processor performance, the use of FPGA for SURF implementation is proposed in some papers [6]. Tola et al. [3, 4] used the log-polar grid with Gaussian

Article History:

DOI: IJMEC/10.225162

Received Date: Mar. 11, 2017

Accepted Date: Aug. 15, 2017

Available Online: Jan. 05, 2018

weights and speeded up the computation by applying Gaussian convolutions for histogram binning. They named their descriptor DAISY, due to the flower-like arrangement of the local image regions for histogram creation. Since computational speed of DAISY is more than SIFT, this descriptor can be calculated densely for every pixels of image, and therefore enabling wide-baseline stereo matching. They compared DAISY with SIFT, SURF, Normalized Cross Correlation (NCC), and pixel difference by creating dense depth maps from stereo images. DAISY and SIFT perform equally well and outperform the others. The speed of DAISY over SIFT is due to the replacement of the weighted sums from SIFT by fast-to-compute Gaussian convolutions, which are proved to be advantageous for FPGA implementation as well. Calculating the DAISY descriptor for different orientations reduces to shifting values of histograms, which is much better suited for FPGA design than the patch rotation of SIFT. In other words, DAISY natively supports its implementation on FPGA, because it treats every pixel in the same way.

Also, the usage of Gaussian masks for weighting reduces the necessary number of multiplications and additions because Gaussian convolutions are separable and symmetric, thus reducing the required amount of resources.

A popular choice for embedded machine vision is usage of the FPGA because they effectively utilize the parallel nature of image processing. Several authors [10, 13] reported successful implementations of robotic vision algorithms on a FPGA-based hardware.

As described in the following, DAISY algorithm consists of convolution with several filters. FPGA makes it possible to implement several parallel filters. Each filter may consist of thousands of elements that can be addressed with each clock signal. The use of highly parallel distribution of the processes in FPGA makes it possible to ensure a real-time video processing on lower clock rate to obtain less power consumption in comparison with the CPU. Here, we present an implementation of DAISY algorithm massively accelerated by the FPGA. The original DAISY algorithm is suitable for implementation using FPGA logical blocks.

The rest of this paper is organized as follows. In the Section 2, the DAISY descriptor algorithm is briefly explained. Section 3 describes how to implement the DAISY algorithm on FPGA in order to accelerate its computation. In this section, two architectures based on pipelining and multi-cycle are proposed. Section 4 presents the implementation results. Finally, Section 5 concludes the paper.

2. DAISY Algorithm

This section briefly describes the DAISY algorithm. DAISY samples the local gradient information in the way which is visualized by “Figure 1”. Each circle represents one histogram region, which is part of the descriptor vector. Each histogram represents the Gradient Orientations (GOs) within this region. The gradient is split into H discrete orientations, so each single histogram has H entries. The algorithm uses Q rings in the log polar arrangement around the center on which the histograms are sampled. Also, T represents the number of histograms (circles in “Figure 1”) on each ring. Therefore, the resulting descriptor has $D_S = (Q \times T + 1) \times H$ entries. The DAISY implementation uses H = 8 discrete GOs, Q =3 rings, and T =8 histograms on each ring, which means that the resulting feature vector has in total D_S = 200 entries, as can be seen in “Figure 1”.

In DAISY descriptor, several oriented derivatives of Gaussian filters have to be computed as in ‘(1)’

$$G_o^{\Sigma} = G_{\Sigma} * \left(\frac{\partial I}{\partial o}\right)^{+}, \qquad \left(\frac{\partial I}{\partial o}\right)^{+} = \max\left(\frac{\partial I}{\partial o}, 0\right) \tag{1}$$

where G_{Σ} is a Gaussian kernel and o is the orientation of the derivative. The convolution results G_o^{Σ} are referred as Convolved Orientation Maps (COMs). DAISY descriptor will be built by reading the values in the COMs. Also, convolution with large Gaussian kernel can be obtained from several consecutive convolutions with smaller kernels. Indeed $G_o^{\Sigma 2}$ can be computed by using $G_o^{\Sigma 1}$ of

$$G_o^{\Sigma 2} = G_{\Sigma 2} * \left(\frac{\partial I}{\partial o}\right)^{+} = G_{\Sigma} * G_{\Sigma 1} * \left(\frac{\partial I}{\partial o}\right)^{+} = G_{\Sigma} * G_o^{\Sigma 1}, \quad \Sigma = \sqrt{\Sigma_2^2 - \Sigma_1^2} \tag{2}$$

