A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection

Vanderlei Bonato, Eduardo Marques, and George A. Constantinides, *Senior Member 2008, IEEE*

***Abstract*—This paper proposes a parallel hardware architecture for image feature detection based on the SIFT (Scale Invariant Feature Transform) algorithm and applied to the SLAM (Simultaneous Localization And Mapping) problem. The work also proposes specific hardware optimizations considered fundamental to embed such a robotic control system on-a-chip. The proposed architecture is completely stand-alone; it reads the input data directly from a CMOS image sensor and provides the results via a Field-Programmable Gate Array (FPGA) coupled to an embedded processor. The results may either be used directly in an on-chip application or accessed through an Ethernet connection. The system is able to detect features up to 30 frames per second (320×240 pixels) and has accuracy similar to a PC-based implementation. The achieved system performance is at least one order of magnitude better than a PC-based solution, a result achieved by investigating the impact of several hardware-orientated optimizations on performance, area and accuracy.**

***Index Terms*—SIFT, FPGA, SLAM, Embedded Robotics.**

I. Introduction

THE continuous improvement in image processing algorithms and the intense advance in hardware technologies have supported the development of solutions for complex problems that were considered impractical to be solved two decades ago [1]. This is the case for the image matching problem where practical solutions related to object motion tracking [2] and object recognition [3], [4], [5] are now available. Some derivations of this image matching research have also been applied to specific problems, such as in the localization and mapping problem based on vision in mobile robotics [6]. However, as these algorithms are normally implemented to be executed on personal computers, their direct application on mobile robot platforms sometimes is not possible due to processing restrictions [7], [8].

Considering the necessity of customized architectures to process such algorithms in real time, we present in this paper a parallel hardware architecture for scale and rotation invariant feature detection, based on the SIFT (Scale Invariant Feature Transform) algorithm [4] for the SLAM (Simultaneous Localization And Mapping) problem [9]. The proposed architecture is stand-alone and completely embedded on an FPGA (Field-Programmable Gate Array). It processes up to 30 frames/second read directly from a CMOS camera, and returns the detected features associated with descriptors.

The main contributions of this work are:

- as far we know, the first complete presentation of an on-chip architecture for the SIFT algorithm,
- the definition of a system configuration and algorithm modifications appropriate for hardware-orientated implementation, and
- an analysis of the throughput requirement for each SIFT algorithm function in a real-time embedded setting.

Additionally, this work presents an on-chip hardware/software co-design, which gives flexibility for the users to customize features descriptors according to the application needs.

The paper is organized as follows. Section II presents some related work starting with early results and reviewing some related implementations customized for FPGAs. Section III presents the SIFT algorithm. Section IV presents our own hardware-orientated architecture of the algorithm, along with a detailed description of the architecture developed. In Section V, experimental results are presented in order to verify the system reliability and the FPGA resources needed. Finally, Section VI concludes the work.

II. Related Work

One of the first widely used interest point detection algorithms for image matching purpose was proposed by Harris and Stephens [10] extended from the Moravec corner detector [1], which extracts interest points from edge and corner regions based on gradient magnitude information. Although the Harris and Stephens algorithm is susceptible to changes in image scale [4], it has been intensively used as a base in stereo and motion tracking problems. Shi and Tomasi have also proposed a system where good features are detected by analysing the pixel intensity behavior of the feature regions during the tracking operation [11]. This feature monitoring method is efficient at rejecting points that vanish in different image viewpoints and is robust to affine image transformations. However, as it is based on window intensity variation, it is consequently sensitive to illumination changes.

From these important contributions there has been a continuous effort in to develop robust algorithms to detect features invariant to scale, affine transformations, rotation and change in illumination [12], [13]. Among these proposals, Lowe [4] has presented one of the most complete and robust results, which has been named SIFT.

V. Bonato and E. Marques are with the Institute of Mathematical and Computing Sciences, The University of São Paulo, São Carlos, 13560-970, Brazil, email: vbonato, emarques@icmc.usp.br.

G. A. Constantinides is with the Department of Electrical and Electronic Engineering, Imperial College London, London, SW7 2BT, U.K., email: g.constantinides@imperial.ac.uk.

When fully implemented, SIFT has a high computational cost, which has led to the proposal of some optimized variants [14], [15]. Se [16] cites that a customized version of this algorithm has been implemented on FPGA to support a stereo vision system for robotic navigation, which has improved the performance by 10× in relation to a Pentium III 700MHz (from 600ms to 60ms for images of 640×480 pixels), however this work does not present architecture details and does not discuss the implementation of stability checks for location, contrast and edge responses in hardware.

In [17] a partial implementation of the SIFT on FPGA for stereo calibration is demonstrated. It presents an architecture to detect feature candidates, which operates at 60 frames/second. However, the work does not state the image resolution or discuss FPGA area allocation. Another FPGA-based partial implementation for the SIFT is presented in [18], which needs 0.8ms to detect keypoints from images of 320×240 pixels. However, again little information about the architecture has been provided. Kim [19] proposes a dedicated processor for the SIFT computation, without considering the keypoint stability checks, which takes up to 62ms per 320×240 frame. In contrast, we present a complete SIFT implementation, introducing all phases of the original algorithm, but optimised for hardware performance. Our architecture requires 33ms per 320×240 frame.

III. The Scale and Rotation Invariant Feature Detection Algorithm

This section describes the SIFT algorithm, on which our scale end rotation invariant feature detection system is based [4]. The algorithm is divided into three main stages: the first one identifies pixel candidates to be a feature, the next one then applies a set of tests to verify whether the candidates are stable in relation to image transformations in order to accept or reject them. Having detected stable features, the final stage generates a descriptor/signature to be associated with the feature.

A. Selecting Feature Candidates

A candidate feature (keypoint) is a pixel located at a specific image frequency scale that has either the minimum or the maximum value in relation to its neighbourhood, defined by a 3×3 window located at the same scale space and at the upper and lower adjacent scales (total 26 pixels). The frequency band for each scale is obtained by using the difference of Gaussian (DoG) operation, which is computed by subtracting two identical images convolved by two different Gaussian kernels. Equations (1) and (2) define the convolution operation ($*$) for the first and the other subsequent scales, respectively, and Equation (3) defines the difference operation, where I is the input image, K the Gaussian kernel, G the smoothed image and D the resulting image at a specific frequency scale defined by the kernel values.

$$G_0(x,y) = K_0(x,y) * I(x,y) \tag{1}$$

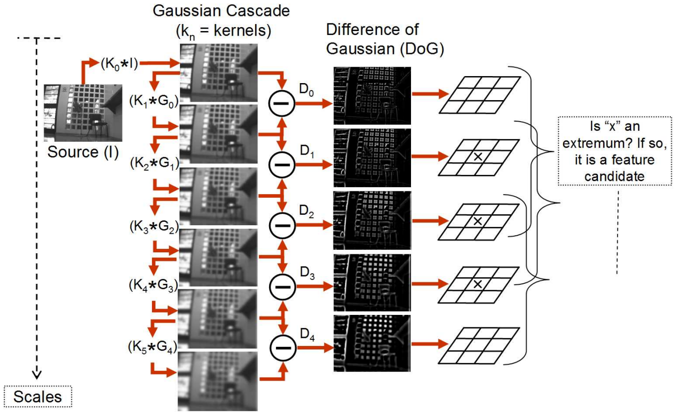


Fig. 1. Block diagram representing the main operations used to detect feature candidates in one octave by using a set of six Gaussian smoothed images.

$$G_{s+1}(x,y) = K_{s+1}(x,y) * G_s(x,y) \tag{2}$$

$$D_i(x,y) = G_{s+1}(x,y) - G_s(x,y) \tag{3}$$

Fig. 1 illustrates these operations by showing an example with five frequency bands computed from six Gaussian convolution operations. In this case, as the keypoint is determined by analysing its scale and the other two adjacent, it is possible to have keypoints belonging up to three different scales. In the figure, the "x" symbol corresponds to a keypoint.

