Journal of Real-Time Image Processing
https://doi.org/10.1007/s11554-018-0781-0

ORIGINAL RESEARCH PAPER

All-hardware SIFT implementation for real-time VGA images feature extraction

Ginés Doménech-Asensi[1] · Juan Zapata-Pérez[1] · Ramón Ruiz-Merino[1] · José Alejandro López-Alcantud[1] · José Ángel Díaz-Madrid[3] · Víctor Manuel Brea[2] · Paula López[2]

Received: 24 July 2017 / Accepted: 23 April 2018

Abstract

This paper presents a real time hardware implementation of the scale invariant feature transform (SIFT) algorithm. To achieve real time requirements, pipeline structures have been widely exploited both in the keypoint extraction and in the descriptor generation stages. Simplifications to the original algorithm have been also applied to allow a simpler hardware implementation. The proposed architecture has been synthesized on a Xilinx Virtex 5 FPGA. It generates 3072 descriptor vectors for VGA images at 99 frames per second at a clock rate of 100 MHz.

Keywords Scale-invariant feature transform (SIFT) · Field Programmable Gate Array (FPGA) · Parallel architecture · Pipeline architecture

1 Introduction

The scale invariant feature transform (SIFT) algorithm is one of the most popular feature detectors reported in the literature [1]. It consists of two major stages of computation: the feature or keypoint (KP) extraction stage and the descriptor generation stage. While the first one requires most of the workload of the whole algorithm, the second one requires complex arithmetic operations. The complexity of the algorithm has led to different architectures which have progressively reached real-time operation. A common characteristic to all these implementations is the use of FPGAs, due to the inherent parallel nature of the algorithm. Two hardware/software co-designs using embedded FPGA processors are presented in [2, 3]. These implementations are able to generate descriptors for QVGA and VGA images, respectively, at 30 frames per second (fps), a rate which is usually considered as real-time in computer vision algorithms for video processing. In [4] an all-hardware FPGA implementations for VGA images, achieving 30 fps, is described. In [5, 6], two CMOS implementations are proposed. They also reach 30 fps for VGA and HD1080 images, respectively. Finally, in [7–9], different hardware FPGA implementations able to generate faster descriptors, above 50 fps, are described. Given its structure, composed by successive serial steps, the SIFT algorithm is suitable to be implemented exploiting pipeline architectures, easily synthesizable in FPGAs.

In this paper, an alternative real time hardware implementation of the SIFT algorithm is presented. It widely exploits pipeline implementations both in the keypoint extraction and in the descriptor generation stages to achieve processing rates faster that real time for VGA images and 1% of keypoints.

The rest of the paper is organized as follows. In Sect. 2 an overview of the SIFT algorithm is presented. Section 3 describes the proposed real time hardware architecture. In Sect. 4, the results obtained from the synthesis are shown. Finally, conclusions are drawn in Sect. 5.

✉ Ginés Doménech-Asensi
gines.domenech@upct.es

1 Dpto. de Electrónica y Tecnología de Computadoras, Universidad Politécnica de Cartagena, Cartagena, Spain

2 Centro de Inv. en Tecnoloxías da Información (CITIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain

3 Departamento de Integración, Centro Universitario de la Defensa CUD-UPCT, Santiago de la Ribera, Murcia, Spain

Published online: 05 May 2018

2 Overview of the SIFT algorithm

As it has been mentioned, the SIFT algorithm comprises two major stages. The first stage of the algorithm is devoted to extract the KPs from the image, as it is summarized in Fig. 1.

An input image G_0 is successively convolved with a Gaussian function K_i of variance σ_i^2, to create a so called Gaussian Scale space (GSS). Then, each pair of adjacent Gaussian images G_{i+1} and G_i are subtracted to obtain a Difference-of-Gaussians (DoG) space (Fig. 1b). Once the GSS and the DoG spaces have been created, every pixel in these spaces is identified by its x,y coordinates and as well as its σ.

The next step is the identification of extrema points in the DoG space. This is achieved comparing each pixel of a given DoG with its 26 neighbors, 8 in the same DoG, 9 in the upper one and 9 in the lower one (Fig. 1c). If the pixel value is the maximum or the minimum value of all these pixel values, then this pixel is considered as a KP candidate. As shown in Fig. 1, for a Gaussian pyramid with six Gaussian images, there are five DoGs and the extrema identification can be done over DoGs 1–3 (comparing each one with their respectives adjacent DoGs). Then, this process is repeated for the next octave, which is obtained using as input image a downsampled image of Gaussian 4. For an initial image with a resolution of 640×480 pixels, the second octave deals with 320×240 pixel images.

The KP extraction stage requires two more processes. First, some of the KP candidates will be points with low contrast with respect to its neighbors, which are discarded. Second, a subpixel refinement is applied to determine the exact $X=(x, y, \sigma)$ coordinates of the remaining KPs. This is performed by means of a Taylor expansion of the scale-space function, DoG(X), so that the exact extremum position with respect to the original one is given by:

$$\hat{X} = -\frac{\partial^2 \mathrm{DoG}^{-1}}{\mathrm{d}X^2}\frac{\partial \mathrm{DoG}}{\partial X} \tag{1}$$

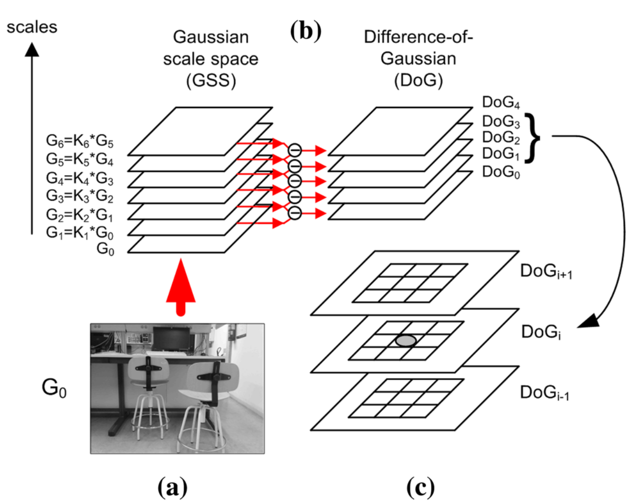


Fig. 1 Key point extraction stage in the SIFT algorithm: **a** original image, **b** creation of the GSS and DoG spaces, **c** detection of extrema points

The second stage of the algorithm is the descriptor generation, which consist of several processes. First, the principal orientation of the keypoint is calculated. This calculation employs the pixel values of a square area with a given width centered at the KP. The typical size of this area, called neighborhood area, is 16×16 pixels. The gradient magnitude $m\ (x, y)$ and orientation $\theta\ (x, y)$ of each one of these pixels are calculated using (2) to (4), where $L\ (x, y)$ is the intensity level of pixel (x, y).

$$\begin{aligned} \Delta x &= L(x+1, y) - L(x-1, y) \\ \Delta y &= L(x, y+1) - L(x, y-1) \end{aligned} \tag{2}$$

$$m(x, y) = \sqrt{\Delta x^2 + \Delta y^2} \tag{3}$$

$$\theta(x, y) = \tan^{-1}\left(\frac{\Delta y}{\Delta x}\right) \tag{4}$$

The gradient magnitudes of these pixels are accumulated in a histogram, leading to the so-called gradient histogram of orientation. This histogram is composed by 36 elements (36 bins), each 1 of them corresponding with an angular interval of 10°. Each value accumulated in the histogram is multiplied by a smoothing coefficient which depends on the σ value. The main orientation of the KP corresponds to the bin of the histogram with the highest value. The elements of the histogram whose values exceed the 80% of the peak value create new KPs, with the same coordinates and new orientations.

