J Real-Time Image Proc (2015) 10:43–57
DOI 10.1007/s11554-012-0249-6

ORIGINAL RESEARCH PAPER

Codebook hardware implementation on FPGA for background subtraction

Rafael Rodriguez-Gomez · Enrique J. Fernandez-Sanchez · Javier Diaz · Eduardo Ros

Received: 18 October 2011 / Accepted: 19 March 2012 / Published online: 24 April 2012

Abstract Object detection and tracking are main tasks in video surveillance systems. Extracting the background is an intensive task with high computational cost. This work proposes a hardware computing engine to perform background subtraction on low-cost field programmable gate arrays (FPGAs), focused on resource-limited environments. Our approach is based on the codebook algorithm and offers very low accuracy degradation. We have analyzed resource consumption and performance trade-offs in Spartan-3 FPGAs by Xilinx. In addition, an accuracy evaluation with standard benchmark sequences has been performed, obtaining better results than previous hardware approaches. The implementation is able to segment objects in sequences with resolution 768×576 at 50 fps using a robust and accurate approach, and an estimated power consumption of 5.13 W.

Keywords Field programmable gate arrays · Fixed-point arithmetic · Real-time image processing · Video surveillance

R. Rodriguez-Gomez · E. J. Fernandez-Sanchez (✉) · J. Diaz · E. Ros
Department of Computer Architecture and Technology, University of Granada, Granada, Spain
e-mail: efernandez@atc.ugr.es

R. Rodriguez-Gomez
e-mail: rrodriguez@atc.ugr.es

J. Diaz
e-mail: jdiaz@atc.ugr.es

E. Ros
e-mail: eduardo@atc.ugr.es

1 Introduction

There has been an increasing interest in the application of computer vision methods to video surveillance issues. The main interest of video analytics systems is the detection of people who appear in a scene and behaviors that can lead to alarm situations. For that reason, background subtraction (BGS) is usually considered a necessary initial phase in this kind of system [6]. This technique consists of analyzing a video stream to detect which regions of each frame belong to objects in movement and which ones are parts of the background. BGS is an intensive pixel-wise processing stage. Its efficient implementation can lead to a dramatic reduction in the system computing power requirements. This can allow stand-alone video-analytic platforms on remote places where low power consumption may be a critical requirement. Higher-level stages can be significantly simplified if this BGS is done accurately.

1.1 Related work

The current state of the art of background segmentation algorithm is able to deal not only with static backgrounds, but also with moving ones (such as waving trees, escalators, etc.). In order to model these complex scenarios, many models have been developed. The generalized mixture of Gaussians, MOG, presented in [27], has been used to model non-static backgrounds with multiple values per pixel. The more advanced methods have adapted MOG features to different specific scene characteristics and have incorporated methods that use Bayesian frameworks [21], and edge and region-based information [14, 33].

However, the large amount of operations of these intensive pixel-wise models often prevent from reaching real-time requirements on small processors (suitable for

embedded applications) or significantly reduces the scalability of the system on commodity processors if the number of cameras grow beyond few units. In this contribution, we focus on a distributed architecture based on embedded systems and low-power devices such as field programmable gate arrays (FPGAs). Much work can be found in the literature about FPGA implementations for vision applications, such as object recognition [4], object tracking [7] or image segmentation [29].

On the other hand, hardware acceleration with dedicated computing architectures (for instance, ASIC or FPGA devices) may lead to a considerable loss of accuracy due to necessary simplifications to achieve real-time requirements at affordable computing resources. The complexity that characterizes those models makes them difficult to implement in environments with limited resources. For that reason, several less complex approaches can be found in the literature, as FPGA-oriented simplifications of MOG [1, 15], a spatial BGS technique [17], a DSP-embedded implementation [11], a memory reduction scheme [16] or static algorithms where it is assumed that the background is fixed as the one proposed by Karaman et al. [18] and Horprasert et al. [9], which has been implemented on FPGA in [25].

In this work, we describe a novel FPGA-based video BGS architecture based on the Codebook algorithm proposed by Kim et al. [19]. This algorithm has been classified by many authors [12, 30] as a good trade-off between accuracy and efficiency. This has motivated our choice as a target model for implementation even though it is a much more complex model than the previous hardware implementations available in the literature. With this architecture, we can accomplish real-time requirements with a very small loss of accuracy compared to the software implementation and it is very well suited for integration into smart cameras or for utilization as system coprocessor in power-aware multi-camera systems.

Therefore, the paper presents two important contributions to the state of the art. Firstly, to our knowledge, it is the first time that the foreground–background segmentation using a codebook-based model is implemented on FPGA, and specific datapaths have been developed to perform this task in real time on a resource-constrained FPGA device. Secondly, a proper combination of co-design strategies has been used, as well as high abstraction level and RT level datapaths. This allows us to achieve a very high-performance system finely tuned to the proposed algorithm, enabling small resources consumption. The novel methodology is seldom applied in the literature and is the reason that our system overcomes other approaches regarding accuracy results.