The number of scales is a parameter and is defined according to the application. In addition to the number of scales, the algorithm is also parameterized by the number of octaves, where the image size is down-sampled by two for each new octave.

B. Testing the Feature Stability

The stability check stage has the objective of rejecting or accepting a keypoint as a feature, evaluated through three functions: location refinement, contrast check and edge responses. The location refinement is performed by (4), where ω is the location correction offset for the coordinates $< x, y >$ and the scale s, D the DoG function, and l the pixel location vector (x, y, s) [20]. This equation performs an interpolation operation with the pixels found inside of the keypoint's neighbourhood (26 pixels), with ω added to the current keypoint position in order to produce its new location. In our implementation the refinement stops when all $< x, y, s >$ offsets are smaller than 0.5 pixel, the new keypoint location is on the image border, or if more than three consecutive corrections have been performed. In the latter two cases the keypoint is rejected and in the first one it is held for the next steps of stability checks.

$$\omega = -\left(\frac{\partial^2 D}{\partial l^2}\right)^{-1} \frac{\partial D}{\partial l} \tag{4}$$

The ω offset is also used in (5) to compute the keypoint contrast. If the result is smaller than a user-defined threshold then the keypoint is rejected and, if not, the final stability test is performed.

$$\xi = D + \frac{1}{2}\left(\frac{\partial D}{\partial l}\right)^{T}\omega \tag{5}$$

Finally, principal curvature analysis is evaluated to reject keypoints that are located at poorly defined edges, which are consequently highly unstable to noise. This is particulary necessary for the difference of Gaussian function because most of the detected keypoints are located at edges. The bigger the principal curvature, the poorer is its edge. Thus, (6) computes the principal curvature and rejects if it is above a pre-established threshold.

$$\frac{\left(\frac{\partial D}{\partial x^2} + \frac{\partial D}{\partial y^2}\right)^2}{Det(H)} < Threshold \tag{6}$$

where:

$$H = \begin{bmatrix} \frac{\partial D}{\partial x^2} & \frac{\partial D}{\partial x \partial y} \\ \frac{\partial D}{\partial y \partial x} & \frac{\partial D}{\partial y^2} \end{bmatrix} \tag{7}$$

Once the keypoint has passed through the previous phases, it is finally classified as a feature. At this point a keypoint is identified by its $< x, y >$ coordinates and scale s.

C. Labeling Features

In this stage, the features receive a label represented by a vector of 128 bytes. From here the difference-of-Gaussian image is not used any more; all information needed to compute the descriptor is now obtained from the Gaussian cascade images (see Fig. 1).

Given a feature, the first step is to compute its dominant orientation. So as to have this information, an image from the Gaussian cascade is chosen according to its scale. Having chosen the appropriate image, the orientation and gradient magnitude for each pixel in a 16×16 region centred in the $< x, y >$ coordinates is evaluated by (8) and (9), respectively. From these data an orientation histogram of 36 bins is created, where each element in this histogram is weighted according to its gradient magnitude and its distance from the feature location. As a result, the highest peak in the histogram will represent the dominant feature orientation. However, in order to improve the stability, the final result is given by the interpolation among the three highest peaks.

$$\begin{aligned} \theta(x, y) = \text{atan2}(&G(x, y+1) - G(x, y-1), \\ &G(x+1, y) - G(x-1, y)) \end{aligned} \tag{8}$$

$$\begin{aligned} m(x, y) = ((&G(x+1, y) - G(x-1, y))^2 + \\ &(G(x, y+1) - G(x, y-1))^2)^{\frac{1}{2}} \end{aligned} \tag{9}$$

The previous orientation and gradient magnitude computed from the 16×16 region around the feature is also used to generate the feature label. However, in this case, firstly this region is rotated relative to the dominant orientation and then an orientation histogram of 8 bins is generated for each 4×4 sub-regions. This rotation is necessary to generate a descriptor vector invariant to rotation. Finally the descriptor is formed by copying the magnitude of each histogram bin into a vector of 128 bytes.

IV. The Proposed Parallel Hardware Architecture

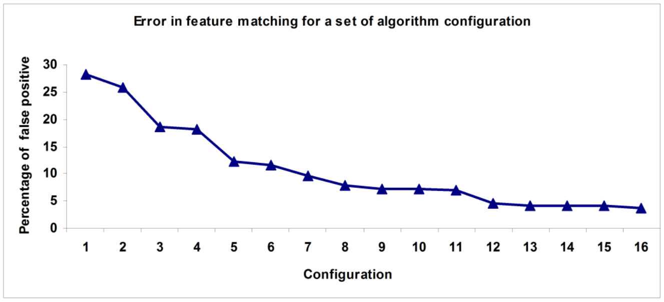
The algorithm presented in Section III is considered an efficient solution to detect features, however it has high computational cost caused mainly by the Gaussian filter cascade and the keypoint detection with stability checks. A system configured with 3 octaves and 6 scales per octave based only on software takes 1.1s to extract 514 features from an image of 320×240 pixels in an Intel Core 2 1.66GHz processor (the software used for this performance test is provided by Hess [21]). Based on a different implementation, Se [16] presents a better software performance of 600ms to compute an image of 640×480 pixels, however these figures are unsatisfactory for real time applications. In this section we present a System-on-a-Programmable-Chip (SOPC) to detect feature at 30 frames/second from images of 320×240 pixels.

The following subsections initially present a set of SIFT algorithm configurations so as to find out which option is most suitable for hardware implementation, followed by an overall architecture description where the throughput requirement for each system function is analysed along with the required and achieved parallelism gain. Then, the implemented hardware modules for each function are presented and their performance are demonstrated.

A. System Configuration

Prior to defining the hardware architecture for the SIFT algorithm, a set of possible system implementations is analysed in order to establish a configuration that has high feature matching reliability and is appropriate for hardware-orientated implementation. Fig. 2 shows 16 configurations and their influence on the reliability, which was generated from two sets of ten images each, where the second set is a transformation of the first one in relation to scale, rotation and viewpoint. The first parameter determines whether the orientation and gradient magnitude are computed from an image chosen by the feature scale or from a pre-defined image scale. The second parameter gives the option to either duplicate the image size used as input in octave 0 or to keep its original size. The other two parameters are related to feature stability checks; the first one activates the keypoint location and the second one verifies the contrast and the edge response.