Once the main orientation has been obtained, the features descriptor vector is built. As shown in Fig. 2 the descriptor vector is an array of values constructed from the pixels around de feature point and is used to identify the keypoint. To generate the descriptor, first the 16×16 keypoint neighborhood is divided into 4×4 square subregions (Fig. 2a). Then for each one of these subregions, its principal orientation is calculated as it was done previously for the KP main orientation but rotating the image according to the keypoint main orientation. 8-bin histograms are used to compute the principal orientation of each subregion (Fig. 2.b). Finally, the 16 8-bin histograms are linked to obtain a 128-element vector (Fig. 2c). Once the feature descriptor vector is built, it is normalized twice to reduce the effects produced by changes in the illumination.

3 Proposed hardware architecture

Figure 3 shows an outline of the proposed system architecture for a single octave operation. Following the structure of the algorithm, it consists of two main parts working in

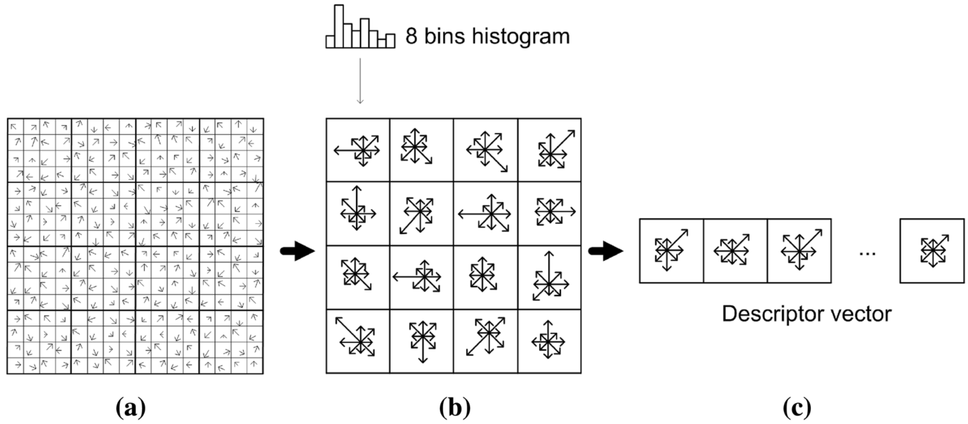


Fig. 2 Descriptor generation procedure

parallel: a keypoint extraction module and a descriptor generation module. Both of them are linked through the KP memory, where the first module writes the addresses of the extracted keypoints. The KP memory contains two symmetric memories with a ping-pong operation, to store the keypoints extracted, respectively, from odd and even images in a real time operation.

The descriptor generation module reads the KP addresses from the KP memory once the keypoint extractions has finished. For each KP, its neighborhood is read from the image memory and then processed to build the descriptor. Since the descriptor generation process is independent from the keypoint extraction, both processes run pipelined. So, when the descriptors of all the KPs of an odd image are being built, an even image can be simultaneously processed to extract its keypoints. These operations will be further detailed in next paragraphs.

3.1 Keypoint extraction

The keypoint extraction module has been synthesized to evaluate an image pixel per clock cycle. The value of every new incoming pixel is simultaneously written in the 640×480 memory (Fig. 3), and passed to the KP extraction module. This evaluation gives as a result the coordinates of that pixel in the case it is a keypoint candidate. Thus, the GSS and the DoG spaces as well as the local extrema detection are performed simultaneously, exploiting the FPGA parallelism.

There are two main approaches to build the GSS space. As shown in Fig. 1, each Gaussian image G_i in the GSS is obtained from its precedent Gaussian $G_{i-1.}$ This implies an increase in the latency of the operations, because every Gaussian image requires that its precedent image is previously generated. To avoid this, all Gaussian images should be created from the original image. However, this could not be an optimal implementation. Let σ_1 and σ_2 be, respectively, the variances of filtering functions K_1 and K_2, then:

$$\sigma_{12}^2 = \sigma_1^2 + \sigma_2^2 \tag{5}$$

where σ_{12} is the variance of the filtering function required to obtain G_2 directly from G_0 and σ_1 and σ_2 are, respectively, the variances of the filtering functions employed to obtain G_1 from G_0 and G_2 from G_1, respectively (Fig. 1).

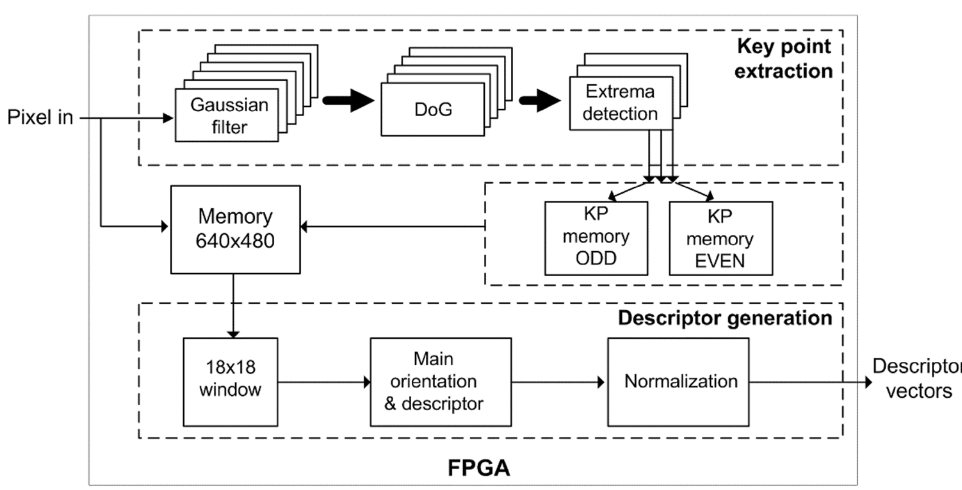


Fig. 3 Outline of the proposed system architecture to be synthesized on a FPGA

This means that the values of σ_1, and σ_2 are lower than the value of σ_{12}. Given that the size of the Gaussian kernel is proportional to the value of σ [1], the use of successive convolutions requires smaller kernel sizes than those required if direct convolutions were performed. This allows the use of fewer hardware registers and also fewer processing units. Table 1 shows a comparative of kernel sizes between both approaches. The proper values of σ_i, which vary from 1.6 for the first convolution to 5.08 for the last one, are those recommended by [1].