In the following, $h_\Sigma(u,v)$ represents the vector made of the values at location (u,v) in the Orientation Maps (OMs) after convolution by a Gaussian kernel with standard deviation Σ. It is normalized to unit norm, and the normalized vectors are denoted by

$$h_\Sigma(u,v) = [\, G_1^\Sigma(u,v), \dots, \, G_8^\Sigma(u,v)]^T \quad (3)$$

The full DAISY descriptor $D(u_0,v_0)$ for location (u_0,v_0) is then defined as a concatenation of $\widetilde{h_\Sigma}$ vectors, and can be defined by

$$D(u_0,v_0) = \Big[\, \widetilde{h_{\Sigma 1}}^T(u_0,v_0),\ \widetilde{h_{\Sigma 1}}^T\big(I_1(u_0,v_0,R_1)\big), \dots, \widetilde{h_{\Sigma 1}}^T\big(I_N(u_0,v_0,R_1)\big),$$
$$\widetilde{h_{\Sigma 2}}^T\big(I_1(u_0,v_0,R_2)\big), \dots, \widetilde{h_{\Sigma 2}}^T\big(I_N(u_0,v_0,R_2)\big),\ \widetilde{h_{\Sigma 3}}^T\big(I_1(u_0,v_0,R_3)\big), \dots, \widetilde{h_{\Sigma 3}}^T\big(I_N(u_0,v_0,R_3)\big)\Big] \quad (4)$$

where $\big(I_j(u_0,v_0,R_1)\big)$ is the location with distance R_1 from (u_0,v_0) in the direction given by j when the directions are quantized into N values.

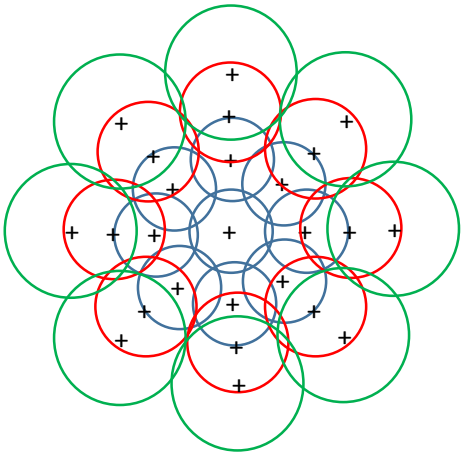


Figure 1: DAISY descriptor. Each circle represents a region where the radius is proportional to the standard deviations of the Gaussian kernels. The '+' sign represents the locations where we sample the Convolved Orientation Maps. By using these sampled Convolved Orientation Maps, the descriptor at each pixel location of image is computed. By overlapping the regions, smooth transitions among regions and a degree of rotational robustness is achieved.

3. Proposed FPGA Implementation

This section is dedicated to describe hardware design and software controlling functions. As mentioned before, Daisy descriptor is computed faster than other descriptors. Also, it is suitable to be implemented on FPGA because of compatibility to FPGA structure. Calculation of DAISY descriptor consists of computing gradients of the image in different orientations and then smoothing GOs. It is performed by sliding kernels over an image and computing convolutions.

3.1. Proposed Structure of Kernel Core

In order to compute gradient of an image in different orientations, kernels have to be convolved with that image. "Figure 2" shows a kernel core which implements a convolution between a 3×3 kernel and an image. A structure is proposed based on pipelining in order to calculate each pixel value of gradient image in a clock cycle. Here, 4 pipeline stages are considered in order to compute convolution. For convolving a 3×3 kernel with an image, 9 multiplication and 8 addition operators are needed as it can be seen from "Figure 2". Also, 4 registers have to be assumed in order to implement 4 pipeline stages. By sliding a 3×3 kernel over an image with one pixel displacement, 3 new pixel values have to be read from block RAMs and 6 pixel values have to be shifted. Thus, a controller has to be considered in order to determine the address of these 3 block RAMs. Furthermore, a block RAM is considered for saving the output of last stage of the pipeline. The

address of output block RAM has to be selected by the controller. For an image with size of 300×400 pixels and by considering 200 MHz clock rate, the convolution result (OM) is computed in approximately 0.6 milliseconds as it can be calculated in'5'.

