A Hardware Architecture for SIFT Candidate Keypoints Detection

Leonardo Chang and José Hernández-Palancar

Advanced Technologies Application Center
7^{th} Ave. ♯ 21812 % 218 y 222, Siboney, Playa, C.P. 12200, Havana City, Cuba
{lchang,jpalancar}@cenatav.co.cu

Abstract. This paper proposes a parallel hardware architecture for the scale-space extrema detection part of the SIFT (Scale Invariant Feature Transform) method. The implementation of this architecture on a FPGA (Field Programmable Gate Array) and its reliability tests are also presented. The obtained features are very similar to Lowe's. The system is able to detect scale-space extrema on a 320 × 240 image in 3 ms, what represents a speed up of 250x compared to a software version of the method.

Keywords: FPGA, SIFT hardware architecture, parallel SIFT.

1 Introduction

In the last few years the use of local features has become very popular due to their promising performance. They have exhibited considerable results in a variety of applications such as object recognition, image retrieval, robot localization, panorama stitching, face recognition, etc.

Almost certainly the most popular and widely used local approach is the SIFT (Scale Invariant Feature Transform) method [5] proposed by Lowe. The features extracted by SIFT are reasonably invariant to image scale, rotation, changes in illumination, image noise, and small changes in viewpoint. This method has been used effectively in all the above mentioned application fields. Lowe divided his method in four major computation stages: i) scale-space extrema detection, ii) keypoint localization, iii) orientation assignment, and iv) keypoint descriptor.

In its first stage, in order to detect scale invariant interest points, Lowe proposed to use scale-space extrema in the Difference-of-Gaussian (DoG) function convolved with the image, which can be computed from the difference of adjacent scale images. To obtain the DoG images several convolutions with Gaussians are produced. This represents a significative computational cost (about 30% of the whole algorithm), which makes it an expensive procedure. Some work has already been done to increase SIFT performance, by using a GPU (Graphic Processor Unit) in PCs [8] or by simplifying the algorithm through approximation [3][4].

E. Bayro-Corrochano and J.-O. Eklundh (Eds.): CIARP 2009, LNCS 5856, pp. 95–102, 2009.

The use of FPGAs (Field Programmable Gate Arrays) is a solution that a large number of researchers has successfully applied to speed up computing applications. Deeper in the SIFT algorithm and specifically in the DoG calculation, it turns out that this task has a great degree of parallelism, making it ideal for implementation in a FPGA. It is mentioned in [7] a system that implements SIFT to aid robotic navigation, which takes 60 ms for a 640 × 480 image. Nevertheless, architecture details or results discussion are not presented. In [6] the most expensive parts of the SIFT algorithm are implemented (i.e. Gaussian Pyramid and Sobel) and some architecture details and algorithm adequacies for hardware are given. This system can run at 60 fps but the image size is not mentioned, neither FPGA area allocation. Another system able to detect SIFT keypoints is presented in [2], which is capable to process 320 × 240 images in 0,8 ms. However, just a few information about the hardware architecture and none from the FPGA area usage are given. A complete implementation is demonstrated in [1], which requires 33 ms per 320 × 240 image.

This paper presents a parallel hardware architecture for one of the most intensive parts of the SIFT algorithm: the scale-space extrema detection. This architecture is implemented in a FPGA, where only 3 ms for scale-space extrema detection on a 320 × 240 sized image are required.

The organization of this paper is as follows: In Section 2 the target algorithm is explained. In Section 3 some issues of the scale-space extrema detection are discussed, which are later used by the architecture proposed in Section 4. In Section 5, implementation details and reliability tests for our system are presented. The work is concluded in Section 6.

2 Detection of Scale-Space Extrema in the SIFT Method

For a given image $I(x, y)$, the SIFT detector is constructed from its Gaussian scale-space, $L(x, y, \sigma)$, that is built from the convolution of $I(x, y)$ with a variable-scale Gaussian: $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$, where $G(x, y, \sigma)$ is a Gaussian kernel and $*$ is the convolution operator in x and y. The Gaussian scale space is created by generating a series of smoothed images at discrete values of σ. Thus the σ domain is quantised in logarithmic steps arranged in O octaves, where each octave is further subdivided in S sub-levels. The value of σ at a given octave o and sub-level s is given by: $\sigma(o, s) = \sigma_0 2^{o+s/S}, \quad o \in [0, ..., O-1], \quad s \in [0, ..., S-1]$, where σ_0 is the base scale level, e.g., $\sigma_0 = 1.6$. At each successive octave the data is spatially down-sampled by a factor of two.