Table 1 Successive filtering vs direct filtering

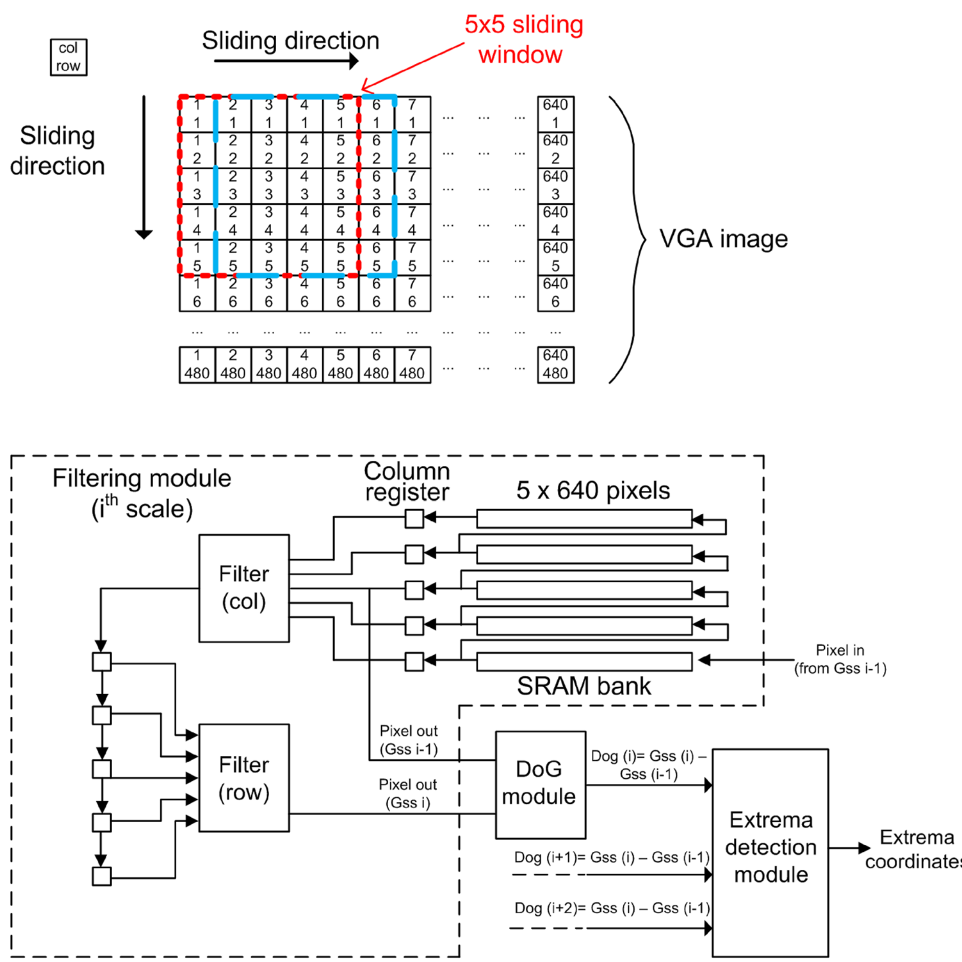
Scale	Direct		Successive	
	σ	Kernel size	σ	Kernel size
6	5.08	27×27	3.0900	15×15
5	4.03	21×21	2.4525	13×13
4	3.20	17×17	1.9466	11×11
3	2.54	13×13	1.5450	9×9
2	2.02	11×11	1.2263	7×7
1	1.6	9×9	1.6	9×9

As it can be observed in Table 1, the kernels are considerably smaller when successive filtering is applied. Although this approach implies a data dependency of upper scales from lower ones, the following lines detail how this drawback is overcomed.

The keypoint extraction stage has been synthesized using a sliding data window schema. Figure 4 shows the architecture proposed for this first stage of the algorithm using as example a 5×5 Gaussian filtering. This architecture builds the GSS and the DoG spaces and also detects the extrema points. It consists on a filtering module composed by a 5-row SRAM bank which feeds two cascaded filtering stages, a module to compute the difference of Gaussians and a module to obtain the extrema points. In the figure, a 5×5 smoothing is performed as example, but in the real architecture, both the size of the filters and the number of rows in the SRAM bank is specified by the kernel size in Table 1. In the implementation proposed in this work, in which the number of scales is six, there are six filtering modules, five DoG modules and three extrema detection modules.

The input to the SRAM bank are the pixels coming from the previous Gaussian, or from the camera, in the case of the first scale. Each row feeds the adjacent upper row as well

Fig. 4 Structure of the proposed architecture for the first step of the algorithm



as a static register in the column register. So, the Gaussian operation is performed over the window defined by pixels (1, 1) in the upper left corner and (5, 5) in the lower right corner. The second Gaussian is applied to the window defined by the pair (2, 1) and (6, 5) and so on.

The implementation of the Gaussian filters has been done exploiting its separability property [10], which allows to decompose a two-dimensional NxN Gaussian filtering into two cascaded one-dimensional Gaussian filters. The use of this property reduces the number of multipliers from N^2 to $2N$.

So, for the filtering operation, the separability of Gaussians is implemented by two filters (row and column). So, the output of the column register is taken to a filter which performs the column filtering and its output then feeds a FIFO register which feeds in parallel a second filter which performs the row filtering.

The difference of Gaussians is performed in the DoG module. The two inputs are, on the one side the output pixel of the filtering operation (Gss(i)) and on the other, the input pixel from the previous stage (Gss(i-1)) obtained from an intermediate output of the SRAM filters. This allows both pixels to be in phase.

Finally, the outputs from the adjacent DoGs are compared to detect extrema points. When a pixel is identified as extrema, its coordinates are stored in the KP memory (Fig. 3). This architecture allows the evaluation of a pixel every clock cycle.

Figure 5 shows the timing diagram of a pipelined structure which exploits parallelism between the successive filtering stages. The timing diagram details both the KP extraction and the descriptor generation stages. Focusing on the first stage, each consecutive filtering operation, from Gauss 2 to Gauss 6, begins with an increasing delay, because the respective kernel sizes are bigger (Table 1). This means that more lines need to be generated in a given Gaussian image G_i before the next Gaussian operation starts. The first filtering operation (Gauss1) performed on the left image requires 3219 cycles to output the first valid pixel, and then 640×480 additional cycles to complete the image filtering and obtain G_1. The pipelined structure allows the first DoG operation (DoG_1) to begin when the first pixel of G_2 is available, at cycle 5793. The following DoG operations begin successively and, when the first pixel of the DoG_2 is available, the comparison between pixels in DoG_1 and its neighbors in DoG_0 and DoG_2 begins, to detect the extrema points (cycle 12,864). The same process is repeated to compare DoG_1, DoG_2 and DoG_3 and to compare DoG_2, DoG_3 and DoG_4. The sixth Gaussian operation starts after 22,499 cycles and the whole KP extraction process ends after 329,699 clock cycles.