$$\frac{1}{200\ MHz} \times (300 \times 400\ pixels) \cong 0.6\ milliseconds \qquad (5)$$

As mentioned in previous section on which DAISY descriptor is explained briefly, after computing the OM, it has to be convolved with several Gaussian kernels in order to achieve COMs. Therefore, by assuming different size of kernels for smoothing, similar structure with different amount of resources has to be considered. For example by assuming a 5×5 kernel, 25 multiplications, 12 additions, and 5 pipeline stages are needed in order to compute the COM. To decrease the computational time consuming, instead of using a larger kernel to compute a smoother COM, another 5×5 kernel is needed to convolve with previous COM.

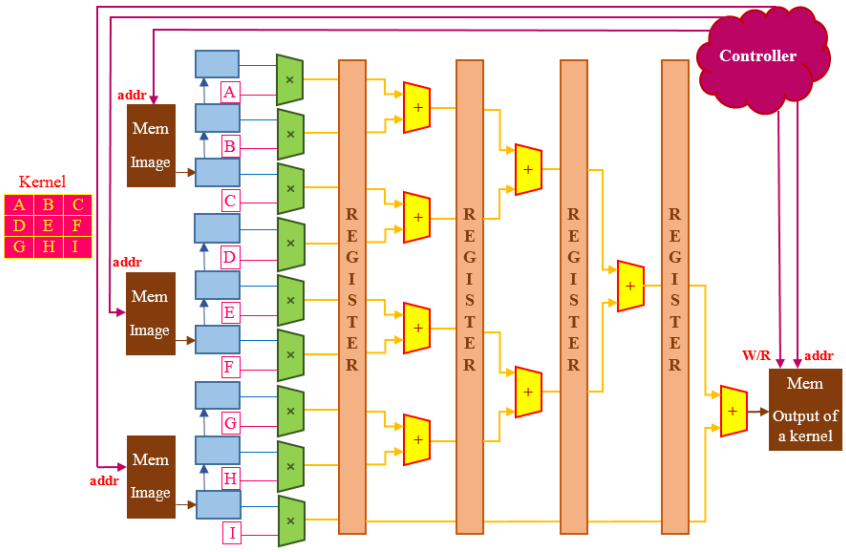


Figure 2: Kernel core 3×3. Implementation of convolving an image with a kernel is performed by sliding a kernel over an image. Parallelism and pipelining are utilized in order to speed up DAISY calculation. 4 pipeline stages are considered in order to calculate pixel values of GO in a clock cycle.

3.2. Proposed Structure based on Pipelining Architecture

The first structure based on pipelining for implementing DAISY descriptor on FPGA is proposed in this subsection. To speed up the DAISY computation, parallelism and pipelining are utilized. To do so, 4 pipeline stages is considered in order to calculate the DAISY descriptor of an image. By considering 4 gradient kernels in order to compute gradients of image in 4 different orientations $H = 4$, a compatible structure with FPGA architecture which is depicted in "Figure 3" is proposed in this paper. In order to compute gradient images in 4 different orientations, 4 gradient kernels are considered. "Figure 3" shows the steps of computing the convolutions between an image and gradient kernels. First of all, a kernel is convolved with an image and convolution image which is named OM is stored in the first stage of pipeline block RAM. It is worth to mention that the size of each block RAM is proportional to the size of the image. As mentioned in previous section, three different radiuses $\sum_1, \sum_2, \sum_3$ for Gaussian kernels are considered to smooth gradient images. If the relations between these radiuses are so as $\sqrt{{\sum_2}^2 - {\sum_1}^2} = \sqrt{{\sum_3}^2 - {\sum_2}^2} = \sum$, instead of using different radiuses of Gaussian kernel which has to convolve with a gradient image, we can use just a kernel with radius of $\sum$ which is convolved with the COM.

In the second stage of the pipeline, the OM is convolved with the Gaussian kernel and the result of convolution is stored in the first block RAM of the second stage. In this implementation, just a single Gaussian kernel core is considered.

In the third stage of the pipeline, $\sum_1$-COM are convolved with the Gaussian kernel and the result of this stage is stored in the first block RAM of this stage. Also, in this stage, the first block RAM of the second stage of the pipeline is copied into the second block RAM.