To efficiently detect stable keypoint locations in scale space, Lowe proposed using scale-space extrema in the DoG scale-space, $D(x, y, \sigma)$, computed from the difference of adjacent scales: $D(x, y, \sigma(o, s)) = L(x, y, \sigma(o, s+1)) - L(x, y, \sigma(o, s))$.

In order to detect the local maxima and minima of $D(x, y, \sigma)$, each pixel in the DoG images is compared to its eight neighbors at the same image, plus the nine corresponding neighbors at adjacent scales. If the pixel is larger or smaller than all these neighbors, it is selected as a candidate keypoint.

3 The Proposed Parallel Detection of Scale-Space Extrema

The main motivation for the use of FPGAs over conventional processors is given by the need to achieve higher performance, better tradeoff cost-benefits and scalability of a system. This is possible thanks to the inherent parallelism in these devices, which by their physical characteristics, is able to keep all operations activated. Therefore, to achieve such profits and a significant speedup in the detection of scale-space extrema, is essential to exploit the parallelism of this algorithm. Nevertheless, there are other factors to consider in a FPGA design such as area and power requirements. Hence, this algorithm must be rewritten to take advantage of the parallel structure afforded by implementation in hardware, taking into account area and power requirements.

3.1 Exploiting Data Parallelism

Convolution is one of the most expensive operations that are used in image processing applications and particularly in the SIFT method. Then, it is an important issue to deal with.

If I is a two-dimensional image and g is a convolution mask of odd size $k \times k$, then the convolution of I and g is defined as:

$$f(x,y) = \sum_{-i}^{i}\sum_{-j}^{j} I(i,j)g(x-i,y-j), \text{where } i,j = \lfloor \frac{k}{2} \rfloor. \tag{1}$$

As can be seen in (1), for the calculation of $f(x_1, y_1)$ only a neighborhood in I of size $k \times k$ with center (x_1, y_1) is needed. Therefore, in 2D convolution a high potential for data parallelism is available, specifically of SPMD (Single Process, Multiple Data) type.

3.2 Exploiting Separability Property of the Gaussian Kernel

With the aim of reducing the number of arithmetic operations, the separability and the symmetry properties of the Gaussian are considered.

A 2D filter kernel is separable if it can be broken into two 1D signals: a vertical and a horizontal projection. The Gaussian kernel could be separated as follows:

$$G(x,y,\sigma) = h(x,\sigma) * v(y,\sigma),$$

where

$$h(x,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2}, \text{and } v(y,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-y^2/2\sigma^2}$$

Thus, the 2D convolution can be performed by first convolving with $h(x,\sigma)$ in the horizontal direction, and then convolving with $v(y,\sigma)$ in the vertical direction. 1D convolution, to compute a value of the output, requires k MAC operations. However, as is described in (1), 2D convolution in spatial domain requires k^2 MAC (multiplication and accumulation) operations. Therefore, the computational advantage of separable convolution versus nonseparable convolution is $k^2/2k$.

3.3 Octaves Processing Interleaving

As stated in Section 2, at each successive octave, the image size is downsampled by a factor of two by taking every second pixel in each row and column, i.e. $I_o(x, y) = I_{o-1}(2x, 2y)$. After downsampling by a factor of two, the total number of pixels is reduced by four. In hardware, to reduce the size of the data, its sample rate is reduced by the same factor. If at each successive octave the data size is reduced by four, the sample period τ of an octave o is given by

$$\tau(o) = \tau_0 4^o, \tag{2}$$

where τ_0 is the first octave sample period. Consequently, after subsampling, there is a large percentage of idle processing time $\hat{\imath}$ in respect of the first octave sample period, which is defined by $\hat{\imath} = \frac{\tau(o)-1}{\tau(o)}$.

