Sensors **2010**, *10*, 9232-9251; doi:10.3390/s101009232

OPEN ACCESS

sensors

ISSN 1424-8220

www.mdpi.com/journal/sensors

Article

An Intelligent Architecture Based on Field Programmable Gate Arrays Designed to Detect Moving Objects by Using Principal Component Analysis

Ignacio Bravo *, Manuel Mazo, José L. Lázaro, Alfredo Gardel, Pedro Jiménez and Daniel Pizarro

Electronics Department, University Alcala, Escuela Politecnica, Campus Universitario, Ctra. Madrid Barcelona km. 33.6 28871, Alcala de Henares, Madrid, Spain; E-Mails: mazo@depeca.uah.es (M.M.); lazaro@depeca.uah.es (J.L.L.); alfredo@depeca.uah.es (A.G.); pjimenez@depeca.uah.es (P.J.); pizarro@depeca.uah.es (D.P.)

* Author to whom correspondence should be addressed: E-Mail: ibravo@depeca.uah.es; Tel.: +34-918-856-580; Fax: +34-918-856-540.

Received: 2 September 2010; in revised form: 1 October 2010 / Accepted: 10 October 2010 / Published: 15 October 2010

Abstract: This paper presents a complete implementation of the Principal Component Analysis (PCA) algorithm in Field Programmable Gate Array (FPGA) devices applied to high rate background segmentation of images. The classical sequential execution of different parts of the PCA algorithm has been parallelized. This parallelization has led to the specific development and implementation in hardware of the different stages of PCA, such as computation of the correlation matrix, matrix diagonalization using the Jacobi method and subspace projections of images. On the application side, the paper presents a motion detection algorithm, also entirely implemented on the FPGA, and based on the developed PCA core. This consists of dynamically thresholding the differences between the input image and the one obtained by expressing the input image using the PCA linear subspace previously obtained as a background model. The proposal achieves a high ratio of processed images (up to 120 frames per second) and high quality segmentation results, with a completely embedded and reliable hardware architecture based on commercial CMOS sensors and FPGA devices.

Keywords: FPGA; PCA; CMOS sensor; object detection; image processing

1. Introduction

One of the main research areas in the field of computer vision is the automatic description of the features of a given scene [1,2]. The greater demand made by the performance of image processing algorithms, together with improved spatial resolution and the increased rate of images per second from the new CMOS sensors, means that the need for computational power is continuously increasing. If real time performance is to be achieved, the need to reduce algorithm execution time is even greater, requiring the incorporation of an operating system in the processor capable of executing deterministic tasks, which in turn increases the cost of the products and makes it more difficult to program.

It is usual for the platforms chosen to carry out these algorithms based on sequential programs, in which the only improvements currently available consist in applying multi-threading programming techniques so the power of the new multicore processors may be used. However, from a performance point of view these processing architectures are not so efficient in many applications like the digital processing of images, which normally requires a high number of operations to be handled at the bit level as quickly as possible, by processing in parallel a small number of input samples. Due to the sequential architecture of conventional computers, a notorious amount of operations cannot be performed concurrently. Another issue is the amount of data processed in each instruction, which is limited by the type and width of used communication bus and the image capture board. For this reason, when a large amount of data must be handled, the system performs slowly. This has given rise to solutions that make use of coprocessor systems that handle low level preprocessing tasks, where the amount of data to be processed is high but the operations to be carried out are simple [3]. Our proposal is to create a hardware platform for a specific purpose (designed specifically for one application), as it can produce excellent results working in an *ad-hoc* low-cost platform. In fact the FPGA used to validate the proposal could be considered as a FPGA with medium/low features (Xilinx V2P7).

The detection of both static and moving objects within a captured area is one of the more common tasks undertaken by many computer vision applications. Movement analysis is involved, among other things, in real time applications such as navigation and tracking and obtaining information about static and moving objects within a scene [4]. Movement analysis, which is closely related to the image transfer rate from the video sensor, is fundamental for addressing topics such as image sequence reconstruction, video compression, fixed image capture and multi- resolution, techniques, *etc.*

Within the field of image processing previous works have partially developed the processing algorithm of PCA using programmable devices. In [5] for example, all of the PCA is implemented on the FPGA, however the calculation of eigenvalues is implemented on a PC due to it is mathematically too complex to be implemented on the FPGA. In [6] on the other hand, a variant of PCA called a Modular PCA, applied to face recognition, has been implemented on an FPGA, as this version of PCA has a much lower volume of mathematical operations than the conventional PCA algorithm. In [7] a system based on FPGA is proposed for detecting objects known a priori by comparing their eigenvectors. However, as far as the authors know no work has been found on the detection of moving objects employing PCA that uses FPGAs as the processing element. It is important to point out that in none of the works found is PCA implemented exclusively on FPGAs, due mainly to the heavy data dependence and complex mathematical operations involve within PCA. The data dependences cause several hazards which make difficult the implementation of efficient pipeline systems. On the other hand, the mathematical operations needed by PCA algorithm, are not usual operations used for other algorithms (e.g., solving eigenproblems). Due to this fact, new specific mathematical cores have been designed for this algorithm.

These situations make difficult to segment/divide the hardware processing of the different parts of PCA. For this reason, executing PCA is normally divided between an FPGA and a PC or microprocessor [5], so that normally an *ad-hoc* HW/SW partition of the system is made, without adequately exploring the design space (HW/SW co-design methods).

One of the main contributions of this work is the FPGA implementation of the complete PCA algorithm on reconfigurable hardware; indeed it is the first work in the literature to do so. Classic sequential execution of different parts of the PCA algorithm has been parallelized. This parallelization has led to the development and implementation of seldom used alternatives for the different stages of PCA. One example is the calculation of eigenvalues and eigenvectors, matrix multiplication in hardware or calculation of a dynamic threshold for detecting moving objects. This latter issue is another major contribution of the paper because the information generated by PCA is used to detect moving objects. In this work, PCA is implemented on an FPGA to detect moving objects within a scene, based on the PCA algorithm. To achieve this, a specifically designed intelligent camera has been implemented based on a CMOS sensor and an FPGA [8]. Thanks to the design and implementation of this new proposal it can be used in any situation requiring an autonomous system (without PC).

The other sections of this paper are as follows: Section 2 sets out the mathematical foundations of the PCA algorithm applied to image processing; Section 3 describes the platform design; Section 4 presents the implementation in VHDL of the PCA algorithm on an FPGA; and finally, Sections 5 and 6 set out the results and present the conclusions respectively.

2. The PCA Algorithm

Principal Component Analysis (PCA) is a method that is used in different fields, such as statistics, power electronics or artificial vision. The main feature of PCA is the reduction of redundant information, retaining only information that is fundamental (principal components).

Artificial vision is a good example of a field where the PCA technique can be applied directly, as an image contains a large number of highly correlated variables (pixels). Therefore, applying the PCA technique to image processing allows us to reduce the redundant information of the initial variables and determine the degree of similarity between two or more images by analyzing only the basic features within the transformed space. This last feature is of interest as far as the detection of new objects within the scene is concerned.

2.1. Obtaining the principal components of an image

The PCA algorithm can be applied to images using the following steps [9,10]:

1. *Capturing M images to construct a reference model of the scene.* We identify each of M references image by $\mathbf{I}_i \in \Re^{N \times N}$, with $i = 1,..M$ and where it is assumed that the spatial resolution of the images is N x N.
2. Each image is represented as a column vector of the dimensions $N^2 \times 1$

3. Calculating the mean image from the M reference images : $\mathbf{\Psi} \in \Re^{N^2 \times 1}$ given for:

$$\mathbf{\Psi} = \frac{1}{M} \cdot \sum_{i=1}^{M} \mathbf{I}_i = \begin{bmatrix} \frac{I_{1,1} + I_{2,1} + \ldots + I_{M,1}}{M} \\ \frac{I_{1,2} + I_{2,2} + \ldots + I_{M,2}}{M} \\ \cdots \\ \frac{I_{1,N^2} + I_{2,N^2} + \ldots + I_{M,N^2}}{M} \end{bmatrix} = \begin{bmatrix} \Psi_1 \\ \Psi_2 \\ \cdots \\ \Psi_{N^2} \end{bmatrix}_{N^2 \times 1} \quad ; i = 1,\ldots, M \tag{1}$$

where $\mathbf{I}_{i,j}$ is the j ($j = 1,\ldots, N^2$) element of $\mathbf{I}_i$ image.