The processing of the second, or even, image starts 1922 cycles after the first Gaussian is completed. This is, before the whole KP extraction of the first image ends. Compared to the alternative of using absolute filtering, the recursive filtering implies a certain latency in the pipeline structure. However, the throughput achieved is not affected.

During the extrema detection process, some KP candidates are discarded because they have low contrast with respect to its neighbors. So, a KP candidate is considered a real KP if it is a local extrema above a given threshold value. According to [1], its recommended value is 0.03.

3.2 Descriptor generation

Descriptor generation requires higher level computations that the keypoint extraction stage. To achieve real time processing, simplifications on the algorithm are usual to

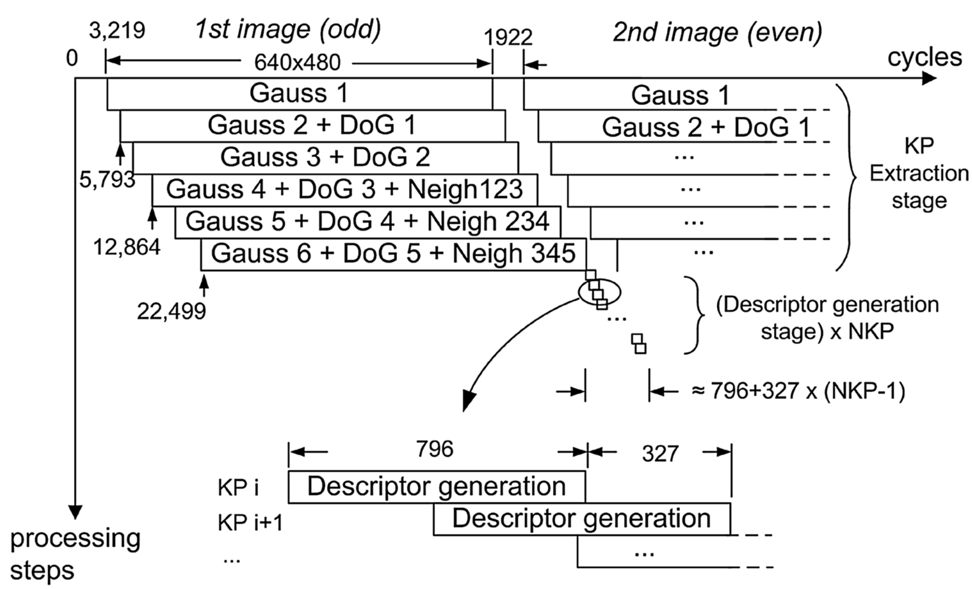


Fig. 5 Timing diagram of the proposed pipelined architecture for SIFT algorithm

optimize the hardware resources and the computation time. Two simplifications are presented in [11]. On the one hand, the smoothing coefficient that multiplies the magnitude gradient of each orientation before adding it to its corresponding bin in the histogram is replaced with a constant of unity value. On the other, it is assumed that each feature point has two main orientations at most. In [3] another simplification is proposed. It replaces Euclidean distance (3) by Manhattan distance (6) to calculate the gradient magnitude. Moreover, it proposes an orientation histogram with 8 bins, instead of the 36 bins histogram, of the original SIFT.

$$m(x,y) = |X| + |Y| \tag{6}$$

Equation (6) is easier to synthesize than (3), which is usually a very computationally expensive operation on FPGA solutions for the SIFT algorithm [2, 3]. The Shifting-Based Orientation Calculation introduced in [11] leads to much less area consumption, and it is the implementation adopted in this work. In the SIFT algorithm the orientation angle selects the bin where the gradient magnitude will be added, in such a way that a pixel magnitude must be added to a bin B_i if its orientation is in the interval $[\theta_i, \theta_{i+1})$, satisfying:

$$X \tan\left(\theta_i\right) \leqslant Y \leqslant X \tan\left(\theta_{i+1}\right) \tag{7}$$

It should be noted that the histogram comprises 36 bins, and that thanks to the symmetry of the trigonometric functions, the assignment of pixels to the bins located in the second, third and fourth quadrants is obtained from the bin assignment in the first quadrant, taking into account the sign of X and Y. The implementation proposed in this paper uses 8-bit words to represent the precomputed values of $\tan(\theta_i)$. The highest loss of accuracy due to this discretization is around 1°. However, if the following $\cot(\theta_i)$ function is employed:

$$Y \cot\left(\theta_i\right) \geqslant X \geqslant Y \cot\left(\theta_{i+1}\right) \tag{8}$$

the inaccuracy is cut down to 0.2°. The proposed implementation uses the most convenient function, either $\tan(\theta_i)$ or $\cot(\theta_i)$ for the main orientation vector calculation. As proved in [11], all these simplifications combined lead to a loss of accuracy of 1.1% in the detection of main orientation compared with the original algorithm.

Figure 6 shown the overview of the architecture proposed in this work to implement the descriptor generation step. The architecture is an evolution of the implementation described in [12] to speed up this step. The architecture is divided in two modules: the main orientation and descriptor generation module and the normalization module. The descriptor has 128 12-bit elements, instead of the 27 element vector used in [12]. Equations (6) and (8) are used to, respectively, compute the gradient module and orientation, but keeping a gradient histogram of 36 bins as in the original SIFT algorithm. This implementation provides a good tradeoff between accuracy and usage of resources. As shown in Fig. 5, the descriptor generation operation starts once the keypoint extraction has finished and it runs in parallel with the next image keypoint extraction process.

The keypoint neighborhood window size is 16 × 16 pixels and is centered in the keypoint coordinates. An 18 × 18 window is initially required to obtain the gradient of the pixels placed on the edges of the 16 × 16 window. This topology allows to simultaneously calculate both the keypoint principal orientation and each one of the 16 subregions orientation histograms.