In the fourth stage of the pipeline, $\sum_2$-COM are convolved with the Gaussian kernel and the result of this stage is stored in the block RAM of this stage. Moreover, the second block RAM of the second stage of the pipeline is copied into the third block RAM and the first block RAM of the third stage of the pipeline is copied into the second block RAM. Now, there are 3 different COMs which are saved in the third block RAM of the second stage of the pipeline, the second block RAM of the third stage of the pipeline, and the block RAM of the fourth stage of the pipeline.

Overall, there are $4\ directions \times 7\ block\ RAMs = 28$ block RAMs in this structure. It means that for an image with size of 300×400 by considering 8 bit for intensity value, $300 \times 400 \times 8 \times 28 = 26{,}880\ Kbit$ RAM is needed. Furthermore, the latency of one frame is 0.908×4 = 3.632 milliseconds. In this fast method, a large amount of block RAM is needed. For an image with size of 300×400, this structure computes DAISY descriptor in approximately 0.908 milliseconds which means 1100 fps 300×400 video frames on 165 MHz clock.

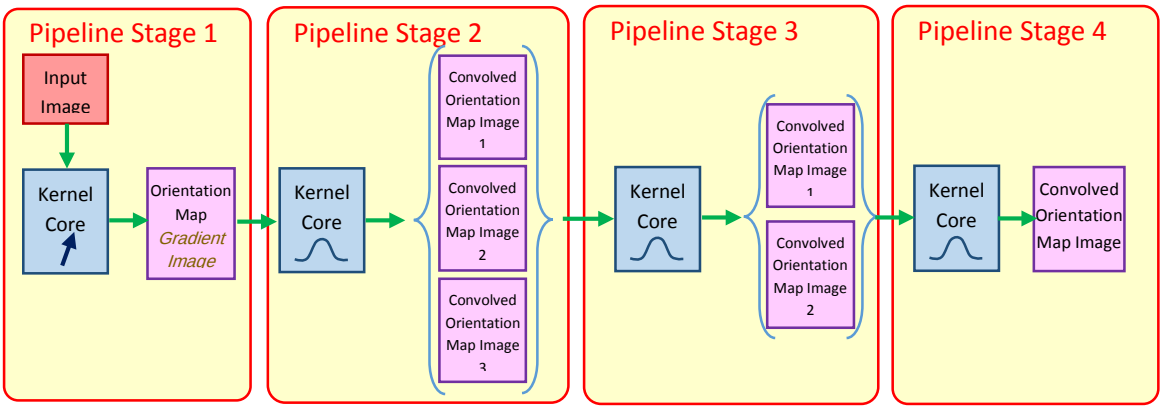


Figure 3: Proposed structure based on pipelining for implementing DAISY descriptor on FPGA. Parallelism and pipelining are utilized in order to speed up DAISY computation. Four pipeline stages are considered to calculate DAISY descriptor of an image.

3.3. Proposed Structure based on Multi-Cycle Architecture

As mentioned before, the previous structure needs a large amount of block RAMs. In order to overcome this disadvantage, another structure is proposed based on multi-cycle architecture for implementing DAISY descriptor on FPGA. The structure for each GO is shown in "Figure 4". For each GO, there are three block RAMs for storing three different Σ- COM.

In this subsection, the structure based on multi-cycle architecture is presented. First of all, a controller is needed to control multiplexers. The type of kernel which includes the gradient kernel or Gaussian kernel is chosen by the multiplexer. This chosen kernel is convolved with an input image by using the kernel core. Three different Σ- COMs are stored in the Image Memory A, B, and C. Moreover, COM can be selected instead of input image by multiplexer and it can be convolved with the Gaussian kernel again and the result is stored on Image Memory B. Also, if the result is convolved with the Gaussian kernel again, the convolution result is stored on Image Memory C.

In other words, there are 3 block RAMs for each GO. By considering 4 gradient's kernels, 12 block RAMs are needed for implementing the DAISY descriptor on FPGA. It means that for an image with size of 300×400, $300 \times 400 \times 8 \times 12 = 11{,}520\ Kbit$ RAM is needed. For an image with the size of 300×400, this structure computes the DAISY descriptor in approximately 4.56 milliseconds which means 220 fps 300×400 video frames on 133 MHz clock.