As seen in Fig. 2, the configuration options from 12 to 16 produce a very similar rate of incorrect matching (false positive). Thus, giving priority to the matching reliability criteria, the final system configuration is chosen from this range based now only on hardware requirements. Analysing these five options, the most suitable one is 12 as it avoids the necessity of processing the orientation and gradient magnitude for all scales and also reduces the on-chip memory needs by processing the input image in its original size. Another motive



Configuration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ori and mag based on the feature scale	0	1	1	0	1	0	1	0	1	0	0	0	0	1	1	1
duplicated image	1	1	1	1	0	0	1	0	0	1	0	0	1	1	0	0
location	0	0	1	1	0	0	0	1	1	0	0	1	1	1	1	0
contrast threshold and edge response	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1

Fig. 2. An analysis showing the feature matching error in relation to the system configuration.

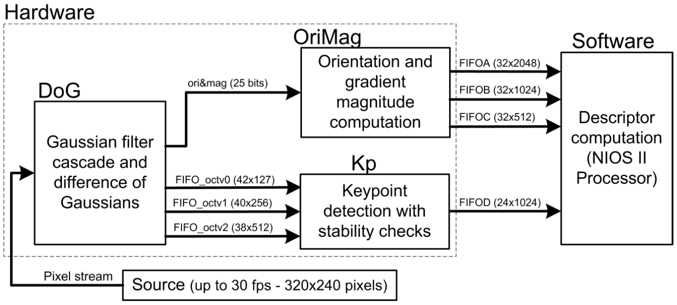

Hardware
Software
DoG
Gaussian filter cascade and difference of Gaussians
OriMag
Orientation and gradient magnitude computation
Kp
Keypoint detection with stability checks
Descriptor computation (NIOS II Processor)
ori&mag (25 bits)
FIFO_octv0 (42x127)
FIFO_octv1 (40x256)
FIFO_octv2 (38x512)
FIFOA (32x2048)
FIFOB (32x1024)
FIFOC (32x512)
FIFOD (24x1024)
Pixel stream
Source (up to 30 fps - 320x240 pixels)

Fig. 3. Block diagram of the implemented system, composed of three hardware blocks, used to extract features from the images and to pre-compute data for the descriptors, and one block in software, which associates descriptors to features. The blocks are connected using dedicated channels, where each one, except the channel between DoG and OriMag, has internal FIFOs used as buffer while a receiver block is temporarily unavailable.

of choosing option 12 is that most computation in the contrast threshold and edge response functions, which are activated in every configuration option inside this range, are reused from the keypoint location function.

The system has also been structured to compute the orientation and gradient magnitude of every pixel for one image per octave independent of whether or not the pixel lies inside a keypoint neighbourhood. Although this computation could have been performed only around a feature location, the current approach reduces the overhead in time to compute a feature descriptor, since the orientation and gradient magnitude can be computed in parallel while features are being detected. In addition, the FPGA area dedicated to this computation would otherwise be lying idle.

In addition to the previous settings, the system is configured for three octaves with six scales in each one and processes input image of 320x240 pixels. This number of octaves is sufficient for this image size, since a fourth octave would produce small and blurred images, resulting in a remote chance of detecting features. However, the number of scales was chosen based on [4], where it has demonstrated that this number has the highest feature repeatability rate.

B. Overall System Architecture

Fig. 3 shows a block diagram of the proposed architecture, featuring three blocks in hardware (DoG, OriMag, Kp) and one in software (NIOS II). The DoG block receives as input a pixel stream from a camera and performs two operations, the Gaussian filter and the difference of Gaussian, whose results are then sent to the OriMag and Kp blocks, respectively. The OriMag computes the pixel orientation and gradient magnitude while the Kp block detects keypoints and realizes the stability checks to classify keypoints as features. Finally, the software block, based on the NIOS II softcore processor, generates a descriptor for each feature detected by the Kp block based on the pixel orientation and gradient magnitude produced by the OriMag block.

Most system functions are implemented in hardware since their high computational requirement can be achieved by exploiting parallelism on chip. However, the major part of the feature descriptor is processed in software because its computational requirement is viable to be achieved by an embedded processor; it is easer to implement it in software than hardware, and finally, it gives more flexibility to customize the descriptor according to the final application.

1) Parallelism Requirement Analysis: The parallelism requirement of our system is defined based on the number of operations executed per second in real time (30 frames p/s), which is computed by multiplying the system throughput with the number of operations performed to generate a result; by assuming that each operation takes one clock cycle, which happens when all fractional circuits are fully pipelined and there are no pipeline stalls[1]; and by predicting what is the clock frequency supported by the target device (FPGA)[2] where the operations are implemented.

According to the analysis presented in detail below, the whole system needs to perform 0.96G operations/second (see Table I), which is achieved through two leves of parallelism. In the first one, the system is divided into four blocks to run in parallel (see Fig. 3), and in the second one, the parallelism is exploited inside of each hardware block. The decision to divide the system into blocks and what each block implements is based on the data dependency among the algorithm functions and on the requirement performance of each part, in order to have a good balance between performance, flexibility and FPGA area.

The DoG block has a constant throughput defined by the number of octaves and scales of the system. As the proposed system has 3 octaves and 6 scales, for each frame provided by the camera 18 images need to be filtered and 15 pairs of filtered images need to be differentiated. Considering that octave 0 processes images in its original dimensions (320x240 pixels) and that for each subsequent octave the image dimensions are reduced to half, the throughput in real time for the Gaussian filter is 18.1M pixels/second and for the difference of Gaussian is 15.1M pixels/second. As the Gaussian filter needs to perform

[1] Such condition is feasible for hardware implementations, since each circuit computes a pre-defined function and can be supported by dedicated data path.

[2] The clock frequency of a circuit on an FPGA depends on the operation type, synthesis tool, FPGA family and FPGA resource availability.

28 and the difference of Gaussian one operation per result, the number of operations/second for each function is 506.8M and 15.1M, respectively. Thus, if we assume that the DoG block can run on an FPGA at 50MHz, the required parallelism degree for the Gaussian filter is $10.1\times$ and for the difference of Gaussian is $1\times$. In this paper, the required parallelism degree is calculated multiplying the system throughput (in real time) by the number of clock cycles per throughput, and then, dividing its result by the maximum clock frequency supported by the circuit. For instance, the Gaussian filter gain of $10.1\times$ is equal to (18.1M x 28)/50M, which means, for a limited clock frequency, how many times the system throughput must be increased in order to achieve the desired performance.

Another block that has a constant throughput is OriMag. However, its performance requirement is independent of the number of scales. It is given only by the number of octaves and the image dimensions. Thus, considering an input image of 320x240 pixels and 3 octaves, its required throughput in real time for the orientation and gradient magnitude computation is 6M results/second. As each result is produced by 30 operations, the number of operations/second is 180M. Since this block implements only shift, addition and subtraction operations, it is expected that its circuit can run on an FPGA at 100MHz, which consequently requires a parallelism degree of $1.8\times$.