The procedure is the following. Given the coordinates of a keypoint, the 18 × 18 neighborhood window is loaded from the image memory into an internal FPGA memory. The loading of this window takes 324 clock cycles (Fig. 7). Then,

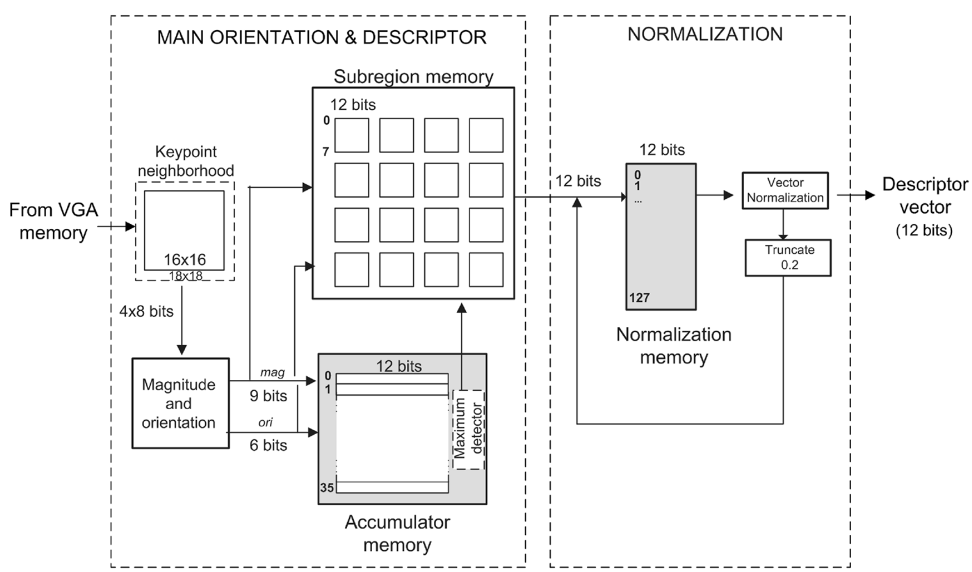


Fig. 6 Structure of the proposed architecture for the second step of the algorithm

for each pixel $x_{i,j}$ of the 16×16 neighborhood, its magnitude and orientation are calculated in a single clock cycle using (2), and (6) to (8). Four 8 bit buses drive simultaneously the values of pixels $x_{i,j+1}$, $x_{i,j-1}$, $x_{i+1,j}$ and $x_{i-1,j}$ to the magnitude and orientation module.

The magnitude and orientation values are accumulated in their respective orientation histograms in the pixel subregion memory and in the accumulator memory. Thus, each subregion memory requires also an accumulator of 8 words of 12 bits size inside the subregion memory block. At the end of this stage, which has required 354 clock cycles (Fig. 7), a combinational circuit identifies the maximum value in the accumulator memory, which is the keypoint principal orientation. This value is transferred to the subregion memory block and the 16 histograms are then rotated according to the principal orientation value. This rotation takes 8 clock cycles. Simultaneously, each one of the sixteen 36-bin histograms of the *subregion memory* is grouped to obtain 8-bin histograms. Finally, the sixteen 8-bin histograms are linked together to obtain the 16×8 dimension feature descriptor vector which is transferred to the normalization module. This transfer ends at cycle 492.

Once the descriptor has been generated, the next step consists on the normalization of the feature descriptor vector. Given a descriptor vector $V = (v_1, v_2, \ldots, v_{128})$, its normalized value is obtained as:

$$\overline{V} = \frac{V}{|V|} = V \frac{1}{\sqrt{\sum_{k=1}^{128} v_k^2}} \tag{9}$$

Figure 8 shows a more detailed structure of the normalization module. Two normalizations are carried out by the normalization factor and the normalization operator structures. The first structure computes the sum of the 128 squared terms of the descriptor vector. The second one multiplies the descriptor vector by the inverse of the square root of that value. To achieve a fast operation, the incoming descriptor vector is simultaneously stored in the normalization memory and processed in the normalization factor module. After the first normalizations of the feature descriptor vector, the values exceeding 0.2 units are truncated to this value according to [1]. This truncation is done to reduce the influence of large gradient magnitudes. The squared root value is available at cycle 492, and the inverse value is obtained using a CORDIC algorithm at cycle 513. The normalization takes 128 cycles, this is, the length of the descriptor vector, where each component of the vector is multiplied by the inverse value previously calculated. Simultaneously, the truncation to 0.2 is performed, as well as the calculation of the sum of squared terms, which ends at cycle 643. Then, the second normalization is carried out. Again, the squared root and inverse are calculated, and the normalization itself finishes at cycle 796. Thus, this architecture requires 796 cycles to produce a feature descriptor vector. However, due to the hardware parallelism, the calculation of a new keypoint descriptor vector starts every 327 cycles, as shown in Fig. 7. This means that, for at clock frequency of 100 MHz, 3072 KPs (1% of VGA image size) can be processed in 10.045 ms, at a frame rate of 99.5 fps.

4 Results

The proposed hardware architecture has been prototyped on a Virtex 5 FPGA. The Gaussian pyramid was built using the values of σ and kernel size shown in Table 1 for successive filtering operations.

A fixed point data format has been used in this implementation. It uses 8 bits for the integer part. The number of

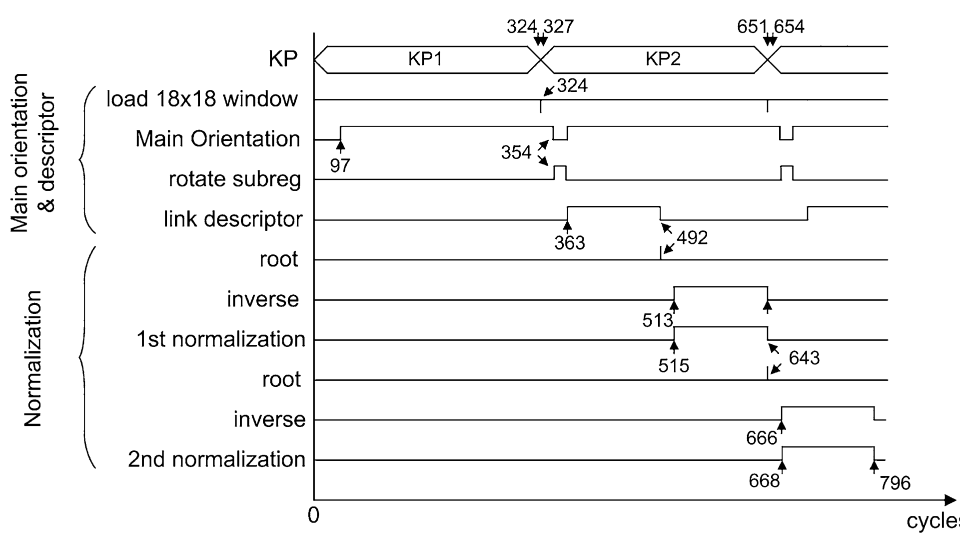


Fig. 7 Timing diagram of the descriptor generation stage. The calculation of the second keypoint descriptor starts after 327 cycles