4. Form a matrix $\mathbf{A} \in \Re^{N^2 \times M}$ (3) whose columns are the vectors $\mathbf{\Phi}_j = \mathbf{I}_j - \mathbf{\Psi}$ (2):

$$\mathbf{\Phi}_i = \mathbf{I}_i - \mathbf{\Psi} = \begin{bmatrix} I_{i,1} & -\Psi_1 \\ I_{i,2} & -\Psi_2 \\ \multicolumn{2}{c}{\ldots\ldots\ldots\ldots\ldots} \\ I_{i,N^2} & -\Psi_{N^2} \end{bmatrix}_{N^2 \times 1} = \begin{bmatrix} \Phi_{i,1} \\ \Phi_{i,2} \\ \ldots\ldots \\ \Phi_{i,N^2} \end{bmatrix}_{N^2 \times 1} \quad ; \ i = 1,\ldots, M \tag{2}$$

$$\mathbf{A} = \left[\mathbf{\Phi}_1 \ldots \mathbf{\Phi}_M\right] = \begin{bmatrix} \Phi_{1.1} & \Phi_{2,1} & \ldots & \Phi_{M,1} \\ \Phi_{1,2} & \Phi_{2,2} & \ldots & \Phi_{M,2} \\ \multicolumn{4}{c}{\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots} \\ \Phi_{1,N^2} & \Phi_{2,N^2} & \ldots & \Phi_{M,N^2} \end{bmatrix}_{N^2 \times M} \tag{3}$$

5. *Obtaining the covariance matrix,* $\mathbf{C} \in \Re^{N^2 \times N^2}$ from the matrix **A** (4):

$$\mathbf{C} = \frac{1}{M} \cdot \mathbf{A} \cdot \mathbf{A}^T \tag{4}$$

6. *Obtaining the associated eigenvalues and eigenvectors of the matrix* **C**. Given that matrix **A** is of the size $N^2 \times M$ and generally $N^2 >> M$, to reduce the number of operations that must be performed, the eigenvalues and eigenvectors of $\left(\mathbf{A}^T \cdot \mathbf{A}\right)$ are calculated (5):

$$\mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{V} = \lambda I \cdot \mathbf{V} \tag{5}$$

where $\mathbf{V} \in \Re^{M \times M}$ is the matrix of the eigenvectors of $\mathbf{A}^T \cdot \mathbf{A}$. The eigenvalues of **C** match up with those of $\mathbf{A}^T \cdot \mathbf{A}$ while the eigenvectors of **C** are obtained from (6):

$$\mathbf{U} = \mathbf{A} \cdot \mathbf{V} \tag{6}$$

7. *Obtaining the principal eigenvalues.* From the eigenvalues obtained in point 6 the most significant eigenvalues t are selected, using for example, the criteria the normalized root mean square error (RMSE) [10,11] given by (7), that is the eigenvalues of greatest value ($\lambda_1 > \lambda_2 > \ldots > \lambda_t$):

$$RMSE = \frac{\sum_{i=t+1}^{M} \lambda_i}{\sum_{i=1}^{M} \lambda_i} < P \tag{7}$$

where *P* is the percentage of necessary eigenvalues required to achieved the most significant eigenvalues *t*.

The transformation matrix $\mathbf{U}_t \in \Re^{N^2 \times t}$ is given by (8) where $[\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t]$ are the eigenvectors associated to the eigenvalues $\lambda_1 > \lambda_2 > ... > \lambda_t$:

$$\mathbf{U}_t = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t] \tag{8}$$

An important issue is the quantification of the value M, which is the number of captured reference images used to build the background. Theoretically, it is a good idea to employ a high *M* value that allows different lighting conditions of the same scene to be considered. However the use of a high *M* number implies a significant increase in computational load and memory storage. The features of external memory in which background images will be loaded, will determinate the size of M. Due to this fact, the bus width of used external memory (128 bits) and according to the results shown in [12] in our case, it has been chosen size of $M = 8$ associated to the same scene without moving obstacles and under soft natural lighting variations. Thanks to this size and CMOS features, it is possible to read from the sensor 8 pixels in each clock period.

According to the results shown in [12] in our case, a size of $M = 8$ has been chosen. Once the transformation matrix $\mathbf{U}_t$ has been obtained, the next step is to determine whether in a newly captured image of the scene new objects have appeared. To do this the following steps must be performed:

1. *Projection onto the transformed space*. The first step is the projection onto the transformed space using (9):

$$\mathbf{\Omega} = \mathbf{U}_t^T \cdot \mathbf{\Phi}_j = \mathbf{U}_t^T \cdot (\mathbf{I}_j - \mathbf{\Psi}) \tag{9}$$

where $\mathbf{\Omega} \in \Re^{t \times 1}$ is characterized by a vector of dimension t ($\mathbf{\Omega} = [\omega_1, \omega_2, ..., \omega_t]^T$), where each component ω_i represents the contribution of each eigenvector in the representation of $\mathbf{I}_j$.

2. *Recovering the projected image*. Once the image has been projected onto the transformed space using (9), then $\hat{\mathbf{\Phi}} \in \Re^{N^2 \times 1}$ is recovered using (10):

$$\hat{\mathbf{\Phi}}_j = \mathbf{U}_t \cdot \mathbf{\Omega} \tag{10}$$

3. *Determining the existence of new objects in the scene*. Finally, the captured image is compared with the recovered image, thus obtaining what is termed the error recovery. If the result of the comparison is above a determined threshold (Th_{MD}) it implies the presence of new objects in the scene (11):

$$\begin{aligned} \left\| \mathbf{\Phi}_j - \hat{\mathbf{\Phi}}_j \right\| \le Th_{MD} &\rightarrow \text{there are no new objects in the scene} \\ \left\| \mathbf{\Phi}_j - \hat{\mathbf{\Phi}}_j \right\| > Th_{MD} &\rightarrow \text{there are new objects in the scene} \end{aligned} \tag{11}$$

The threshold Th_{MD} is a dynamically obtained value that is adjusted according to the conditions of the scene.

4. *Spatial localization of the detected object*. To detect the presence of new objects it is only necessary to apply expression (11). However, if we want to know in which part of the scene the new object has appeared a localization method must be found.. With the aim of reducing the effect of noise, the value of each pixel of the captured image and that of the recovered image is averaged with that of the adjacent pixels by means of a mask of $q \times q$ elements. As a result a matrix known as an average distance map is obtained ($\mathbf{MD}_V \in \Re^{N^2 \times N^2}$), where every one of its elements ($\varepsilon_{w,i}$) corresponds to the Euclidean distance between the corresponding average pixels of the original and recovered image (12).

$$\varepsilon_{w.i} = \left\| \mathbf{\Phi}_{w,i} - \hat{\mathbf{\Phi}}_{w,i} \right\| \quad w = 1,2,\ldots N \quad i = 1,2,\ldots.,N \tag{12}$$

Once the $\mathbf{MD}_V$ has been obtained, the next step is to threshold the map so that the new objects can be found easily. To do this, a new binary image is built $(\mathbf{BW})$ where each element is the result of a comparison between each $\mathbf{MD}_V$ pixel and a Th_{MD} threshold (13):

$$\begin{aligned} BW_i = 255 \quad \text{if} \quad (\varepsilon_{wi}) = \left\| \mathbf{\Phi}_{wi} - \hat{\mathbf{\Phi}}_{wi} \right\| > Th_{MD} \\ BW_i = 0 \quad \text{if} \quad (\varepsilon_{wi}) = \left\| \mathbf{\Phi}_{wi} - \hat{\mathbf{\Phi}}_{wi} \right\| \le Th_{MD} \end{aligned} \tag{13}$$