In contrast to the previous blocks, the throughput for Kp depends on the pixel value. To predict in advance the system requirement, we may assume 1% of pixels in an input image are keypoints. Thus, for the worst case, when the stability check is classifying all keypoints as features, the Kp block throughput in real time is 23k features/second. Although the whole Kp block throughput varies, internally, the keypoint detection has a constant throughput given by 9.1M results/second, which needs 26 operations to produce one result and, consequently, needs to perform 236.6M operations/second. As the stability check, which has throughput of 23k features/second, needs to compute 367 operations (worst case) for each result, its number of operations/second is 8.4M. Assuming that the clock frequency for this Kp block, which envolves a significant number of fixed-point operations, is 50MHz, the required parallelism gain for the keypoint detection is $4.73\times$. However, the required parallelism gain for the stability check is a special case, since it is unknown the input data sequence. If we assume that all keypoints are coming at once one after another and that a result must be produced for each clock cycle, then the number of operations/second would be 3.3G (9.1M x 367) and consequently the parallelism degree should be $66\times$. Such approach is inefficient, since the stability check hardware would remain idle for 99% of whole processing time. Therefore, a better solution is to develop a hardware module along with a buffer in order to store data whenever it comes before the previous one has been processed, which considerably reduce the parallelism requirement and, consequently, the FPGA area. Since the slower is the stability check module frequency the bigger is the buffer size requirement, the parallelism degree has been defined to be at least $2\times$, since it provides a good balance for FPGAs between logic resource and internal memory requirements (see Section IV-C3 for the results).

Finally, assuming that the stability check, which has a high rejection rate defined specially for our application, classifies only 5% of keypoints as features, the software block throughput in real time is 1.1k descriptors/second. As for each result 10k operations are computed, the NIOS II processor needs to process 11M operations/second. Since the processor computes sequentially and it has a clock frequency of 100MHz, each operation has 9.1 clock cycles to be executed.

Table I summarizes the required parallelism degree for each system module previously presented, along with the actual parallelism obtained with the implemented hardware modules. As can be seen from this table, the only module that has the achieved gain smaller than the required one is the OriMag block. In order to solve this performance difference, the system is implemented with two OriMag blocks running in parallel. The implementation details of every system module are described in Section IV-C, where is shown how the parallelism gains were achieved and how FPGA resources were saved through hardware optimizations.

C. Hardware Module Implementation

1) DoG: The Gaussian filter is implemented considering that its kernel is separable [22], since this approach is computationally more efficient than the traditional one, which directly convolves a 2D kernel with the image. For a 7x7 kernel, this approach needs to perform 28 operations to produce one result.

To achieve the minimal required parallelism of $10.1\times$, we propose an optimized hardware module, as presented in Fig. 4. This module takes the advantage of the Gaussian kernel symmetry and save two multipliers by reusing data from the multiplication result at k_1 and k_2 (row order) and save four more multipliers by assuming that the kernel vector has always the values one or zero at position 1 and 7 and consequently avoid the multiplications at those points. It is possible to keep this pattern for kernels generated with any σ values by simply multiplying the kernel coefficients by an appropriate constant. Since this multiplication only alters the kernel values, the filtered pixel is normalized simply by dividing its value by the new kernel sum (N).

The proposed system implements 18 Gaussian modules, one for each scale/image, which allowed the clock frequency to be 2.1MHz, which corresponds to the same pixel stream rate provided by the camera. Another approach would be to filter more than one image per Gaussian module, since its 2.1MHz could be increased to the previously planned 50MHz. However, dedicated modules avoid FPGA-expensive multiplexing operations to share computations. In addition, as the adopted FPGA provides DSP multipliers, the number of logic elements needed to implement an individual module can be considered small.

On the other hand, the difference of Gaussian performs just a subtraction operation per throughput, which can be implemented by a single hardware module running at 15.1MHz. However, as this operation needs small FPGA area, this subtraction operation is replicated for each pair of images to avoid multiplexing operations as well. Fig. 6 presents the proposed architecture for whole DoG block.

TABLE I
THE REQUIRED AND ACHIEVED PARALLELISM GAIN FOR EACH SYSTEM FUNCTION.

System block	Function	Required throughput p/s	Operations per result	Operations p/s	Required parallelism gain	Achieved parallelism gain
DoG	Gaussian filter	18.1M	28	506.8M	10.1x	28x
	difference of Gaussian	15.1M	1	15.1M	1x	1x
OriMag	orientation and gradient magnitude	6M	30	180M	1.8x	1.42x
Kp	keypoint detection	9.1M	26	236.6M	4.73x	8.69x
	stability checks	23k(expected)	367(worst case)	8.4M	2x	2.78x
NIOS II	descriptor computation	1.1k(expected)	10k(can vary)	11M	-	-
Total	Whole system modules	48.3M	-	0.96G	-	-

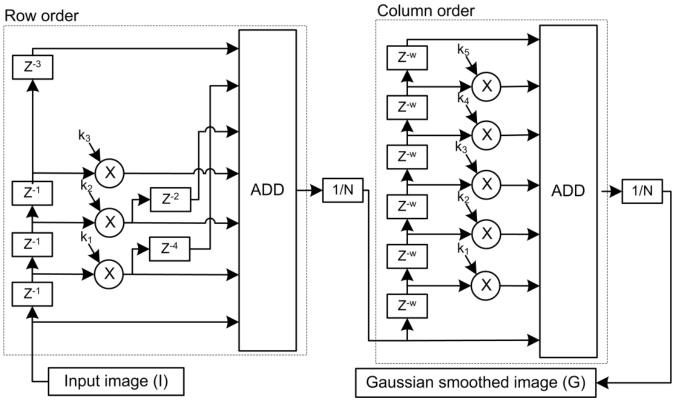


Fig. 4. An pipeline architecture for the Gaussian filter optimized module; the left side shows the convolution in row and the right in column order for the separable kernel approach.

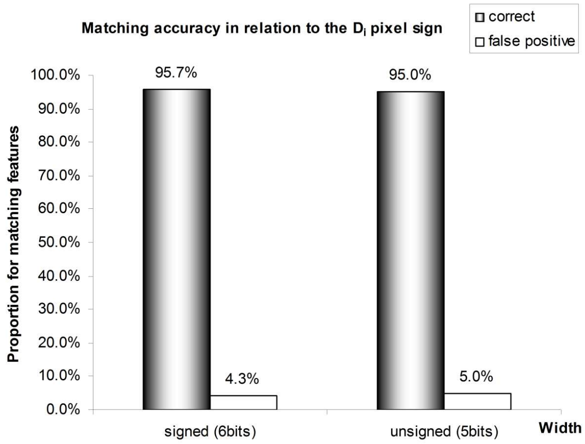


Fig. 5. Proportion of false positive and true feature matching in function of the D_i pixel sign.