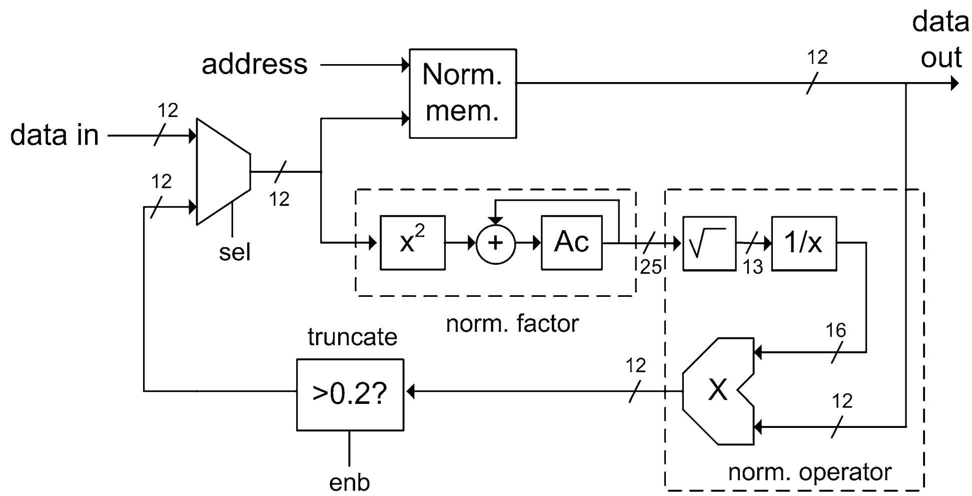


Fig. 8 Architecture of the normalization module

fractional bits has been selected according to the following criteria. First, according to [1], the recommended value for the threshold value employed in the extrema detection is 0.03. This means at least a 5 bit precision is needed in the fractional part. Second, Fig. 9 shows the percentage of keypoints obtained from VHDL implementation that match the keypoints obtained using the Vedaldi code [13], for different number of fractional bits for different images obtained from the Oxford affine covariant regions dataset [14].

As expected, the accuracy increases with the number of bits and reaches values above 80% for 7 and 8 fractional bits. Given that the difference in the number of resources is not significant when using 7 or 8 bits, an 8 fractional bit implementation has been finally used.

While Fig. 9 shows the performance of the keypoint extraction module, the performance of the proposed architecture for the whole system has been tested using pairs of images from the Oxford affine covariant regions dataset. Figures 10, 11 and 12 compare the matching results obtained using the Vedaldi code and the VHDL implementation for three different pairs of images.

In Fig. 10 the number of matches between the original image and the rotated one using the Vedaldi code was 287 (Fig. 10a). The number of matches between these same images using the synthesized VHDL architecture was 62 (Fig. 10b). This represent a 21% of the matches obtained with the Vedaldi code. For the pair of images shown in Fig. 11 the number of matches was 573 and 195 using the Vedaldi code and the VHDL implementation, respectively. In this case, the percentage of matches obtained with the VHDL implementation is 34%. Finally, 332 and 66 matches (19%) were obtained, respectively, for the images shown in Fig. 12 using the Vedaldi code and the VHDL implementation. These results show that the proposed architecture is able to perform the matching operation between a pair of images. Although the number of matched KPs is strongly decreased due to the simplifications used in the implementation, including the discretization due to the use of fixed point format, there is still a considerable number of properly matched KPs. The accuracy of the implementation has been defined as the ratio between the number of correct matches and the number of total matches between two images. For the images shown in Figs. 10, 11 and 12, the accuracy was 0.96, 0.99 and 1, respectively.

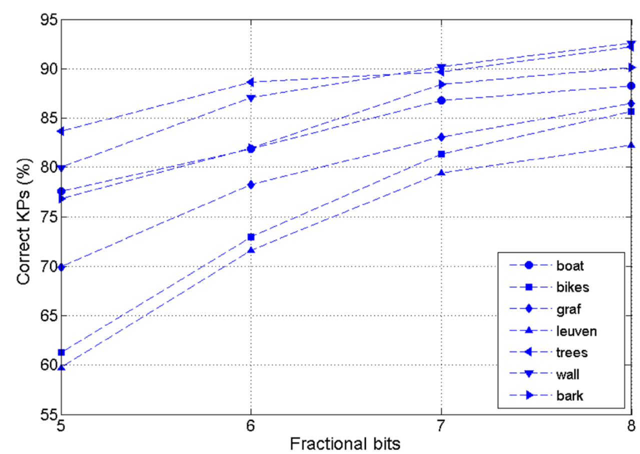
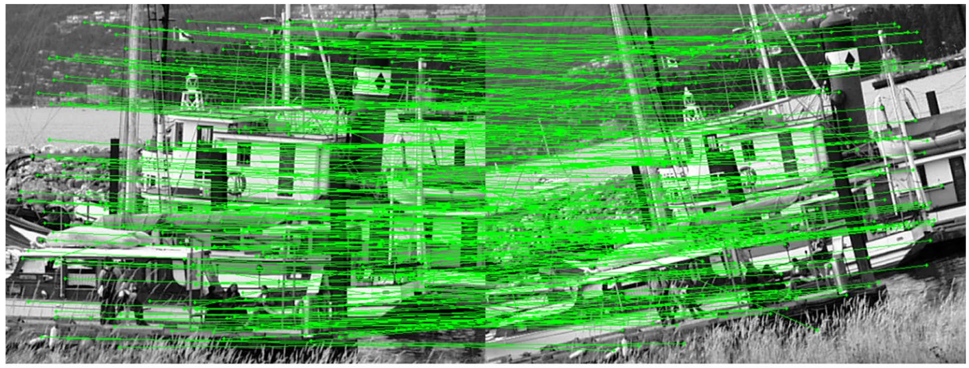


Fig. 9 Percentage of keypoints obtained from VHDL simulations which fit keypoints obtained using Vedaldi code. A KP obtained using the VHDL simulation is correct if its coordinates are the same than those obtained using the Vedaldi code

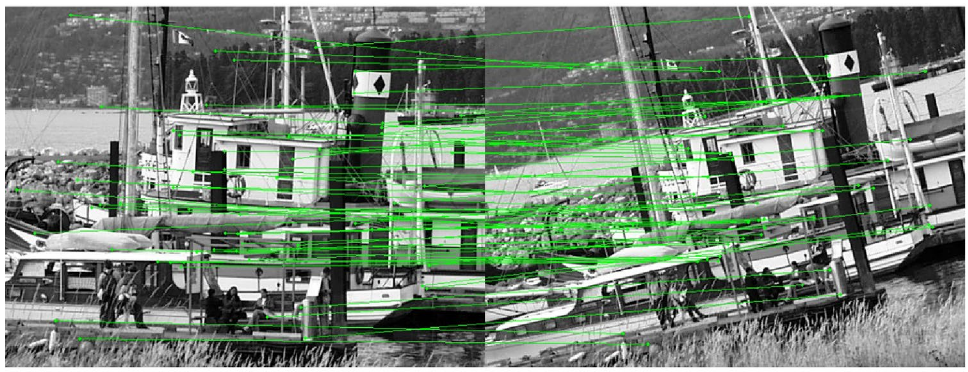
Table 2 details the overall performance of the proposed implementation compared with other implementations of the SIFT algorithm for image sizes of 640×480 pixels.