This idle processing gap makes feasible the processing of the O octaves of a scale in a single convolution processor. This could be possible by interleaving the O convolution processes so that for all the octaves at a given time t the number of processed elements $p(o, t)$ satisfies that

$$p(o, t) = \left\lfloor \frac{t + \varepsilon(o)}{\tau(o)} \right\rfloor, \tag{3}$$

where $\varepsilon(o)$ is the delay of octave o in the interleaving line.

Here, for the O octaves interleaving is assumed that the first octave sample period is equal or greater than two clock cycles, if not, $O - 1$ octaves are interleaved and τ_0 would be the second octave sample period.

4 The Proposed Hardware Architecture

In the architectures proposed in [6] and [1], it is used one convolution block per each convolution operation, dividing the processing by octaves and resulting in $O \cdot S$ convolution blocks. In this work we present an architecture that only uses S convolution blocks for the $O \cdot S$ convolution operations, dividing the processing by scales and providing the same throughput.

A block diagram of the overall architecture is shown in Figure 1 a). This diagram shows a system of four octaves, five scales and a seven coefficient kernel ($O = 4, S = 5, k = 7$); but it could be generalized for any configuration.

The hardware architecture consists, in a major manner, of scale computation blocks (SCB). One SCB performs the O Gaussian filtering operations of a given scale as discussed in Section 3.3. Therefore, each SCB has O input and output ports, one for each octave respectively, where the sample period of each octave is defined by equation (2).

As can be seen in Figure 1 a), the SCB blocks are interconnected in cascade in order to have a constant convolution kernel size and to avoid convolutions with big kernel sizes.

A SCB block, to perform Gaussian filtering, takes advantage of the separability property of the Gaussian kernel as described in Section 3.2. As can be seen

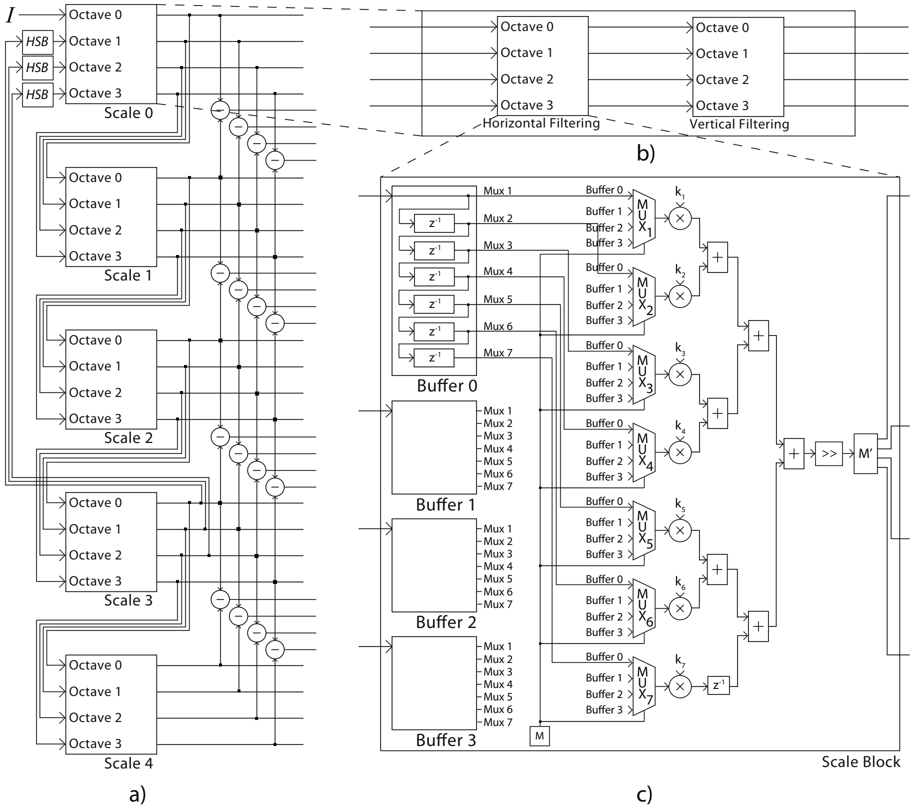

Fig. 1. Proposed pipelined architecture

in Figure 1 b), this block performs Gaussian filtering by convolving an image in the horizontal direction and then in the vertical one.