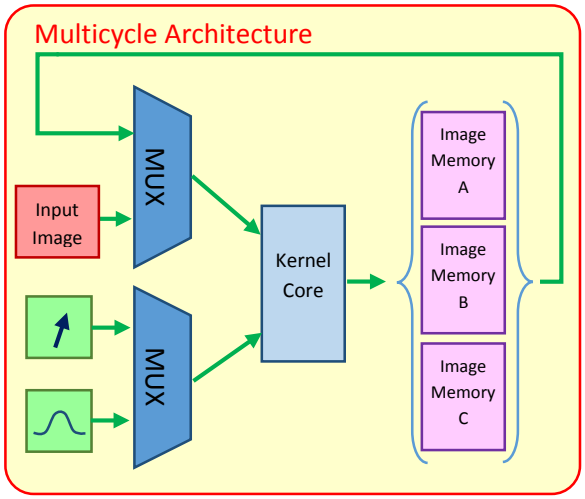


Figure 4: Proposed structure based on multi-cycle architecture for implementing DAISY descriptor on FPGA. This unit is designed for each gradient orientation.

4. Implementation Results

The final implementation clock is considered 125 MHz. All of the experiments are performed on Virtex 7 XC7VX690T. It provides enough resources for the proposed structures. It is shown that by implementing DAISY descriptor on FPGA, the DAISY descriptor of a video frame with size of 640×480 pixels can be computed in 10 millisecond.

In order to demonstrate the efficiency of the proposed method, its performance is compared with the other methods. The FPGA implementation of the optimized SIFT feature detection for an image matcher in [9] has shown that the feature detection module requires 31 milliseconds to process a typical image of 640×480 pixels. Also, 33 milliseconds is required for an image of 320×240 pixels in [13].

In order to demonstrate the efficiency of the proposed method in comparison with the GPU-based descriptor calculation, the speed of the DAISY descriptor calculation which is implemented on FPGA is shown in "Table 1". The speed of DAISY descriptor calculation implemented on FPGA in this paper is approximately 100 fps on 640×480 video frames compared with the 100fps at 640×480 of the SURF GPU implementation [11] or the 2048×1536 at 7 fps of the SIFT GPU system [1]. Furthermore, [12] has implemented a SURF variation on an NVIDIA GeForce 880 GTX, GPU. The maximum performance which is achieved based on that paper results is mentioned 70fps.

In order to measure the efficiency of the proposed system, different situations are considered and these two proposed structures are implemented on FPGA. "Table 1" shows implementation results by considering different situations. This includes two different image sizes, different number of rings of DAISY descriptor, and different number of bins which demonstrates the number of considered GO. In this table, two structures which are mentioned in previous subsections, are applied for DAISY implementation on FPGA. The first structure is based on a pipeline architecture and the second one is based on a multi-cycle architecture. Each of these structures has its own advantages. As it can be seen from "Table 1", the structure based on the pipeline architecture needs a large amount of block RAMs but it is very fast while the multi-cycle architecture needs block RAMs less than pipeline architecture but it is slower than pipeline architecture. The structure based on a multi-cycle is more appropriate for implementing the DAISY descriptor on FPGA because of less required block RAM. The amount of occupied slices and occupied block RAMs is computed for each situation. Furthermore, the consuming time of DAISY descriptor computation of a frame is calculated.

Table 1: Resources needed for FPGA implementation on Virtex 7 XC7VX690T.

Input Image Size	Number of Rings (Q)	Number of bins (H)	Architecture type	Occupied slices (%)	Occupied Block RAM (%)	Max CLK (MHz)	Computational time for a frame (ms)
300 × 400	2	4	Pipeline	6.53	28.9	185	0.81
300 × 400	2	8	Pipeline	12.31	57.89	170	0.882
300 × 400	3	4	Pipeline	16.73	50.79	165	0.908
300 × 400	2	4	Multicycle	6.28	14.51	145	3.1
300 × 400	2	8	Multicycle	11.85	28.94	140	3.21
300 × 400	3	4	Multicycle	6.83	21.76	133	4.56
300 × 400	3	8	Multicycle	12.47	43.53	133	4.511
480 × 640	2	4	Multicycle	6.76	37.15	125	9.96

CONCLUSION

FPGA implementation of DAISY descriptor enables its real-time performance. In this paper, an implementation of the DAISY algorithm accelerated by the FPGA logic is presented. DAISY algorithm consists of convolving image with several filters which are possible to be implemented on FPGA properly. A structure for kernel core is proposed in order to compute the convolution. Additionally, two different structures based on pipeline and multi-cycle architectures are also proposed to implement DAISY on FPGA. The FPGA-DAISY implementation achieves about 100 fps at VGA (480×640 pixels) resolution, which is a necessity for real-time applications.