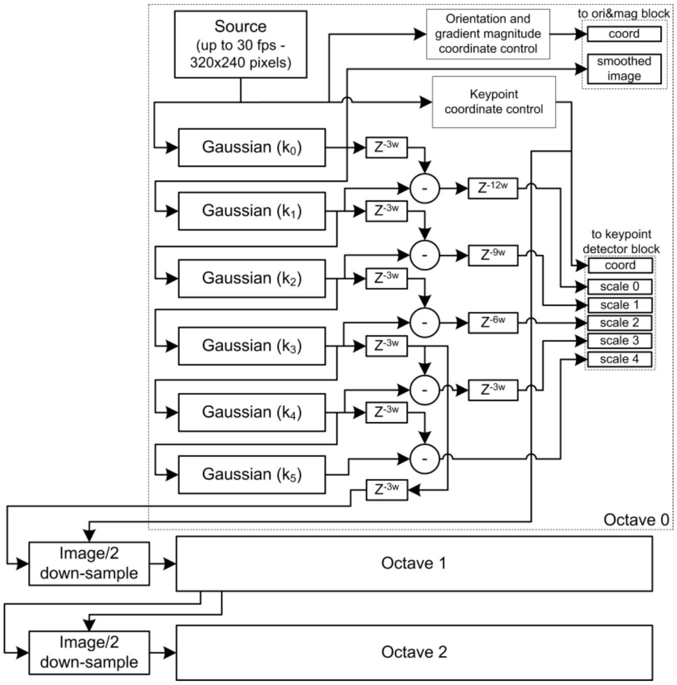


Fig. 6. A pipeline architecture implementing the Gaussian filter cascade and the difference of Gaussian function.

Additionally, as the Gaussian images (G_s) use 8 bits to represent one pixel, 9 bits would be sufficient to store the difference of Gaussian results (D_i) with the sign. However, since a D_i result bigger than 2^5 is highly unlikely, five bits (without the sign) was adopted. Fig. 5 shows how much the D_i pixel sign affects the proportion of false positive and correct matchings, which was computed from the same set of images previously used to determine the system configuration. What is affected by signed or unsigned format is the number of total features detected, which is reduced to half since a keypoint is considered a feature only if it has the maximum value in its neighbourhood, not considering in this case the test for the minimum. This reduction in feature candidates is not considered a problem, since few tens of features are sufficient for our SLAM application. However, by reducing the D_i pixel width in 4 bits a considerable amount of FPGA resources is saved.

2) OriMag: The orientation and gradient magnitude are computed based on the CORDIC algorithm in vector mode as this solution maps efficiently into the FPGA fabric [23]. Another approach frequently used to solve this problem is by pre-computing the functions and storing their results in lookup tables or memories. However, the problem of this solution is that the area or memory needed is exponential to the input-word length. In our case, the input word-length is 16 bits, making CORDIC a more efficient option.

Each CORDIC iteration computes two operations for the orientation (given in 36 bins) and four for the gradient magnitude. For this system five iterations are sufficient for the desired precision, so the final orientation and gradient

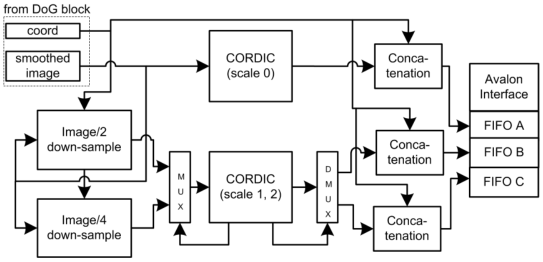


Fig. 7. Orientation and gradient magnitude computation architecture based on the CORDIC algorithm.

magnitude are computed in 30 operations.

In order to achieve the minimal required parallelism gain of 1.8×, the system is proposed with two identical modules, where each one has a gain of 1.42×, being the first one dedicated for octave 0 and the second one shared between octave 1 and 2. Such gain of 1.42× is limited by the data dependency among the CORDIC operations. Fig. 7 shows a block diagram of the proposed OriMag block, where each CORDIC module takes 21 clock cycles to compute the orientation and gradient magnitude results for one pixel. The current OriMag frequency is 100MHz.

Internally, the CORDIC represents data in fixed-point (11.8) format with 11 bits for the integer part since the maximum internal value is 594, which is given by the maximum value from (9) multiplied by the CORDIC gain 1.647, and with 8 bits for the fraction part as, empirically, this resolution is sufficient for the application. However, the final results are given using integer number format where the magnitude is represented by 10 bits and the orientation by 6.

Fixed-point format allows more operations to be processed in parallel than would have in the case of using a floating-point approach in single precision as the FPGA resources needed to implement fixed-point hardware are considerably lower than the alternative [24].

3) Kp: This block contains the keypoint detection, location, contrast and edge response functions, where the data flow between them is defined in this presented order. The keypoint detection is the only function that necessarily processes every pixel received from the DoG block. The other ones depend on the result from the previous function. For instance, the location is computed only for those pixels identified as keypoints. One advantage of this dependency is the possibility of having a module slower than the period between two pixels, since this difference can be compensated when a pixel is rejected by some previous function. The number of operations for each function is 26 for keypoint detection, 69 for each location interaction (can perform up to 5 interactions for a keypoint), 10 for the contrast and 12 for the edge responses. Thus, the maximum number of operations for the stability check (location, contrast and edge response) to compute a result is 367.

Based on the minimal required keypoint detection and stability check parallelism gains (4.73× and 2×), a hardware architecture is shown in Fig. 8, which is shared between the three octaves, where the keypoint candidates for an entire octave are identified in parallel by the *Extreme* module resulting in a parallelism gain of 8.69×. In this hardware block, the fastest case is 4 clock cycles which happens when the data to be processed (DoG pixel) are located at the border region. An intermediate point is when the DoG pixel is on the image active region and is not a keypoint candidate. Another level is when the DoG pixel is a keypoint candidate and the stability checks are carried out. In this case the time varies from 44 to 132 clock cycles, depending on whether the keypoint is rejected during the checks and on how many times the location correction is realised. Thus, the achieved stability check parallelism gain is 2.78×.

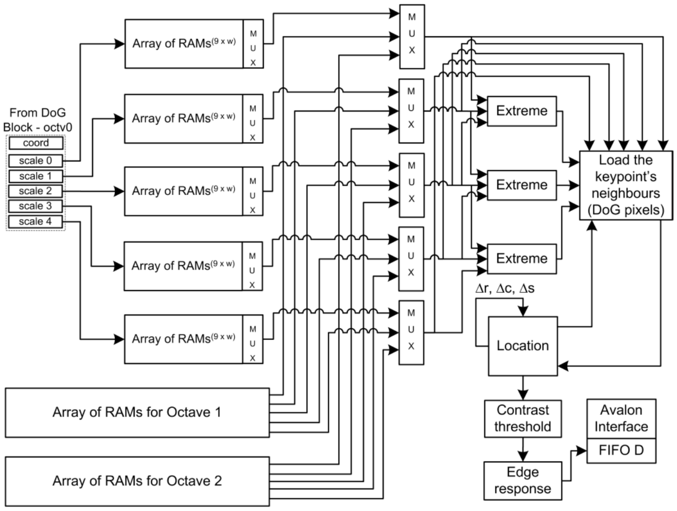

From DoG Block - octv0
coord
scale 0
scale 1
scale 2
scale 3
scale 4
Array of RAMs$^{(9 \times w)}$
Array of RAMs$^{(9 \times w)}$
Array of RAMs$^{(9 \times w)}$
Array of RAMs$^{(9 \times w)}$
Array of RAMs$^{(9 \times w)}$
M U X
Extreme
Extreme
Extreme
Load the keypoint's neighbours (DoG pixels)
Δr, Δc, Δs
Location
Array of RAMs for Octave 1
Array of RAMs for Octave 2
Contrast threshold
Avalon Interface
FIFO D
Edge response

Fig. 8. Architecture for keypoint detection with stability checks.