The paper is organized as follows. In Sect. 2, we describe the original codebook model, specifying the construction, detection and update phases. In Sect. 3, we indicate and justify the simplifications made to adapt the model to limited resources environments. We show, in Sect. 4, the customized architecture for algorithm implementation and how each stage has been developed. In Sect. 5, results are shown and analyzed taking into account system resources, performance and accuracy of the segmentation evaluated using a standard benchmark. Finally, conclusions and discussion are presented in Sect. 6.

2 Codebook background estimation model review

The Codebook algorithm, as proposed by Kim et al. [19], is based on the construction of a background model adopting a quantization/clustering technique described by Kohonen [20] and Ripley [24]. The above-mentioned work shows that the background model for each pixel is given by a codebook consisting of one or more codewords. Codewords are data structures that contain information about pixel colors, color variances and information about how frequently each codeword is updated or accessed.

The different stages of the Codebook algorithm are described below:

2.1 Construction of the initial codebook

Given a set of N time steps (frames), a training sequence S is used for each pixel consisting of N RGB vectors. Each pixel has a different codebook, represented as $C = \{c_1, c_2, c_3, \ldots, c_L\}$, consisting of L codewords, where L can be different for each pixel. Each codeword $c_i, i = 1\ldots, L$; consists of an RGB vector $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ and a 6-tuple $aux_i = \langle \check{I}_i, \hat{I}_i, f_i, \lambda_i, p_i, q_i \rangle$, described as follows:

- $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$, average value of each color component.
- $\check{I}_i, \hat{I}_i$, minimum and maximum brightness, respectively, of all pixels assigned to codeword c_i.
- f_i, the frequency (number of frames) with which codeword c_i has been updated.
- λ_i, the *maximum negative run-length* (MNRL), defined as the longest interval of time during which the codeword c_i has not been updated.
- p, q, the first and last updating access times of codeword c_i.

The detailed algorithm for codebook construction (CBC) is given in Fig. 1.

Conditions (a) and (b) in step 2(ii) must be evaluated to determine if a pixel $x_t = (R, G, B)$ matches the codeword c_m. These two conditions and λ parameter are explained in

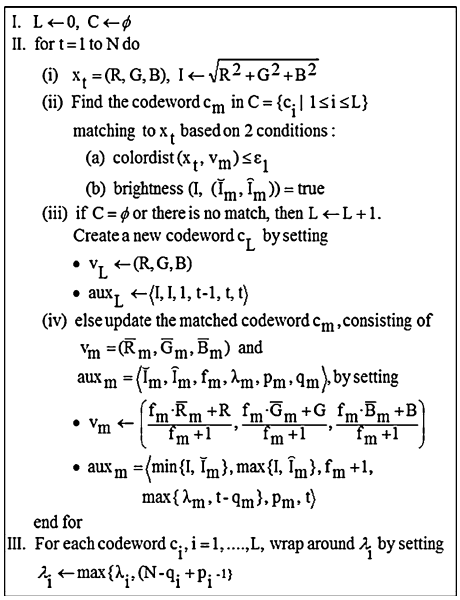

Fig. 1 Algorithm for codebook construction

detail in [19]. Summarizing, we can say that the evaluation of color distortion basically consists of determining the distance between the color of an input pixel $x_t = (R, G, B)$ and $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ of codeword c_i, as indicated in (1).

$$\begin{aligned} \|x_t\|^2 &= R^2 + G^2 + B^2 \\ \|v_i\|^2 &= \bar{R}_i^2 + \bar{G}_i^2 + \bar{B}_i^2 \\ \langle x_t, v_i \rangle^2 &= (\bar{R}_i R + \bar{G}_i G + \bar{B}_i B)^2 \end{aligned} \tag{1}$$

Color distortion δ is calculated as indicated in (2).

$$\begin{aligned} p^2 &= \|x_t\|^2 \cos^2\theta = \frac{\langle x_t, v_i \rangle^2}{\|v_i\|^2} \\ colordist\,(x_t, v_i) &= \delta = \sqrt{\|x_t\|^2 - p^2} \end{aligned} \tag{2}$$

Now, condition (b) evaluates how brightness change of $x_t = (R, G, B)$ lies within $[I_{\text{low}}, I_{\text{hi}}]$ range for codeword c_i. In this way, we allow the brightness change to vary in a certain range, as defined in (3).

$$\begin{aligned} I_{\text{low}} &= \alpha \hat{I} \\ I_{\text{hi}} &= \min\left\{ \beta \hat{I}, \frac{\check{I}}{\alpha} \right\} \end{aligned} \tag{3}$$

Typically, α is in the interval [0.4, 0.8], and β is in the interval [1.1, 1.5]. The brightness function is defined in (4).

$$brightness(I, \langle \check{I}, \hat{I} \rangle) = \begin{cases} \text{true} & \text{if } I_{\text{low}} \le \|x_t\| \le I_{\text{hi}} \\ \text{false} & \text{otherwise} \end{cases} \tag{4}$$