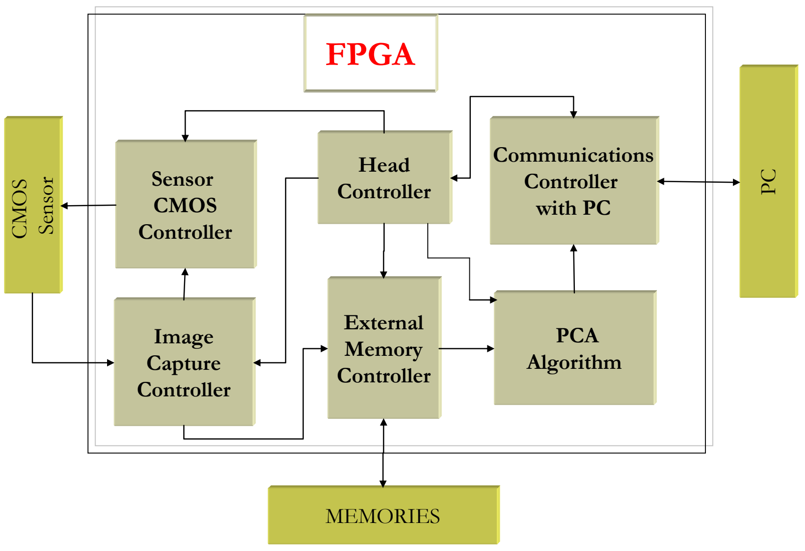
3. Description of the Architecture

The system proposed is based on a high speed CMOS sensor (up to 500 images per second with a maximum resolution of 1,280 × 1,024 [13]) and an FPGA in which a novel design has been implemented for managing and capturing the images from the sensor, as well as executing the PCA algorithm. The system implemented on the FPGA is separated into the following logical blocks as shown in Figure 1 with green color:

- *CMOS sensor controller*: This block is responsible for implementing image demands to the CMOS sensor such as parameterizing its internal registers according to the desired configuration (images per second, exposure time, *etc.*).
- *Image capture controller*: the purpose of this block is to allow the user to select an area of interest within the image from the CMOS sensor.
- *External memory controller*: the system is equipped with a 128 MB, SDRAM memory bank that is external to the FPGA. Images from the CMOS sensor are stored in this bank.
- *Communications Controller with the PC*: this block controls the communication between the FPGA and the PC. This is used to transmit commands and results.
- *Head Controller*: This block is responsible for synchronizing the entire system so that everything works correctly and at maximum speed.

- *PCA algorithm*: This block implements the PCA algorithm and its implementation is the most important contribution of this work.

Figure 1. Block diagram of the internal architecture of the FPGA.



4. Implementing PCA on FPGAs

The mathematical complexity of the operations of the PCA algorithm presented in Section 2 (calculation of eigenvectors, matrix multiplication, square roots, etc) makes it impractical to implement them directly on reconfigurable hardware. The proposal and selection of different hardware structures and computing alternatives in order to obtain an efficient solution to resolve these operations on FPGAs is essential for the PCA implementation and, thus, constitutes one of the major contributions of this paper. This section presents the hardware solution found which permits the PCA algorithm to be implemented on an FPGA. Figure 2 shows a block diagram of the PCA algorithm implemented on the FPGA, grouping the different modules into three stages: generation of eigenvectors (light yellow), on-line (light orange), and object detection (light pink). The three phases in which the PCA algorithm is divided are now described.

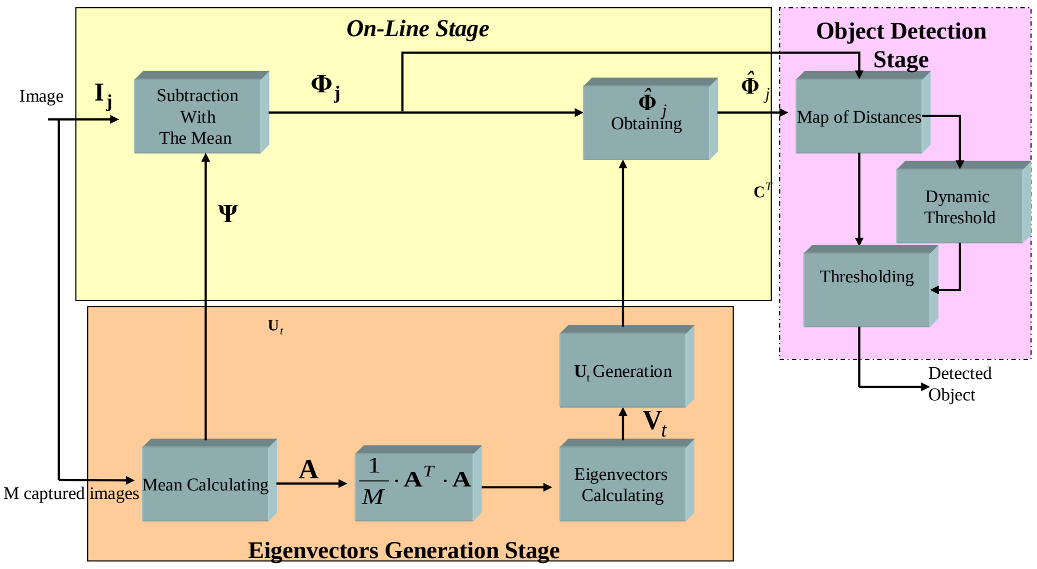
4.1. Generating the eigenvectors

The first phase of the PCA algorithm is the generation of the eigenvectors of the reduced transformation matrix $\mathbf{U}_t$. This first phase includes five stages:

1. Calculating the mean of the M images $\left(\mathbf{\Psi} \in \Re^{N^2 \times 1}\right)$ and the matrix $\mathbf{A} \in \Re^{N^2 \times M}$ (3).
2. Obtaining the covariance matrix $\mathbf{C}^T \in \Re^{M \times M}$ (5).
3. Calculating the eigenvectors of the matrix $\mathbf{V} \in \Re^{M \times M}$ and the posterior matrix of the reduced eigenvectors $\mathbf{V}_t \in \Re^{M \times t}$.
4. Obtaining the eigenvectors of the matrix $\mathbf{U}_t \in \Re^{N^2 \times t}$ from $\mathbf{V}_t \in \Re^{M \times M}$ where $t < M$.

5. Calculating the norms of matrix eigenvectors.

Figure 2. Block diagram of the PCA algorithm implemented on an FPGA.



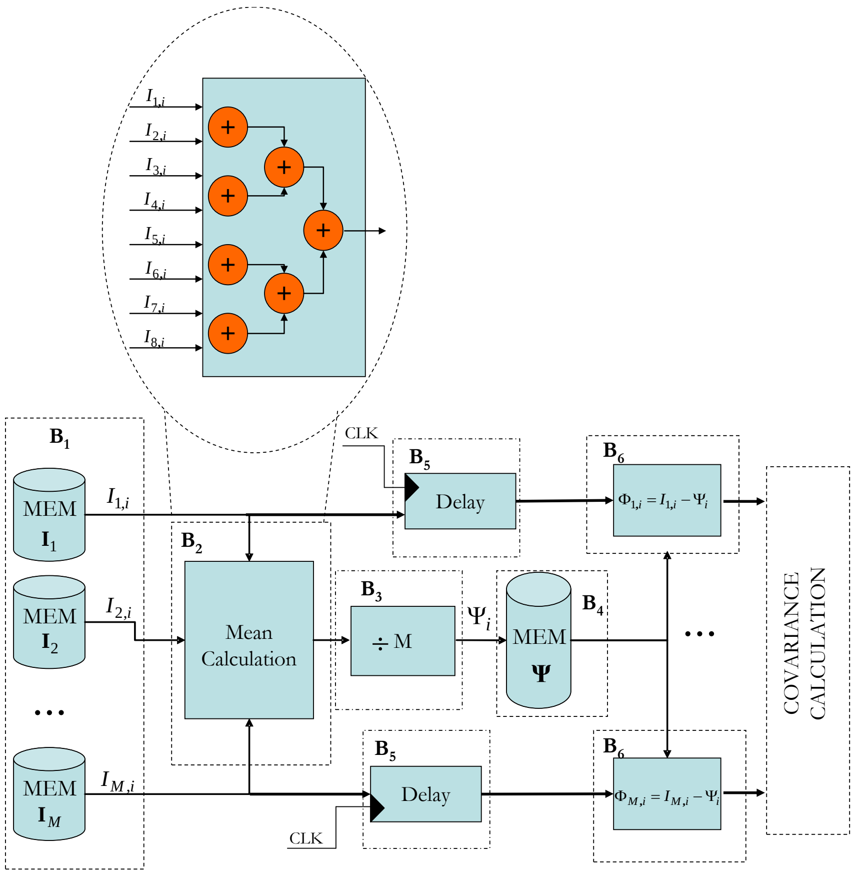
4.1.1. Calculating the mean of the M images $(\mathbf{\Psi})$

The hardware architecture that has been developed for this module stores the captured M images in an SDRAM external memory. The block shown in Figure 3 has been implemented on the FPGA, where the $M = 8$ images are stored in different memory components (B_1). Once the eight pixels have been extracted, one for each image, the mean calculation process is initiated using a set of cascade adders (see B_2 Figure 3). As this process takes three clock cycles and the aim is for the system to be as segmented as possible, the eight extracted pixels are inserted into a delay unit consisting of flip-flops that synchronizes the subtraction process of each pixel with that of the corresponding mean (B_6).