The internal arrangement of the horizontal filter in the SCB block is detailed in Figure 1 c). Each input signal is shifted throughout k registers, where k is the convolution kernel width. The k values of the O octaves are multiplexed with the aim of controlling the octaves processing order and accomplishing octaves processing interleaving. The multiplexers logic for octaves interleaving at a given time t is determined by the M block which implements function $m(t)$ and fulfills the condition stated in (3). The interleaving order is defined as follows:

$$m(t) = \begin{cases} o_0 & \text{if } t \equiv \varepsilon(o_0) \mod \tau(o_0) \\ o_1 & \text{if } t \equiv \varepsilon(o_1) \mod \tau(o_1) \\ \vdots & \vdots \\ o_{O-1} & \text{if } t \equiv \varepsilon(o_{O-1}) \mod \tau(o_{O-1}). \end{cases}$$

The structure of the vertical filter is the same as the horizontal; with the distinction that each buffer stores the last k lines instead of the last k elements.

By interleaving octaves processing in a single convolution processor it is possible to save a lot of silicon area and the consequent power consumption reduction. Also, to avoid operating with fixed point values, kernel coefficients are multiplied by an appropriate constant. Later, the filtered result is normalized by dividing it by this same constant. More desirably, the constant chosen must be a power of two in order to replace the division operation by a simple shift.

The HSB block in Figure 1 a) performs image downsampling.

5 FPGA Implementation and Experimental Results

5.1 Implementation Characteristics

A system configured with $O = 4$, $S = 6$ and $k = 7$ to process 320 × 240 sized images was implemented on a Xilinx Virtex II Pro FPGA (XC2VP30-5FF1152). This system was implemented using System Generator + Simulink. The estimated resources occupied by this implementation and its comparison with Bonato et al. system [1] are summarized on Table 1. As discussed in previous sections, the system returns a result every two clock cycles. Under this implementation, with a 50 MHz clock rate, the time taken to detect scale-space extrema in a 320 × 240 image is 3 ms, so, it is possible to process 330 frames per second. This result was compared, in terms of performance, with Vedaldi software implementation [9] running on a PC (1.8 GHz Core Duo and 1 GB RAM). Our system proved a significative speed up of 250x.

Table 1. Implementation Characteristics and Comparison with [1]

Resources	Our System $O = 4$, $S = 6$ and $k = 7$	DoG part of [1] $O = 3$, $S = 6$ and $k = 7$
Slices	5068	-
Flip-flops	6028	7256
Look Up Tables	6880	15137
Blocks RAM	120 (2.1 Mb)	0.91Mb

5.2 System Reliability

In order to test our implementation reliability, we checked for matches between features found by a complete software version and a hybrid implementation where scale-space extrema were obtained by our system. The SIFT software implementation used was Lowe's [5]. The hybrid implementation was created from Vedaldi's, where the first SIFT computation stage was executed by our system. We looked for matches between these two implementations on 26 images. The main differences between matches are due to the approximations on the DoG calculation process. However, these approximations did not greatly affected the final detected features. The mean errors in coordinates, scale and orientation of the detected features are $\Delta x = 1.127$, $\Delta y = 1.441$, $\Delta\sigma = 0.149$ and $\Delta\theta = 0.047$ respectively.

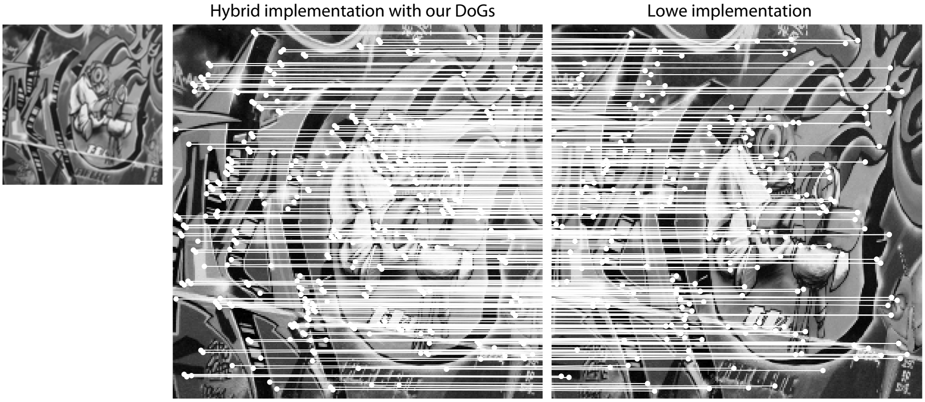

Fig. 2. Matches between the hybrid implementation (using our system results) and Lowe implementation for an example image