The proposed architecture yields a latency of 796 cycles and a throughput of 327 cycles. (Figs. 5, 7). So, to obtain 3072 descriptors (1% of VGA size) the number of cycles required is $796 + (3071 \times 327) = 1{,}005{,}013$ (10.05 ms at 100 MHz). Given that the KP descriptor generation stage

Fig. 10 Matching results for the rotation transform (boat), **a** using the Vedaldi code **b** VHDL implementation

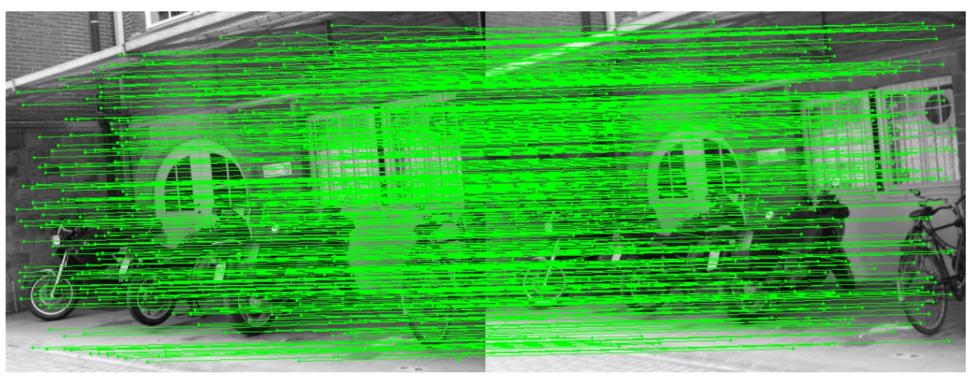


(a)



(b)

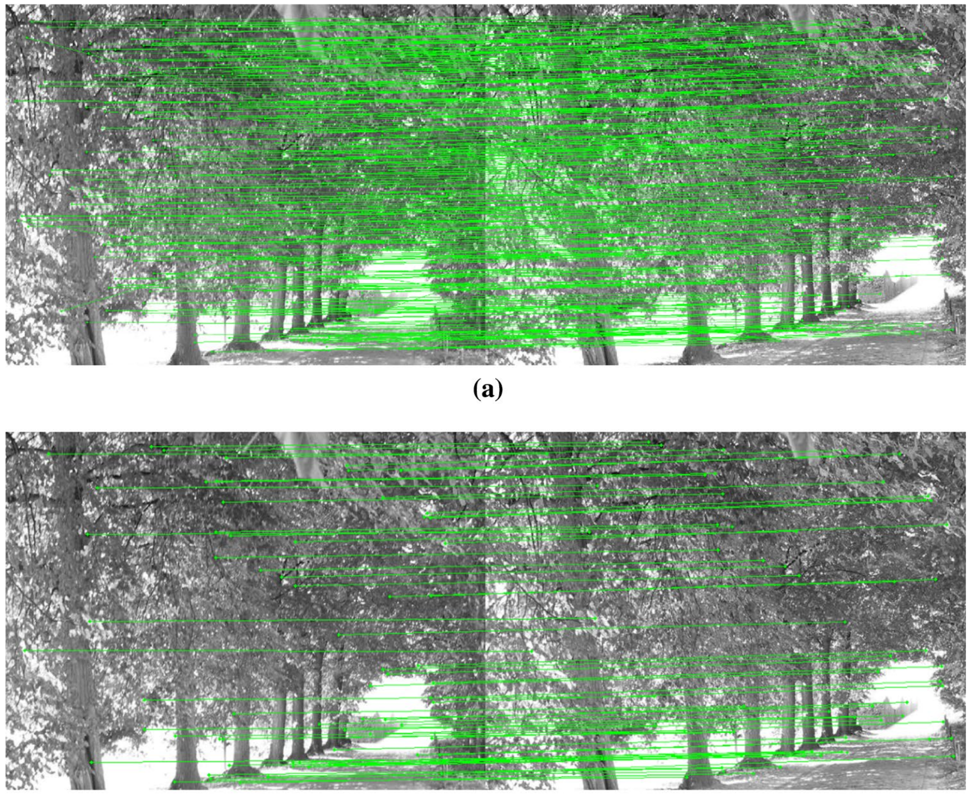
Fig. 11 Matching results for the blur transform (bikes) **a** Using the Vedaldi code **b** VHDL implementation



(a)

(b)

Fig. 12 Matching results for the blur transform (trees) **a** Using the Vedaldi code **b** VHDL implementation



(a)

(b)

requires more time than the KP extraction one, this is the value which limits the implementation speed. Thus, the throughput of this architecture is 99 fps for a single octave and six scales. This value is faster and the number of keypoints greater that that reported for similar image size implementations. Compared to other proposals in the literature, the number of DPS in this work is above the average, while the number of registers and LUTs are below it. So, the average usage of logic resources is similar to other proposals. The RAM required in the proposed architecture is larger than that used in other implementations. This is due to the pipelined architecture, which requires more storage resources. Regarding the performance, works reported in [3, 5] implement the algorithm with two octaves but only four scales, and the last one, do not implement the descriptor vector normalization stage. Works described in [7] achieves a frame rate of 56 fps working at half the frequency that the proposed architecture, but it implements only the descriptor generation module and requires 258 DSP modules. The implementation described in [4] works also at 50 MHz, but its frame rate

Table 2 Performance and use of FPGA resources of the proposed architecture vs other works in the literature for image sizes of 640×480 pixels

Technology	[3] Virtex 5+Micro-Blaze	[4] Virtex 5	[5] TSMC 0.18 µm CMOS	[7] Cyclone II[a]	[8] Virtex 6[a]	This work Virtex 5
Number of KPs	NA	1270	890	NA	2200	3072
Frame rate (fps)	32	30	30	56	60	99
Registers	23,247	7974	–	23,247	29,453	12,065
LUTs	32,592	16,138	–	32,592	64,701	14,335
RAM (MB)	0.67	0.576	5.73	0.87	3.84	8.2
DSP	97	53	–	258	107	150
Clock (MHz)	100	50	100	50	100	100

NA not available

[a]The implementation does not include KP extraction, only descriptor generation

and number of KPs are about one-third of these achieved in this work. The architecture described in [8] uses a reduced descriptor vector of 72 elements instead of 128, to generate 2200 frames at 60 fps and it does not implement the keypoint extraction stage. For the synthesis proposed in this work, processing of two more octaves would require an increase of 75% of cycles in the first stage if no additional parallelization is done. This number of cycles is still smaller than those required for the descriptor generation stage. So, the overall processing speed would not be affected.

5 Conclusions

In this paper, an optimized all-hardware implementation of the SIFT algorithm has been described. The architecture has widely used pipeline structures to speed up the image processing and it achieves more than 3× real time processing speed for VGA images. At this speed, the architecture is able to generate 3072 keypoint descriptor vectors (1% of the image size) at a frame rate of 99 fps. Tests performed over known set of images show a very good performance of the proposed architecture.

Acknowledgements This work has been partially funded by Spanish government project TEC2015-66878-C3-2-R (MINECO/FEDER, UE).