4.1.2. Obtaining the covariance matrix $(\mathbf{C}^T)$

Generating $\mathbf{C}^T$ from matrix $\mathbf{A}$, means the product of two matrices $\mathbf{A}^T \cdot \mathbf{A}$, must be produced on an FPGA, which entails a complex process. In the case of the PCA the aim is to multiplex the matrix multiplication module using it to: generate the covariance matrix, generate the eigenvector matrix ($\mathbf{U}_t$), project an image onto the transformed space and recover the projected image. Different approaches to the matrix multiplication have been analyzed and developed by the authors [14]. After this study, an *ad-hoc* matrix multiplier system based on a semi-systolic array proposed by the authors in [14] has been chosen because the maximum performance for PCA is achieved with this approach thanks to the possibility to reuse the system for the different types of matrix multiplication that PCA needs.

Figure 3. Block diagram of the proposed circuit for calculating the mean (Ψ) of the M captured images.



4.1.3. Calculating the eigenvectors of the matrix $(\mathbf{V})$

The computation of eigenvalues and eigenvectors represents the greatest computational burden on the PCA algorithm. Different techniques have been proposed for obtaining the eigenvalues of a matrix using specific hardware, all of them based on recurrent methods that look to diagonalize the matrix [15,16]. Once the matrix has been diagonalized, the eigenvalues coincide with the values of the diagonal. The method proposed in [17] is the most interesting as it allows parallel processing hardware structures to be implemented [18]. For this reason, the solution developed in this work is based on the Jacobi method. A previous article by the authors [19] describes the architecture developed.

4.1.4. Obtaining the eigenvectors of the matrix $(\mathbf{V}_t)$

The first step in determining the most significant t eigenvectors is to arrange the eigenvalues and their associated eigenvectors in either ascending or descending order. This step is necessary as the Jacobi method does not generate the eigenvalues in order. To determine t, the largest t eigenvalues are found and then their associated eigenvectors are selected depending on how much bigger than the eigenvalues that have been obtained the user wants them to be (7). In this work, *bubble sort* has been used as the sorting algorithm [20].

4.1.5. Obtaining the eigenvectors of the matrix $\mathbf{U}_t$

To obtain the matrix $\mathbf{U}_t$, according to (6), the matrix $\mathbf{A}$ must be multiplied by $\mathbf{V}_t$. To do this, once again the semi-systolic *array* presented in [14] is used.

4.1.6. Calculating the norms of the eigenvectors

The eigenvectors obtained in the previous stage do not possess a unit module so they must be normalized (14) $\mathbf{U}_{tn}$ according to (15):

$$n_j = \sqrt{\sum_{i=1}^{N^2} \left(u_{\mathbf{i,j}}\right)^2} \quad j = 1,\ldots,t \tag{14}$$

$$\mathbf{U}_{tn} = \frac{\mathbf{U}_t}{\mathbf{norms}} = \left(\frac{\mathbf{u}_1}{n_1}\frac{\mathbf{u}_2}{n_2}\cdots\frac{\mathbf{u}_t}{n_t}\right) \qquad \mathbf{norms} = \left[n_1, n_2, \ldots, n_t\right] \in \Re^{1\times t} \tag{15}$$

To implement in hardware the arithmetical operations shown in expressions (14) and (15) is extremely complex as a consequence of the square root, and it also uses a large amount of resources. To avoid calculating the square root when calculating $\hat{\mathbf{\Phi}}_j$ it is only necessary to express this matrix in accordance with the squared norm, as shown in (16):

$$\hat{\mathbf{\Phi}}_j = \mathbf{U}_{tn} \cdot \mathbf{\Omega} = \mathbf{U}_{tn} \cdot \mathbf{U}_{tn}^T \cdot \mathbf{\Phi}_j = \frac{\mathbf{U}_t \cdot \mathbf{U}_t^T \cdot \mathbf{\Phi}_j}{\mathbf{norms}^2} \tag{16}$$

4.2. The on-line stage

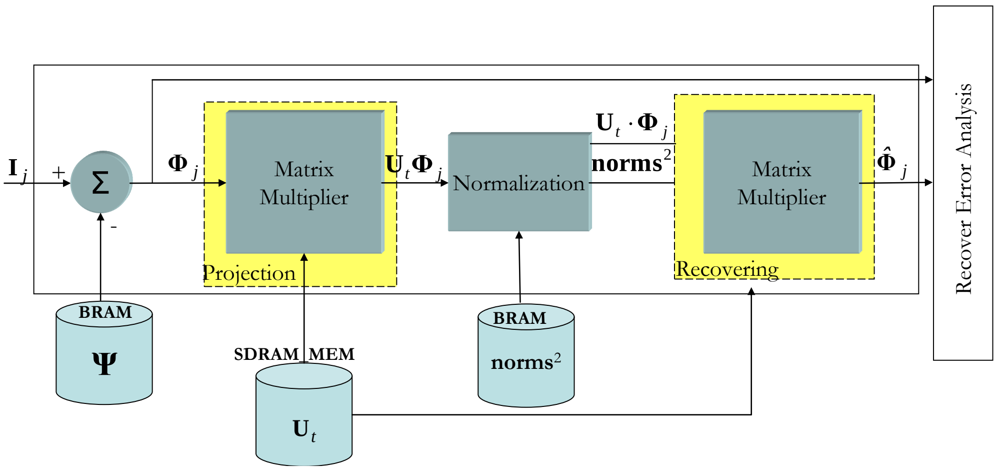
If there is a new object in the captured image, with respect to the reference scene, it is determined during the *on-line* stage. For this to be done, the new captured image is projected onto the transformed space so that it can be recovered later and studied to determine whether or not there is a new object in the scene. To do this the following steps are followed:

1. *Subtraction of the mean of the present image*: If $\mathbf{I}_j$ is the captured image, $\mathbf{\Phi}_j$ is obtained.
2. *Projecting $\mathbf{\Phi}_j$ onto the transformed space and obtaining $\hat{\mathbf{\Phi}}_j$*: With the aim of reaching the maximum concurrence possible when executing (16), first the $\mathbf{U}_t^T \cdot \mathbf{\Phi}_j$ product is performed and this result is divided by the squared norms ($\frac{\left(\mathbf{U}_t^T \cdot \mathbf{\Phi}_j\right)}{\mathbf{norms}^2}$), and finally $\mathbf{U}_t \cdot \left(\frac{\left(\mathbf{U}_t^T \cdot \mathbf{\Phi}_j\right)}{\mathbf{norms}^2}\right)$ product is performed.
3. *Determining the recovery error*: In this final stage, the degree of similarity between $\mathbf{\Phi}_j$ and $\hat{\mathbf{\Phi}}_j$ is evaluated.