The current hardware block frequency is 50MHz, which is 21.7× faster than the DoG pixel rate. Thus, whenever a DoG pixel needs more than 21 clock cycles to be processed, any DoG pixel received in this meanwhile is stored on internal buffers. As a DoG pixel is rejected or classified as a keypoint candidate in 9 clocks, the remaining time is then used to compute those data from the buffers. This solution was implemented using internal FIFOs to store 127 DoG pixels for octave 0, 256 for octave 1 and 512 for octave 2 (see Fig. 3), and with these parameters an overflow has not occurred during our experiments. The biggest number of words accumulated in each FIFO, monitored during five frames, was 11 for octave 0, 28 for octave 1 and 50 for octave 2. Although the lower is the octave number the higher is the pixel rate, octave 0 has the lowest accumulated words since the implemented hardware gives priority to the lowest channel whenever more than one FIFO has data waiting for to be processed.

The location, contrast threshold and the edge response functions were implemented using fixed-point representation (20.8). The 20 bits integer part is needed to store the intermediate values in the matrix inversion operation from (4) and the 8 bits fraction part was adopted since it provides a good balance for the application between hardware requirement and result precision.

4) Software (NIOS II): The number of operations to compute a single feature descriptor from a 16x16 window around its place is approximately 2k for the dominant orientation and 8k for the descriptor vector (variations can occur in function to

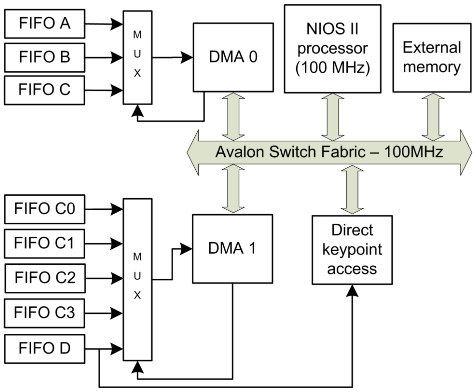


Fig. 9. A NIOS II based architecture to associate the feature's descriptor using the orientation and gradient magnitude data received from the *OriMag* block.

the context). This number would be higher if the orientation and gradient magnitude had not been computed in advance. However, as just few tens of features (5%) are expected to be approved by the stability checks in a frame, the number of operations/second is 11M, which is one of the lowest figure of whole system. Such performance can be achieved in either hardware or software implementation, however we decided to propose a solution via software for the reasons already presented in Section IV-B.

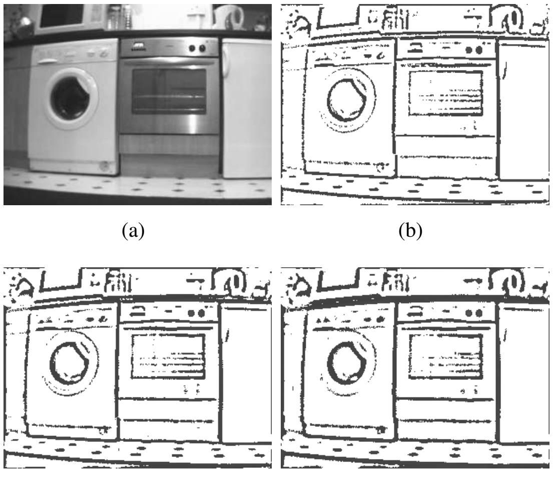
The current software implementation is based on the NIOS II softcore processor running at 100MHz. The processor is supported by two DMA channels; The first one is dedicated to transfer data from FIFO A, B and C (*Orimag* block) to the processor main memory, and the second one, is shared between FIFO D (*Kp* block) and the FIFOs C0 to C3, which are used to read data direct from four CMOS cameras. In the case of the DMA being used to read data from the cameras, there is another component controlled by software that gives direct access to FIFO D. A block diagram of this system is shown in Fig. 9.

The time needed to generate a descriptor for a feature already detected is 11.7ms. In case more speed is required, a fast version of the NIOS II processor can be used, or yet another descriptor algorithm more suitable for embedded processor could be proposed.

Finally, as the proposed system also implements in software a TCP/IP protocol, a feature can be either used in a software application running in the NIOS II processor or being accessed via Ethernet to be used remotely. Both options have been tested and are ready to be utilized for real applications.

V. Experimental Results

The results are analysed in three aspects. Initially, a set of experiments is conducted in order to identify the reliability of feature detection, and then, the resources needed to implement whole system on an FPGA are evaluated, pointing out which parts of the code have high implementation cost in the hardware architecture. At the latter part, a comparison in relation to hardware performance is presented and future work suggested.



(a) (b)

(c) (d)

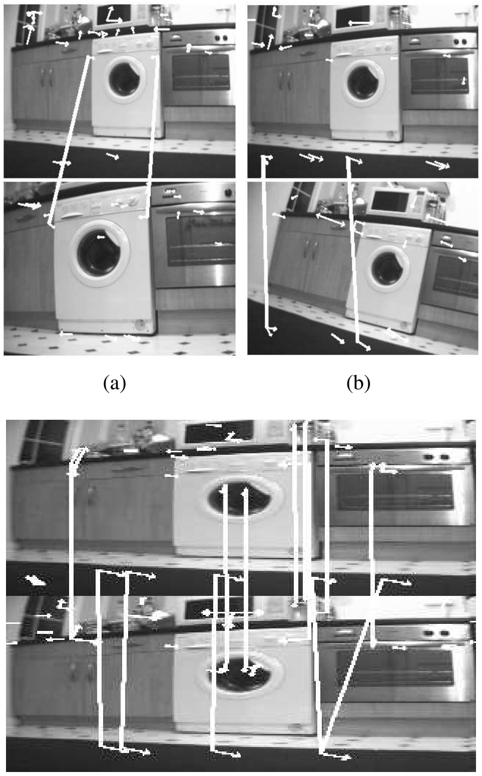
Fig. 10. An example of DoG result for images produced at octave 0 in the (a) scale 0, (b) 1 and (c) 2 taken direct from the DoG hardware block.