References

1. Lowe, D.: Distintive image features from scale-invariant keypoints. Int. J. Compt. Vis. **60**, 91–110 (2004)
2. Bonato, V., Marques, E., Constantinides, A.: A parallel hardware architecture for scale and rotation invariant feature detection. IEEE Trans. Circ. Syst. Video Technol. **18**, 1703–1712 (2008)
3. Yao, L., Feng, H., Zhu, Y., Jiang, Z., Zhao, D., Feng, W.: An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher. Proc Int. Conf. Field-Program. Technol. **2009**, 30–37, (2009)
4. Qasaimeh, M., Sagahyroon, A., Shanableh, T.: FPGA-based parallel hardware architecture for real-time image classification. IEEE Trans. Comput. Imaging **1**, 56–70 (2015)
5. Huang, F.C., Huang, S.Y., Ker, J.W., Chen, Y.C.: High-performance SIFT Hardware accelerator for real-time image feature extraction. IEEE Trans. Circ. Syst. Video Technol. **22**, 340–351 (2012)
6. Chiu, L.C., Chang, T.S., Chen, J.Y., Chang, N.Y.C.: Fast SIFT design for real-time visual feature extraction. IEEE Trans. Image Process. **22**, 3158–3167 (2013)
7. Mizuno, K., Noguchi, H., He, G., Terachi, Y., Kamino, T., Kawaguchi, H., Yoshimoto, M.: Fast and low-memory-bandwidth architecture of SIFT descriptor generation with scalability on speed and accuracy for VGA video. Proc. Int. Conf. Field Program. Logic Appl. **2010**, 608–611 (2010)
8. Deng, W., Zhu, Y., Feng, H., Jiang, Z.: An efficient hardware architecture of the optimised SIFT descriptor generation. Proc Int. Conf. Field Program. Logic Appl. **2012**, 345–352 (2012)
9. Jiang, J., Li, X., Zhang, G.: SIFT hardware implementation for real-time image feature extraction. IEEE Trans. Circuits Syst. Video Technol. **24**, 1209–1220 (2014)
10. Jain, R., Kasturi, R., Schunck, B.G.: Machine Vision. McGraw-Hill, Inc., New York (1995)
11. Qiu, J., Lu, Y., Huang, T., Ikenaga, T.: An FPGA-based real-time hardware accelerator for orientation calculation part in SIFT. Proc Fifth Int. Conf. Intell. Inf. Hiding Multi. Signal Process. **2009**, 1334–1337 (2009)
12. Doménech-Asensi, G., Garrigós, J., López, P., Brea, V., Cabello, D.: Real time architectures for the Scale Invariant Feature Transform algorithm. Proceedings of the 15th International Workshop on Cellular Nanoscale Networks and their Applications, pp. 23–25 (2016)
13. http://vision.ucla.edu/~vedaldi/code/sift.html. Accessed 1 Feb 2017
14. http://www.robots.ox.ac.uk/~vgg/data/data-aff.html. Accessed 4 May 2017

Ginés Doménech-Asensi was born in Barcelona, Spain, in 1972. He received the BSc and MSc degrees in Industrial Engineering from the Universidad de Murcia, Murcia, Spain, in 1993 and 1996, respectively, and the PhD degree from the Universidad Politécnica de Cartagena, Cartagena, Spain, in 2002. Since 1999, he has been an Associate Professor with the Departamento de Electrónica, Tecnología de Computadoras y Proyectos at Universidad Politécnica de Cartagena, where he is currently Associate Professor. His current research is within the field of CMOS mixed signal electronic circuits design and custom architectures for computer vision.

Juan Zapata-Pérez was born in Cartagena, Spain, in 1966. He obtained the MSc degree in Industrial Engineering in 1994 from Universidad de Murcia, and the PhD degree in 2002 from the Universidad Politecnica de Cartagena. He is currently Associate Professor of Electronics at the Department of Electronics, Computer Technology and Projects at the Universidad Politecnica de Cartagena. He has a wide teaching experience in subjects of electronics and computer technology. He is co-author of technical papers and contributions in the fields of electronic design, embedded systems (Wireless Sensor Networks) for Ambient Assisted Living, biomedical and industrial pattern recognition, and bio-inspired systems.

Ramón Ruiz-Merino was born in Jaén (Spain), in 1958. He received the MSc in Physics from the University of Granada (Spain) in 1980 and the PhD degree from the University of Santiago de Compostela (Spain) in 1986. Since 1993, he is a Professor of Electronics, currently with the department of Electronics and Computer Technology at the Technical University of Cartagena. Professor Ruiz is co-author of more than a hundred and thirty scientific papers and contributions in the fields of Biomedical Engineering, Electronic Design and Computer Architecture, having been advisor of seven PhD theses and leader of several research projects. In the last fifteen years, he has also held different university management positions. He is a senior member of the Institute of Electrical and Electronic Engineers (IEEE) and a member of the European Society for Fuzzy Logic and Technology (EUSFLAT) and the Spanish Society of Biomedical Engineering (SEIB).

José Alejandro López-Alcantud was born in Cartagena (Spain) in 1970. He received the BSc and MSc degrees in Industrial Engineering form the "Universidad de Murcia" (Spain) in 1991 and 1994, respectively. In 1995 he joined the "University of Murcia" as Assistant Lecturer. Currently, he is Lecturer of Electronics with the "Departmento de Electrónica, Tecnología de Computadoras y Proyectos" at the "Universidad Politécnica de Cartagena". Not only has he a wide teaching experience in Electronics, but also he has been involved with researching topics such as Neural Networks and Artificial Vision. Currently, he is focused in the CMOS mixed signal electronic circuit design.

José Ángel Díaz-Madrid was born in Cartagena, Spain, in 1975. He received a BSc degree in Electrical Engineering from the Universidad de Murcia, Spain, in 1997, and an MSc degree in Automation and Industrial Electronic Engineering from the Universidad Politécnica de Cartagena, Spain, in 2000. From 2001 to 2009, he joined the Department of Electrical Engineering, Fraunhofer Institute IIS, as a mixed signal analog designer. From 2009 to 2010, he has been with the Departamento de Electrónica, Tecnología de Computadoras y Proyectos, where he was a part-time assistant professor and from 2011 he has been part-time assistant professor in the Centro Universitario de la Defensa in Universidad Politécnica de Cartagena. Currently, he is finishing his PhD in Low Power Pipeline ADC. His current research interests include mixed signal design, analog to digital converter (ADC) and low power design.

Víctor Manuel Brea received the Ph.D. degree in physics in 2003. He is currently an Associate Professor with the Centro Singular de Investigación en Tecnoloxías da Información, University of Santiago de Compostela, Spain. His main research interests lie in the design of efficient architectures and CMOS solutions for computer vision, especially in early vision, as well as micro-energy harvesting.

Paula López (M'02) received the Ph.D. degree in physics in 2003. She is currently an Associate Professor with the University of Santiago de Compostela. She is author or co-author of more than 50 research papers. Her current research interests include the design of mixed signal integrated circuits for image processing applications and the physical modeling of electronic devices, particularly CMOS image sensors, and the translation of these models into hardware description languages.