Figure 4 shows the VHDL encoded modular design of this *on-line* stage. With respect to the internal workings of the system shown in Figure 4, this starts when a new vector image $\mathbf{I}_j$ is captured and later stored in the external memory so that the system has an initial latency of one image. As explained earlier, once $\mathbf{\Phi}_j$ has been obtained the next step is to produce the $\mathbf{U}_t^T \cdot \mathbf{\Phi}_j$. To do this, the semi-systolic array for matrix multiplication is used [14]. It is important to point out at this point that the execution time of the Matrix Multiplier depends on the number of significant eigenvalues (t). In accordance with the percentage of significant eigenvalues (see (7)), a value of t equals 6 has been decided upon. This reduction introduces a recovery error (ε) (17), after analyzing 1,000 images it could be seen that the induced error is approximately 1%:

$$\varepsilon = \left\| \mathbf{\Phi}_j - \hat{\mathbf{\Phi}}_j \right\| = \sqrt{\sum^{N^2} \left(\Phi_{j_i} - \hat{\Phi}_{j_i} \right)^2} \tag{17}$$

Figure 4. Block diagram of the modules of the design in VHDL of the on-line stage of the PCA.



Once the first results from the $\mathbf{U}_t^T \cdot \mathbf{\Phi}_j$ product have been obtained, the next step is to divide these results by **norms²**. As each component of $\mathbf{U}_t^T \cdot \mathbf{\Phi}_j$ is generated in one clock cycle, given that they are output by the semi-systolic array, they are divided by the corresponding squared norm.

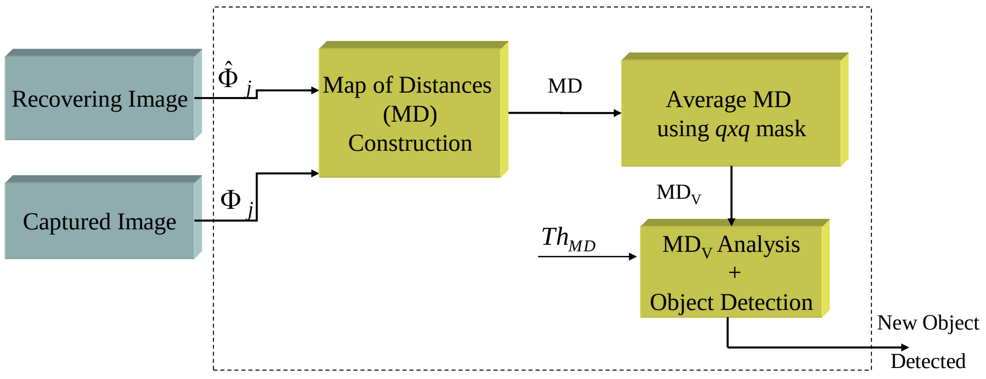
To perform the division operation on an FPGA, there are basically two possibilities: either design a division unit specifically for that purpose, or use a coordinate rotation digital computer (CORDIC) algorithm [21]. In this work the latter option has been chosen, as it consumes fewer resources than the former. Dividing two numbers is feasible in CORDIC if it is used in vectorization mode with a linear coordinate system [22]. To do so, a division module based on a parallel CORDIC architecture has been implemented.

When the first component of the division has been obtained, the next step to be performed in (16) is to obtain $\hat{\mathbf{\Phi}}_j$. Once again, to perform this fourth matrix multiplication, the semi-systolic array described in [14] is used.

4.3. Detecting new objects in the scene

This section presents the solution developed for implementing an identification of new objects from the error recovery (ε) system in reconfigurable hardware (17). It proposes the building of an error recovery map or Map of Distances ($\mathbf{MD}$) that will permit the new objects to be located spatially. The size of this map of distances will coincide with the size of the image, where each of its positions is the pixel to pixel Euclidean distance between $\hat{\mathbf{\Phi}}_j$ and $\mathbf{\Phi}_j$. A new Map of Distances ($\mathbf{MD}_V$) will be built in order to reduce the noise effect. The final detection of moving obstacles will be obtained using the dynamic threshold Th_{MD} (11). Calculating Th_{MD} presents difficulties as it must be adaptable and its value depends on both the features of the scene under analysis and the lighting conditions. For this reason, in this section we present a new method for dynamically calculating the threshold that minimizes the false detection of new objects within the scene of interest. Figure 5 shows a block diagram of this proposal for detecting objects from $\hat{\mathbf{\Phi}}_j$ and $\mathbf{\Phi}_j$ (green blocks). Next the hardware solution implemented in each block of Figure 5 is presented.

Figure 5. Proposal for the system consisting of the construction of the MD, detection of new objects and the updating of the background model.


Recovering Image
Captured Image
$\hat{\Phi}_j$
Φ_j
Map of Distances (MD) Construction
MD
Average MD using *qxq* mask
MD$_V$
Th_{MD}
MD$_V$ Analysis + Object Detection
New Object Detected

4.3.1. Constructing the Map of Distances (MD) and the Map of average Distances (MD$_V$)

The Map of Distances **MD** is obtained from (18), ε'_i being the square of the Euclidean distance between each component $\Phi_{ji} \in \mathbf{\Phi}_j$ and $\hat{\Phi}_{ji} \in \hat{\mathbf{\Phi}}_j$ $i = 1,\ldots,N^2$ (for images of the size $N \times N$):

$$\varepsilon'_i = \left\| \Phi_{j_i} - \hat{\Phi}_{j_i} \right\|^2 \quad i = 1,\ldots,N^2 \tag{18}$$

Working with the square of the Euclidean distance rather than the Euclidean distance (17), facilitates the design of hardware associated with this function, as it avoids the need to perform the square root operation. As such, to obtain **MD** only requires one subtraction and one multiplication operation, so that with an adder/subtraction block and a multiplier connected in cascade the segmented execution of (18) can be performed.

Once the initial components of **MD** have been generated, the generating of the map of average distances ($\mathbf{MD}_V$) can be started. The use of a mask of $q \times q$ components is proposed that averages the pixels adjacent to **MD**, applying a 2D low-pass filter. The components that make up the map $\mathbf{MD}_V$ are $\varepsilon'_{v_i,w}$; $i, w = 1,..,N$.

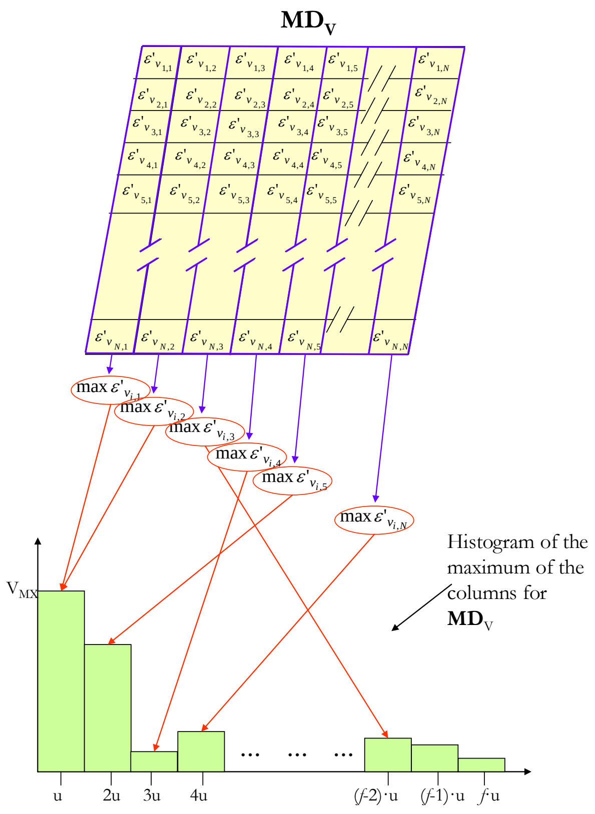
To provide a compromise value to the size of mask *q*, different sizes applied to different maps **MD** have been simulated; all of them are fixed point encoded. The size chosen for *q* is 3, given that it provides the algorithm with a certain degree of robustness and reliability, and few hardware resources are required.

To implement the averaging function with masks of $q \times q$ (3 × 3) on the adjacent pixels the corresponding convolution function is implemented [23]. To select the best alternative for hardware implementation, several proposals for convolutions have been designed [23], evaluating at all times the execution time as well as how much of the FPGA’s internal resources are consumed.