A. Reliability

In this project we gave special attention to the reliability of the results since for the SLAM application, which is based on an Extended Kalman Filter (EKF) [25], it is important to have low false positive occurrences so as to avoid algorithm divergence. Another aspect is that 12 features per frame is considered sufficient for the featured-based map maintenance [26].

To check the system reliability, 30 pairs of images, read direct from the camera, were matched in order to verify the rate of incorrect feature association. This operation was carried out in the NIOS II processor by using the BBF algorithm [27] as suggested in the original SIFT paper. This matching has been performed by comparing features taken from pairs of images with transformations between them, for example rotation and scale, produced by changing the camera position. For this set of images 630 features were detected and 135 of them were matched from which 95.74% were correctly matched. Despite the use of fixed-point format and the DoG pixel bitwise optimization, the rate of incorrect association has remained very similar to a pure software PC-based implementation, as previously demonstrated in Section IV-A. The most important difference between an FPGA and a PC-based platform is that the number of features in the PC is bigger, caused mainly because of the rejection of negative DoG pixels. Nonetheless, for our application this is not considered a negative result since the proportion between false positive and true matching has stayed approximately the same. Fig. 10 presents a typical image used in our experiment (a) along with three images produced in the DoG hardware at octave 0 for scales 0 (b), 1 (c) and 2 (d). In order to improve the DoG image visibility the black pixels were transformed in white. We can notice that details from image (b) to (c) are vanishing, showing that the Gaussian cascade filter is producing the desired result. In this demonstration, the Gaussian kernels has $\sigma = 1.25$, $\sigma = 0.96$ and $\sigma = 1.21$ for the images (b), (c), and (d), respectively.

To illustrate the matching operation Fig. 11 shows two example of association for (a) scale and (b) rotation transfor-



(a) (b)

(c)

Fig. 11. The white lines show the feature association between pair of images read directly from a CMOS camera and computed by the proposed system; pairs (a) and (b) demonstrate associations between images taken from different scale and rotation, respectively, and pair (c) shows an example with one case of incorrect association, which is represented by the most inclined white line.

mations, where the white lines between image pairs correspond to the association found by the BBF algorithm. In (c) is also shown one case of incorrect association and, as can be seen, this has happened because despite the global location of the feature being incorrect, locally the associated regions are almost identical. This consequently has resulted in identical orientation and gradient magnitude of both features (see the similarity of the arrows on the image) producing a similar descriptor as well. A solution for this problem could be to expand the region considered for the descriptor that currently is 16×16 pixels (as in the original SIFT algorithm).

B. Development Platform

An FPGA development kit with four CMOS cameras embedded on a robot base has been used to collect and process the images as shown in Fig. 12 [28]. Since this system has four cameras pointing in four fixed directions, practically all the area around the robot is covered, allowing an increase in feature quality by improving the rejection rate in the Kp block. The proposed feature detection system, which is shared between the four cameras, is able to process pixels directly from these image sensors.

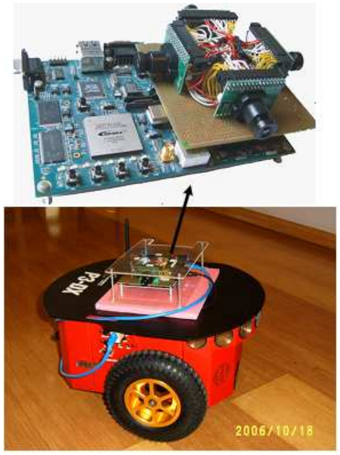


Fig. 12. The developing platform equipped with a multi-camera FPGA-based system embedded on a mobile robot base.

TABLE II
FPGA (ALTERA STRATIX II EP2S60F672C3) RESOURCES FOR EACH HARDWARE BLOCK AND FOR THE WHOLE SYSTEM TOGETHER.

EP2S60	DSP blocks (9bits)	RAM (block Mbits)	Reg.	LUT	Max. freq. (MHz)
DoG	0	0.91	7256	15137	149
OriMag	0	0.03	670	1863	184
Kp	48	0.2	2094	14357	52
Whole system	64 (22%)	1.35M (52%)	19100 (37%)	43366 (90%)	-

The hardware blocks are implemented in Handel-C [24] and the Avalon components (DMA channels, FIFO controls and I^2C camera interface) and the camera interface are in VHDL [29]. Table II presents the resources used to implement the system on an Altera Stratix II 2S60 FPGA [30]. Note that *whole system* in the table adds also the NIOS II processor, the camera interface and the Avalon components. As can be seen, the whole system uses approximately all FPGA resources. This is because the DoG block has 18 Gaussian filters using 7x7 kernels and also because the Kp block implements in fixed-point format a considerable number of operations for the stability checks, such as the 3×3 matrix inversion. For the current FPGA practically no extra logic elements have been left for another applications. Nonetheless, it is necessary to consider that nowadays there are newer FPGAs with 3× more elements than this one.

C. Performance Comparison

According to the related work section, there has been few embedded solutions for the SIFT. Additionally, most hardware solutions implement just a part of the SIFT, which are usually used as a component for a general purpose processor. As far we know, the only completed embedded solution is the one presented by Se [16], which takes 60ms to detect features from images of 640×480 pixels. In comparison, our system takes 33ms for images of 320×240 pixels.

The bottleneck of our system is in the descriptor association implemented in the NIOS II processor, which takes 11.7ms per feature. However, independent of the NIOS II performance, our proposed hardware is able to detect any number of features at 30 frames/second. Although the descriptor association was implemented based on the original SIFT descriptor, we believe there is space for new proposals more suitable or customized for embedded applications. For instance, a new solution to reduce the descriptor vector dimension or a simpler solution for applications where images suffer small transformations.

VI. Conclusion

The optimizations proposed in the hardware blocks were fundamental to allow the whole system being embedded on-chip without compromising the result reliability when compared to a PC-based implementation. Other decisions, such as the use of the CORDIC algorithm and the fixed-point format have also a significant influence in the hardware resource optimization. As a result, the system, fully embedded on an FPGA (Field-Programmable Gate Array), detects features up to 30 frames/second (320×240 pixels).

Although the current performance satisfies our feature-based mapping problem, the software used to associate the feature descriptors has the most critical time in the case of needing higher number of features being extracted per frame. The simplest solutions to solve this problem in software are to upgrade the NIOS II softcore processor from a standard to a fast version or to adopt an FPGA with hardcore processor as generally it has better performance than a softcore. On the other hand, the hardware blocks were developed to support any number of features per frame. The only parameter that needs to be adjusted is the internal buffer size between DoG and Kp blocks so as to avoid overflow.

As the proposed system is a stand-alone embedded system and is able to read pixel stream direct from CMOS image sensor, it could be used to create a new featured-based camera where instead of providing just images the camera would also give a set of features along with their descriptors invariant to rotation and scale.

Acknowledgment

The authors would like to thank CAPES (Ref. BEX2683/06-7) and EPSRC (Grant EP/C549481/1 and EP/C512596/1) for the financial support given to this research project. Additionally, the authors are also grateful to the reviewers for their important contribution to improve the paper clarity and for future work suggestions.