References

[1] C. Wu, “SiftGPU: A GPU implementation of scale invariant feature transform (SIFT),” http://www.cs.unc.edu/~ccwu/siftgpu/, 2007.

[2] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, 2004.

[3] E. Tola, V. Lepetit, and P. Fua. “A fast local descriptor for dense matching,” In CVPR, 2008.

[4] E. Tola, V. Lepetit, and P. Fua, “DAISY: An Efficient Dense Descriptor Applied to Wide-Baseline Stereo,” IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 32, NO. 5, MAY 2010.

[5] H. Bay, T. Tuytelaars, and L. Gool, “Surf: Speeded up robust features,” in Proceedings of the ninth European Conference on Computer Vision, 2006.

[6] J. Švab, T. Krajnik, J. Faigl, and L. Preucil, "FPGA based Speeded up Robust Features". IEEE International Conference on TePRA, pp.35-41, 2009.

[7] J. -Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H.-J. Yoo, “A 201.4 GOPS 496 mW Real-Time Multi-Object Recognition Processor with Bio-Inspired Neural Perception Engine,” IEEE Journals & Magazines, vol. 45, No. 1, pp. 32-45, 2010.

[8] J. Kim, E. Park, X. Cui, H.Kim, and W. A. Gruver, “A fast feature extraction in object recognition using parallel processing on CPU and GPU,” IEEE International Conference on Systems, pp. 3842-3847, 2009.

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

[10] M. Tornow, J. Kaszubiak, R. W. Kuhn, B. Michaelis, and T. Schindler, "Hardware approach for real time machine stereo vision," 9th World Multi-Conference on Systemics, Cybernetics and Informatics, Vol 5, pp. 111–116, 2005.

[11] N. Cornelis, and L. V. Gool, "Fast scale invariant feature detection and matching on programmable graphics hardware," in Computer Vision and Pattern Recognition Workshops. IEEE Computer Society Conference on, pp. 1–8, 2008.

[12] T. B. Terriberry, L. M. French, and J. Helmsen, "GPU Accelerating Speeded-Up Robust Features", International Journal of Parallel Programming, 2009.

[13] V. Bonato, E. Marques, G. A. Constantinides, "A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection," IEEE Transactions on Circuits and Systems for Video Technology, vol. 18, no. 12, 2008.

Authors

Ensieh Iranmehr received the B.Sc. degree in electrical engineering from Shahed University, Tehran, Iran, in 2010 and the M.Sc. degree in digital electronics from Amirkabir University of Technology, Tehran, Iran, in 2013. She is currently Ph.D. student in digital electronics of Sharif University of Technology, Tehran, Iran. She works currently on Learning in Spiking Neural Networks. Her research interests include Neural Networks, Fuzzy Systems, Machine Learning Algorithms, Brain Simulation, Soft Computing, FPGA, GPU, and Image Processing.

email: eiranmehr@ee.sharif.edu

Prof. Shohreh Kasaei received the B.Sc. degree from the Department of Electronics, Faculty of Electrical and Computer Engineering, Isfahan University of Technology, Khomeynishahr, Iran, in 1986, the M.Sc. degree from the Graduate School of Engineering, Department of Electrical and Electronic Engineering, University of the Ryukyus, Okinawa, Japan, in 1994, and the Ph.D. degree from Signal Processing Research Centre, School of Electrical and Electronic Systems Engineering, Queensland University of Technology, Queensland, Australia, in 1998.

She is currently a Full Professor and the Director of Image Processing Laboratory (IPL) with the Sharif University of Technology since 1999. Her current research interests include multiresolution texture analysis, 3-D computer vision, 3-D object tracking, scalable video coding, image retrieval, video indexing, face recognition, hyperspectral change detection, video restoration, and fingerprint authentication.

email: skasaei@sharif.edu