To perform the convolution between a matrix and a generic mask, nine multiplication operations and eight accumulation operations must be performed for each resulting component. However, when all the coefficients of the mask have been identified, as happens in our case, another way of performing the convolution is according to (19), whereby one that reduces the number of multiplications to one. In this way, to obtain each $\varepsilon'_{v_{i,w}}$ component of the $\mathbf{MD}_V$, it is necessary to perform a nine component sum backlog and one multiplication for the equivalent factor in fixed point:

$$\varepsilon'_{v_{i,w}} = 1/9 \cdot \left(\varepsilon'_{i-1,w-1} + \varepsilon'_{i-1,w} + \varepsilon'_{i-1,w+1} + \varepsilon'_{i,w-1} + \varepsilon'_{i,w} + \varepsilon'_{i,w+1} + \varepsilon'_{i+1,w-1} + \varepsilon'_{i+1,w} + \varepsilon'_{i+1,w+1}\right) \tag{19}$$

Figure 6. Example of histogram construction of the maximum of the columns for an average map of distances ($\mathbf{MD}_V$).



4.3.2. Detecting objects from the $\mathbf{MD}_V$ map

Once the map of average distances ($\mathbf{MD}_V$) has been obtained, the next step is to analyze the map to evaluate whether or not there are new objects in the scene of interest. To do so, a threshold Th_{MD} is obtained, which, when applied to $\mathbf{MD}_V$ makes it possible to perform the segmentation and as a consequence detect the presence of new objects. The value of Th_{MD} must be dynamic as its value must adapt, amongst other factors, to changes in light within the scene. In order to obtain this dynamic Th_{MD} different alternatives have been proposed, [12,24,25]. Our proposal calculates the histogram (with f intervals) of the maximum Euclidean distances of each column of the $\mathbf{MD}_V$ (Figure 6) and then obtains the dynamic threshold Th_{MD} from the histogram. This algorithm, implemented on an FPGA, generates excellent results, as will be seen later in the results section.

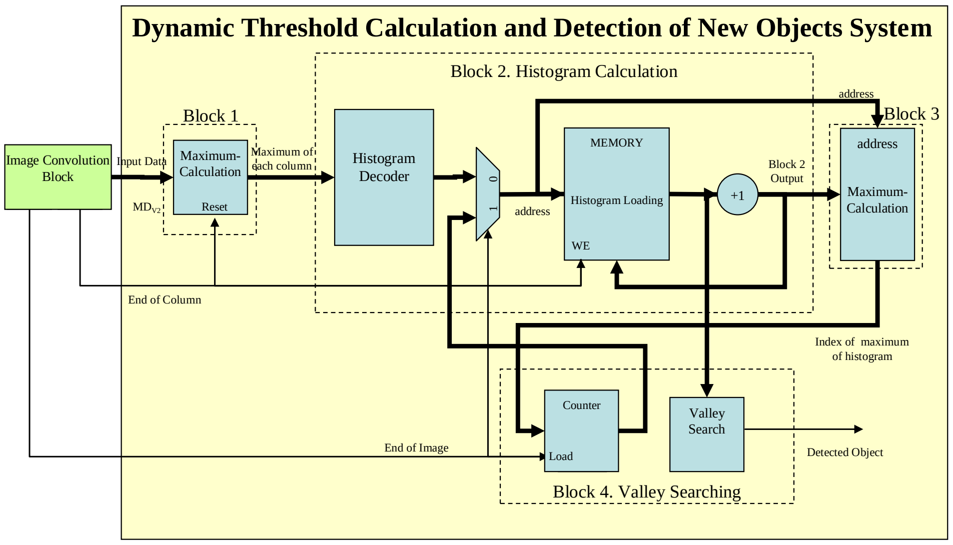
Analyzing the information supplied by the histogram on the maximums of the $\mathbf{MD}_V$ columns, it can be seen how most of the maximum Euclidean distances represented are concentrated in the lower intervals. However, when a new object appears in the scene being studied, the maximum Euclidean distances of the $\mathbf{MD}_V$ columns where the object is located are expressed by a valley in the histogram. If there is no new object in the scene, then the valley does not appear. On the basis of this last feature of the histogram, to threshold $\mathbf{MD}_V$ it is necessary to find the value of Th_{MD} that makes it possible to discriminate between the new object and the background. The minimum value of Th_{MD} needed to correctly detect new objects must be the same as the value of the histogram interval that contains the valley associated with the new object.

The hardware to perform the threshold is shown in Figure 7. Each block in Figure 7 is described below:

- *Block 1*: this block is responsible for calculating the maximum of each column of the map of distances $\mathbf{MD}_V$. Internally it consists of a single register that stores the maximum value and a comparator that evaluates whether the new data is bigger or smaller than the stored temporal maximum.
- *Block 2*: After calculating the maximums of the columns of the $\mathbf{MD}_V$, *Block 2* is responsible for building the histogram of the maximums of the columns. It is executed in parallel with *Block 1* once the maximum of the first column has been obtained. Every time a maximum is obtained the histogram interval that belongs to that maximum must be looked for and its accumulator increased by 1.
- *Block 3*: This module, which is executed when *Block 2* generates the first data, is responsible for calculating the maximum values of the histogram (V_{MX} of Figure 6). This block works as follows: every time the maximum of a column is obtained in *Block 2*, a new value is added to the corresponding histogram interval and the number of the histogram interval with the maximum accumulated value is updated. At the same time, in *Block 3* the increased value is evaluated to see whether it is the largest. If it proves to be so, then it is stored so that it can be compared with the following output from *Block 2* and its memory address, which gives the location of the new maximum, generated by *Block 2* is also stored.
- *Block 4*: Finally, this component is responsible for looking for valleys in the histogram once Block 2 and Block 3 have finished.

To find a valley, a hardware block has been designed to check the memory of *Block 2*, which contains the histogram of the maximums of the columns of $\mathbf{MD}_V$. The counter starts from the address stored in *Block 3*, that is to say, the address of the histogram interval with the maximum accumulated value. To find a valley, it is only necessary to find a value in the memory that is bigger than the one stored in the position before it. If no local minimum exists the system will increase the threshold (checking the intervals defined by the histogram) until it considers that the threshold is situated in the extreme interval and then classifies all the pixels in the image as belonging to the background. The number of histogram intervals (f) has been empirically set at 10, as with this value the developed proposal works correctly.

Figure 7. Block diagram on an FPGA of the dynamic threshold calculating system for detecting new objects.



5. Results

This section sets out the results obtained in detecting new objects with a FPGA running PCA algorithm. All the images presented in this work have been captured by an "intelligent camera" described in [8].

From a quantitative point of view, in calculating the execution time of the entire proposal presented in this work (T_{PCA_TOTAL}) from the moment the first M images are captured, the total time consumed is given by (20), with Table 1 giving a description of each of the times in (20):

$$T_{PCA\_TOTAL} = T_{GEN\_WR\_U} + T_{IMAGE} + L_{MEM} + T_{OBJ} \qquad (20)$$

Table 1. Description of the partial times of T_{PCA_TOTAL}.

$T_{GEN\_WR\_U}$	Time the FPGA takes to generate and write in SDRAM the eigenvectors of the matrix $\mathbf{U}_t$.
T_{IMAGE}	Time employed in capturing a new image and its subsequent writing in SDRAM.
L_{MEM}	Latency of the SDRAM memory, from the time it gives the order to read an image until the first data is received.
T_{OBJ}	Time consumed in detecting new objects after the recovered image ($\hat{\mathbf{\Phi}}_j$) has been obtained from the transformed space