References

[1] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover," Ph.D. dissertation, Stanford University, Stanford, 1980.

[2] Q. Wang, W. Zhang, X. Tang, and H.-Y. Shum, "Real-time Bayesian 3-D pose tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 12, pp. 1533–1541, Dec. 2006.

[3] T. Serre, L. Wolf, S. Bileschi, and M. Riesenhuber,"Robust object recognition with cortex-like mechanisms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 411–426, 2007.

[4] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, January 2004.

[5] M. E. Munich, P. Pirjanian, E. D. Bernardo, L. Goncalves, N. Karlsson, and D. Lowe, "Break-through visual pattern recognition for robotics and automation," Evolution Robotics, Inc., Pasadena, CA, 2008 [Online]. Available: http://www.evolution.com/news/icra_innovation_award.pdf

[6] S. Se, D. G. Lowe, , and J. J. Little, "Vision-based global localization and mapping for mobile robots," *IEEE Transactions on Robotics*, vol. 3, no. 21, pp. 364–375, 2005.

[7] T. Braunl, *Embedded Robotics: mobile Robot Design and Applications with Embedded Systems*, 2nd ed. Germany: Springer-Verlag Berlin Heidelberg, 2006.

[8] P. Pirjanian, N. Karlsson, L. Goncalves, and E. D. Bernardo, "Lowcost visual localization and mapping for consumer robotics," *Industrial Robot: an international journal*, vol. 30, no. 2, pp. 139–144, January-February 2003.

[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, USA: MIT Press, 2005.

[10] C. Harris and M. J. Stephens, "A combined corner and edge detector," in *Proceeedings of the Avley Vision Conference*, Manchester (UK), 1988, pp. 147–152.

[11] J. Shi and C. Tomasi, "Good features to track," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Seatle (USA), 1994, pp. 593–600.

[12] A. Shokoufandeh, I. Marsic, and S. J. Dickinson, "View-based object recognition using saliency maps," *Image and Vision Computing*, vol. 17, no. 5-6, pp. 445–460, April 1999.

[13] K. Mikolajczyk and C. Schmid, "Scale and affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, January 2004.

[14] Y. Ke and R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Washington (USA), 2004, pp. 506–513.

[15] M. Grabner, H. Grabner, and H. Bischof, "Fast approximated SIFT," in *Asian Conference on Computer Vision (ACCV)*. Hyderabad: Springer-Verlag, 2006, pp. 918–927.

[16] S. Se, H.-K. Ng, P. Jasiobedzki, and T.-J. Moyung, "Vision based modeling and localization for planetary exploration rovers," in *Proceedings of International Astronautical Congress (IAC)*, Vancouver (Canada), October 2004, p. 11.

[17] N. Pettersson and L. Petersson, "Online stereo calibration using FPGAs," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Las Vegas (USA), June 2005, pp. 55–60.

[18] H. D. Chati, F. Muhlbauer, T. Braun, C. Bobda, and K. Berns, "Hardware/ software co-design of a key point detector on FPGA," in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa Valley (USA), 2007, pp. 355–356.

[19] D. Kim, K. Kim, J.-Y. Kim, S. Lee, and H.-J. Yoo, "An 81.6 gops object recognition processor based on noc and visual image processing memory," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, San Jose (CA), 2007, pp. 443–446.

[20] M. Brown and D. G. Lowe, "Invariant features from interest point groups," in *Proceedings of British Machine Vision Conference (BMVC)*, Cardiff (Wales), 2002, pp. 656–665.

[21] R. Hess, "SIFT feature detector (source code)," 2007, version: 1.1.1. [Online]. Available: http://web.engr.oregonstate.edu/

[22] S. K. Mitra, *Digital Signal Processing: A Computer Based Approach*, 3rd ed. McGraw-Hill, 2004.

[23] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proceedings of the ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, Monterey (USA), 1998, pp. 191–200.

[24] Celoxica, "Handel-C language reference manual," 2005, User guide. [Online]. Available: http://www.celoxica.com

[25] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," *Autonomous robot vehicles*, pp. 167–193, 1990.

[26] A. Davison, I. Reid, N. Molton, and O. Stasse, "MonoSLAM: Realtime single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, June 2007.

[27] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearestneighbour search in high-dimensional spaces," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Washington (USA): IEEE Computer Society, 1997, pp. 1000–1006.

[28] V. Bonato, J. A. d. Holanda, and E. Marques, "An embedded multicamera system for simultaneous localization and mapping," in *Proceedings of Applied Reconfigurable Computing, Lecture Notes on Computer Science - LNCS 3985*. Delft (Netherlands): Springer-Verlag, 2006, pp. 109–114.

[29] S. Brown and Z. Vranesic, *Fundamentals of digital logic with VHDL design*, 2nd ed. Toronto: Mc Graw Hill, 2005.

[30] Altera, "Stratix II device handbook," San Jose, CA: Altera, 2007.

Vanderlei Bonato received the M.Sc. and Ph.D. degrees from the Institute of Mathematical and Computing Sciences, University of Sao Paulo, Brazil, in 2004 and 2008, respectively. His Ph.D. was also partially developed at the Department of Electrical and Electronic Engineering, Imperial College London, London, U.K. His interest areas are image processing and probabilistic algorithms applied to mobile robotics. Recently, he has concentrated his work on developing customized architectures based on reconfigurable computing for navigation and mapping problems. Finally, he has also seven years of industrial experience in automation systems and four years in visual electronic communication projects.

Eduardo Marques has a MSc degree in Computer Science, and a PhD degree in Digital System Engineering, both from the University of São Paulo, Brazil. He is a faculty member of the ICMC-USP since 1986.

Dr. Marques is Associate Editor of the International Journal of Reconfigurable Computing. Recently, he has been involved in the organization of some international events in the area of reconfigurable computing, such as ARC´2007 and the 23rd Annual ACM Symposium on Applied Computing (track Robotics: Hardware, Software, and Embedded Systems). He also serves on the technical program committees of FPL, SPL and JCIS.

During the first years of his academic career his main research interest was on Parallel Machines and Dataflow Computing. Afterwards, he has focused on Reconfigurable Computing, currently working on embedded systems and framework development applied to mobile robotics and education.

George A. Constantinides graduated in 1998 from Imperial College (U.K.) with an M.Eng.(Hons.) in Information Systems Engineering. He then joined the Circuits and Systems group in the Electrical and Electronic Engineering department at Imperial College, where he obtained the Ph.D. in 2001 and joined the faculty in 2002.

Dr. Constantinides is Associate Editor of the IEEE Transactions on Computers and the Springer Journal of VLSI Signal Processing. He was programme co-chair of FPT in 2006 and FPL in 2003, and serves on the technical program committees of FPL, FPT, ISCAS, CODES+ISSS and ARC, for which he is also a member of the steering committee.

His research interests include the theory and practice of reconfigurable computing, electronic design automation, and customised computer architectures. He is particularly focused on numerical algorithms from digital signal processing and optimization. He is the author of over 70 peer-reviewed publications in this area.