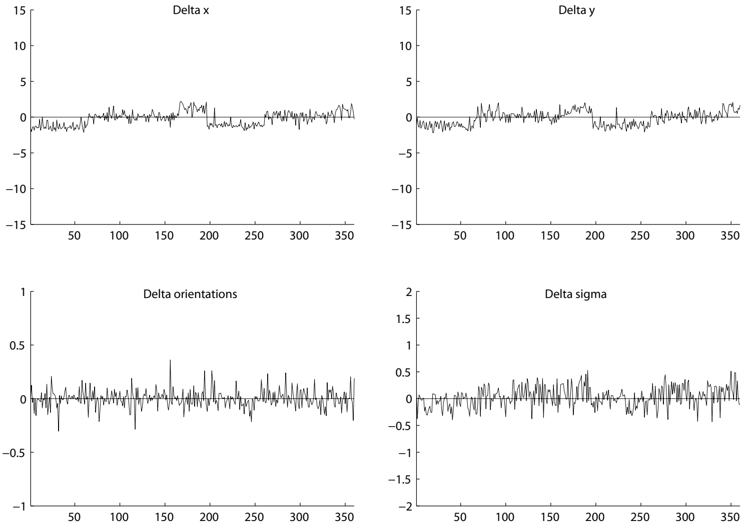

Fig. 3. Errors in coordinates, scale and orientation of the detected features for an example image

Figure 2 shows an example of detected features by the two implementations and their matches. The errors in coordinates, scale and orientation computation for this example image, are shown in Figure 3.

6 Conclusions

We have proposed a parallel hardware architecture for one of the most computationally expensive parts of the SIFT algorithm: the scale-space extrema detection. For this purpose, we exploited some algorithm particularities such as its intrinsic data parallelism, the separability property of the Gaussian kernel and the octaves processing interleaving possibility. The mean errors of the SIFT features detector and descriptor, based on our system results, are $\Delta x = 1.127$, $\Delta y = 1.441$, $\Delta\sigma = 0.149$, $\Delta\theta = 0.047$. The results of the comparisons showed that our system needs less silicon area than Bonato et al. system, even processing one more octave. This area profit is due more to octaves processing interleaving.

References

1. Bonato, V., Marques, E., Constantinides, G.A.: A parallel hardware architecture for scale and rotation invariant feature detection. IEEE Transactions on Circuits and Systems for Video Technology 18(12), 1703–1712 (2008)
2. Djakou Chati, H., Muhlbauer, F., Braun, T., Bobda, C., Berns, K.: Hardware/software co-design of a key point detector on FPGA. In: FCCM 2007: Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, Washington, DC, USA, pp. 355–356. IEEE Computer Society, Los Alamitos (2007)
3. Grabner, M., Grabner, H., Bischof, H.: Fast approximated SIFT. In: Narayanan, P.J., Nayar, S.K., Shum, H.-Y. (eds.) ACCV 2006. LNCS, vol. 3851, pp. 918–927. Springer, Heidelberg (2006)
4. Ke, Y., Sukthankar, R.: PCA-SIFT: a more distinctive representation for local image descriptors. In: 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 506–513 (2004)
5. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60(2), 91–110 (2004)
6. Pettersson, N., Petersson, L.: Online stereo calibration using FPGAs. In: Intelligent Vehicles Symposium, 2005. Proceedings. IEEE, pp. 55–60 (2005)
7. Se, S., Ng, H.k., Jasiobedzki, P., Moyung, T.j.: Vision based modeling and localization for planetary exploration rovers. In: 55th International Astronautical Congress 2004 (2004)
8. Sinha, S., Frahm, J.-M., Pollefeys, M., Genc, Y.: Feature tracking and matching in video using programmable graphics hardware. Machine Vision and Applications
9. Vedaldi, A.: An open implementation of the SIFT detector and descriptor. Technical Report 070012, UCLA CSD (2007)