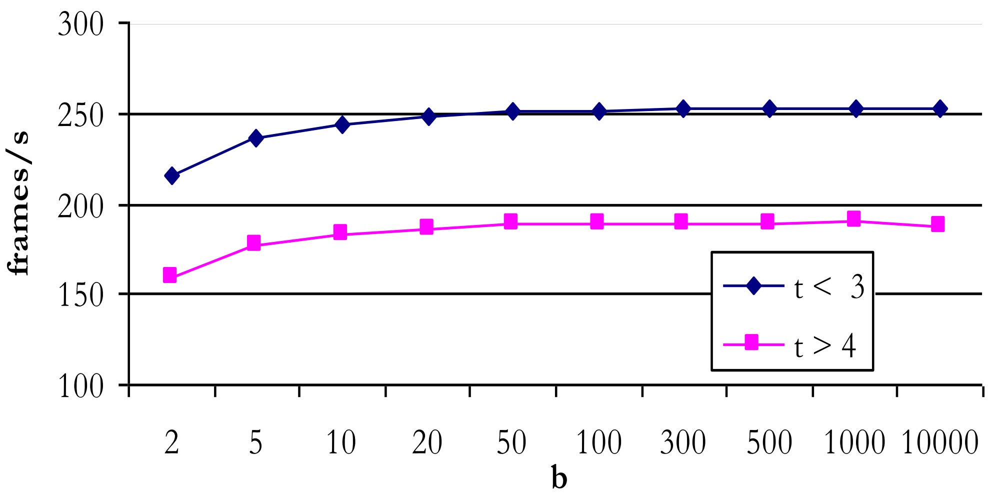
When it comes to calculating the number of complete clock cycles employed by T_{PCA_TOTAL}, the value obtained is not constant as it depends on the number of significant eigenvectors, the size of the matrix and the number of Jacobi algorithm iterations, as explained in [19]. Adjusting the expression (20) for six eigenvectors (worst case), capturing eight images (256 × 256 pixels) to build a reference model ($M = 8$), an internal data width of 18 bits ($n = 18$) and 23 iterations for the Jacobi algorithm the value obtained in clock cycles is:

$$T_{PCA\_TOTAL} = 131076\, T_{CLK\_CAMERA} + 526939\, T_{CLK} \qquad (21)$$

where T_{CLK_CAMERA} is the signal period of the CMOS sensor's clock and T_{CLK} the FPGA's master clock. Clock Camera is generated by the FPGA using a DCM (digital clock management) block. Thanks to this element and a bank register managed for a FSM (finite state machine), both clocks working rightly. To obtain a ratio of the number of images the system processes, if the CMOS sensor's clock (T_{CLK_CAMERA}) is 66 MHz and the FPGA's master clock is 100 MHz (frequency reached once the entire system has been implemented) a minimum of 121 images of 256 × 256 pixels have been processed per second. This ratio increases notably if any of the following situations occur:

- *Number of significant eigenvectors (t) under four*. In this case the number of matrix multiplication operations (6), (9) and (10) are notably reduced. In this way the new T_{PCA_TOTAL} value would reach an equivalent image per second ratio of 189.
- *Selective actualization*. Cadence is another very important factor that conditions the number of images processed per second when updating the eigenvectors of the matrix (background model). If the eigenvalues of the matrix are not continuously updated, but between one update and another b images pass, the new ratio of images per second obtained is shown in Figure 8.

Figure 8. Ratio of images achieved per second with $b \neq 1$.



- As may be seen from this figure, from $b = 100$ onwards, independent of the number of significant eigenvectors, the system reaches its maximum value at around 250 images per second for $t \leq 3$ and around 190 for $t \geq 4$. This is because the system segmentation is at its most efficient at this number of images. In Table 2 a summary of the final amount of resources consumed by the different blocks implemented on the Xilinx FPGA is presented. It is important to point out that due to the limited resources of the FPGA every attempt has been made to optimise the design at all times, with the aim of reducing the use of internal resources. Thanks to this, from a number of BRAM (block RAM) components and *slices* point of view, it has been possible to implement the entire system on a medium to low range FPGA like the Xilinx XC2VP7.

Table 2. Summary of all the resources consumed by the entire developed system on a XC2VP7.

Area (Slices)	**BRAM**	**Multipliers**	**f_{CLKMAX}**
4225 (86%)	40 (91%)	43 (98%)	112,4MHz

With respect to the frequency of the FPGA clock, according to the reports generated by the implementation tool, a maximum value of 1,124 MHz for the entire FPGA is assured. However, the master frequency chosen for our design is 100 MHz as from this value all the other necessary frequencies can be generated (the camera and external memory frequencies).

As for the real results obtained, Figure 9 shows images captured with the developed platform [8] with an initial resolution of 1,280 × 1,023 reducing their size to 256 × 256 by applying a *binning* process on the FPGA. This sequence was captured in the grounds of the University of Alcala where the distance between the objects to be detected, in this case people and the camera, is 25 meters.

Figure 9. Sequence of images captured to determine new objects.

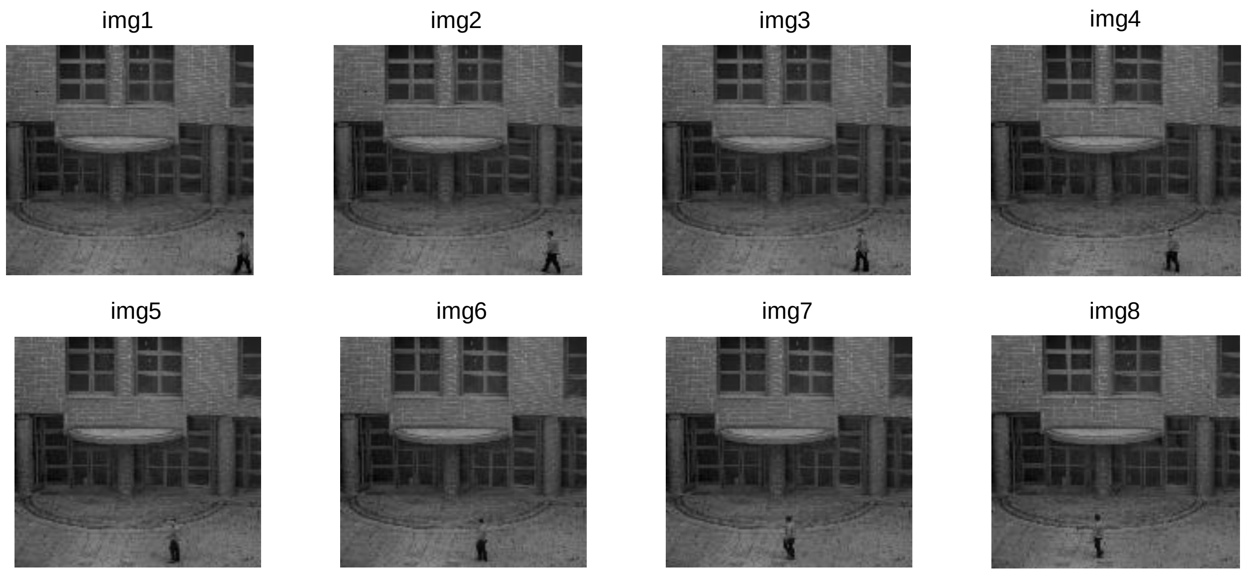
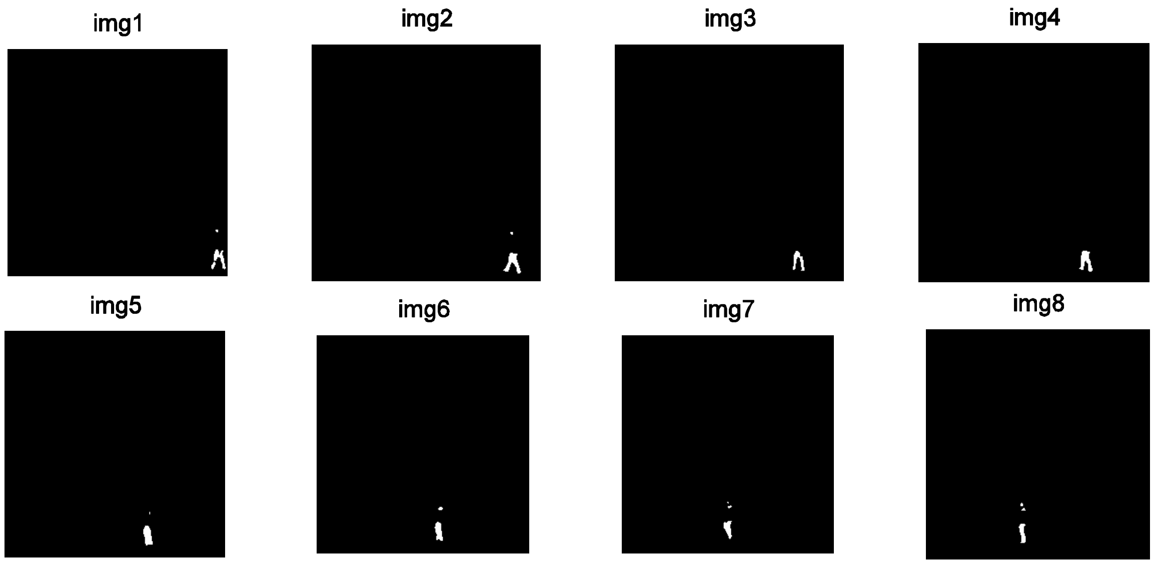


Figure 10 shows the detection that was performed. The proposed design has been tested with a bank of 1,000 images captured under moderate lighting conditions in outside environments. The accuracy achieved in the test was remarkable (around 97% of true matches). Despite the promising results for an embedded architecture, it is widely known that when using PCA for modelling strong illumination changes in the intensity values of the image require a high amount of PCA vectors to train the background. Besides, due to the fact that illumination changes are non-linear variations of the intensity, the PCA subspace cannot model such variations properly, which could increase the number of false detections. In a near future the proposal can be easily applied to other colour spaces, such as the light invariant space proposed in [26], which maps a RGB image to a scalar image where same surfaces under different illuminations are mapped to the same intensity value.

Figure 10. Sequence of images detected to determine a new object from those captured in Figure 9.



6. Conclusions

This work presents a new image capture and processing system implemented on FPGAs for detecting new objects in a scene, starting from a reference model of the scene. To achieve this, the Principle Component Analysis (PCA) technique has been used. The main objective is to parallelize it in order to achieve a concurrent execution which will enable processing speeds of around 120 images per second to be reached. This processing speed, including all stages included in the PCA technique (calculating eigenvalues and eigenvectors, projection and recovery of images to/from the transformed space, obtaining map of distances, *etc.*) responds to the requirements of many applications, where the goal is the detection of new objects in the scene, even in those cases where, for a variety of reasons, (changes in lighting for example) a continuous update of the background model is required. The proposed solution is a significant improvement on other hybrid solutions based on the use of a PC and an FPGA [5]. The complete integrated development of the PCA algorithm on an FPGA was a task that until now had not been achieved or performed, at least according to our thorough review of related work done on this topic. Thanks to the designed solution new applications with PCA algorithm could be implemented for new proposals or applications.

Acknowledgements

This work was made possible thanks to the sponsorship of the Ministry of Education and Science (MEC) and the projects ESPIRA (REF-DPI2009-10143) and SIAUCON (REF-CCG08-UAH/DPI-4139), funded by the University of Alcala and the Madrid Regional Government.

References

1. Ratha, N.K.; Jain, A.K. Computer vision algorithms on reconfigurable logic arrays. *IEEE Trans. Parallel Distrib. Syst.* **1999**, *10*, 29-43.

2. Magdaleno, E.; Rodríguez, M; Rodríguez-Ramos, J.M. An efficient pipeline wavefront phase recovery for the CAFADIS camera for extremely large telescopes. *Sensors* **2010**, *10*, 1-15.
3. Talu, H.M.; Igci, E.; Tekin, M.E.; Sevtekin, H.S.; Genç, B.Ç, Heywood, M.I. Reconfigurable computing implementation of binary morphological operators using 4-, 6- and 8- connectivity. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'00),* Las Vegas, NV, USA, June 2000; pp. 3386-3389.
4. Parra, I.; Fernández, D.; Sotelo, M.A.; Bergasa, L.M.; Revenga, P.; Nuevo, O.M.; García, M.A. Combination of feature extraction methods for SVM pedestrian detection. *IEEE Trans. Intel. Trans. Syst*. **2007**, *8*, 292-307.
5. Fleury, M.; Self, R.P.; Downton, A.C. Development of a fine-grained parallel Karhunen-Loeve transform. *J. Parall. Distrib. Comput.* **2004**, *64*, 520-535.
6. Gottumukkal, R.; Asari, V.K. An improved face recognition technique based on modular PCA approach. *Pattern Recog. Lett.* **2004**, *25*, 429-436.
7. Zhong, F.; Capson, D.W.; Schuurman, D.C. Parallel architecture for PCA image feature detection using FPGA. In *Proceedings of Canadian Conference on Electrical and Computer Engineering (CCECE 2008)*, Dundas, ON, Canada; May 2008; pp. 1341-1344.
8. Bravo, I.; Jiménez, P.; Mazo, M.; Lázaro, J.L.; Martín, E. Architecture based on FPGA's for real-time image processing. *Lect. Notes Comp. Sci Reconfig. Comput.: Architect. Appl.* **2006**, *3985*, 153-157.
9. Dunteman, G.H. Principal Component Analysis. Sage Publications: Newbury Park, CA, USA, 1989.
10. Jolliffe, I.T. Principal Component Analysis, 2nd ed; Springer Series in Statistics, Springer-Verlag: New York, NY, USA, 2002.
11. Swets, D.L.; Weng, J.J.; Using discriminant eigenfeatures for image retrieval. *IEEE Trans. Patt. Anal. Mach. Intell.* **1996**, *18*, 831-836.
12. Vázquez, J.F.; Lázaro, J.L.; Mazo, M.; Luna, C.A. Sensor for object detection in railway environment. *Sensor Lett.* **2008**, *6*, 690-698.
13. Aptina Imaging. *Datasheets: 1.3-Megapixel CMOS Active-Pixel Digital Image Sensor MT9M413C36STM*. Aptina Imaging Corp.: San Jose, CA, USA, 2004.
14. Bravo, I.; Jimenez, P.; Mazo M.; Lazaro, J.L.; de las Heras, J.J.; Gardel, A. Different proposals to matrix multiplication based on FPGAs. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE07),* Vigo, Spain, December 2007; pp. 1709-1714.
15. Ahmedsaid, A.; Amira, A.; Bouridane, A. Accelerating MUSIC method on reconfigurable hardware for source localization. In *Proceedings of the 2004 International Symposium on Circuits and Systems (ISCAS'04),* Tel Aviv, Israel, December 2004; pp. 369-372.
16. Kim, M.; Ichige, K.; Arai, H. Implementation of FPGA based fast DOA estimator using unitary MUSIC algorithm. In *Proceedings of the IEEE 58th Vehicular Technology Conference (VTC 03),* Jeju, Korea, April 2003; pp. 213-217.
17. Wilkinson, J.H. *The Algebraic Eigenvalue Problem*; Oxford University Press: Oxford, UK, 1999.
18. Brent, R.P.; Luk, F.T. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM J. Sci. Stat. Comput.* **1985**, *6*, 69-84.

19. Bravo, I.; Mazo, M.; Lazaro, J.L.; Jimenez, P.; Gardel, A.; Marron, M. Novel HW Architecture Based on FPGAs Oriented to Solve the Eigen Problem. *IEEE Trans. Very Large Scale Integration (VLSI) Syst.* **2008**, *16*, 1722-1725.
20. Todd, S. Algorithm and hardware for a merge sort using multiple processors. *IBM J. Res. Dev.* **1978**, *22*, 509-517.
21. Walther, J.S. A unified algorithm for elementary functions. In *Proceedings of AFIPS Spring Joint Computer Conference*, Atlantic City, NJ, USA, May 1971; pp. 379-385.
22. Karabernou, S.M.; Terranti, F. Real-time FPGA implementation of Hough Transform using gradient and CORDIC algorithm. *Image Vis. Comput.* **2005**, *11*, 1009-1017.
23. Bravo, I.; Hernández, A.; Gardel, A.; Mateos, R.; Lázaro, J.L.; Díaz, V. Different proposals to the multiplication of 3 × 3 vision mask in VHDL for FPGAs. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Lisbon, Portugal, September 2003; pp. 208-211.
24. Nakagawa, Y.; Rosenfeld, A. Some experiments on variable thresholding. *Pattern Recognit.* **1979**, *11*, 191-204.
25. Manay, S; Yezzi, A. Anti-geometric diffusion for thresholding and fast segmentation. *IEEE Trans. Image Process.* **2003**, *12*, 1310-1323.
26. Finlayson, G.D.; Hordley, S.D.; Lu, C.; Drew, M.S. On the removal of shadows from images. *IEEE Trans. Patt. Anal. Mach. Int.* **2006**, *28*, 59-68