The set of codebooks obtained from the previous step (*codebooks construction*) may include some moving foreground objects as well as noise. To obtain the true background model, it is necessary to separate the codewords containing foreground objects from the true background codewords. This true background includes both static pixels and background pixels with quasi-periodic movements (for instance, waving trees in outdoor scenarios). The background model M obtained from the initial set of codebooks C is given in (5).

$$M = \{c_m | c_m \in C \wedge \lambda_M \le T_M\} \tag{5}$$

λ_M (MNRL) is defined as the maximum interval of time that the codeword has not been updated during the training period. T_M is the time threshold set equal to half the number of training frames, $N/2$. Thus, codewords having a large λ_M (larger than T_M) will be eliminated from the corresponding codebook.

2.2 Foreground detection

Subtracting the foreground from the current image is straightforward once we have obtained the background model M. The algorithm performing this task is detailed in Fig. 2.

2.3 Background modeling updating

The original model assumes that the background obtained during the initial background modeling is permanent. To improve the model, making it more useful in a surveillance system, Kim et al. [19] have proposed a layered modeling and detection scheme. The initial scene can change after initial training. Therefore, these changes should be used to update the background model M. This can be done by defining an additional model H, called *cache*, where the new codewords are stored. Three new parameters $(T_H, T_{\text{add}}, T_{\text{delete}})$ are also defined. The periodicity of a codeword h_i stored in cache H is filtered by T_H, as we did previously with T_M in the background model M. The codewords h_i remaining in cache H for a time interval larger than T_{add} are added to the background model M. Codewords of M not accessed for a period of time (T_{delete}) will be deleted from the background model.

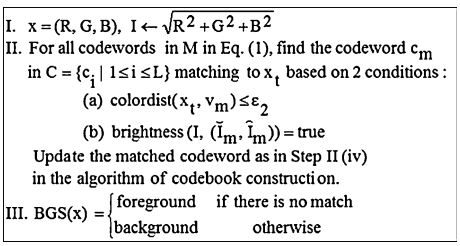

Fig. 2 Algorithm for background subtraction stage

In our case, we have considered that the background model can be subject to total changes and, therefore, the initial codewords can be replaced as a whole by new codewords from cache H. In addition, to adapt the algorithm to the hardware implementation, allowing an easier use of the external memory, we have only layers of background (M and H). The details of the algorithm for layered modeling and updating are given in Fig. 3.

3 Model simplifications towards a hardware-friendly approach

The original model by Kim et al. [19] needs to be simplified to arrive at an affordable high-performance hardware system. The main stages suitable for simplification issues can be summarized as follows:

- Storage and memory management. The amount of memory required to store the codebooks may change because the total number of codewords may increase or decrease dynamically. As described in Sect. 4, our system uses a DDR2 memory, which provides high bandwidth, but it needs a regular access to reach the maximum performance, which complicates the dynamic management of the set of codebooks.
- Color distortion and brightness distortion computation. Equation (2) requires square root and division operations, which are expensive on resources-constrained hardware devices.
- Model accuracy degradation due to fixed point arithmetic. Customized hardware systems normally use fixed-point data representation to reduce hardware resources utilization. However, this strategy requires careful analysis to avoid any degradation in accuracy.

As described below, we have carried out a detailed analysis to determine the modifications required to address the conversion to fixed point arithmetic.

I. After training, the background model M is obtained.
Create a new model H as a cache.
II. For an incoming pixel x, find a matching codeword in M.
If found, update the codeword.
III. Otherwise, try to find a matching codeword in H and update it.
For no match, create a new codeword h and add it to H.
IV. Filter out the cache codewords based on T_H:
$H \leftarrow H - \{h_i \mid h_i \in H, \lambda \text{ of } h_i \text{ is longer than } T_H\}$
V. Move the cache codewords staying for enough time, to M:
$M \leftarrow M \cup \{h_i \mid h_i \in H, h_i \text{ stay longer than } T_{add}\}$
VI. Delete the codewords not accessed for a long time from M:
$M \leftarrow M - \{c_i \mid c_i \in M, c_i \text{ not accessed for } T_{delete}\}$
VII. Repeat the process from the Step II.

Fig. 3 Layered modeling and updating scheme

3.1 Maximum number of codewords per pixel

To adapt the algorithm to the hardware implementation, allowing an easier use of the external memory containing the codewords, we have limited the number of codewords in each pixel. To establish the maximum number of codewords, we have carried out a detailed study in which we look at the optimal number of codewords, without significantly compromising the accuracy of the model and with a moderate consumption of hardware resources. To determine the number of necessary codewords, we have done a comprehensive testing ground using the Wallflower test database [28], which tests different algorithms in several problematic situations from which the ground truth (foreground) is known. Since the target is the segmentation of two elements, foreground and background, the comparison metric is obtained from the total sum of false positives (FP) and false negatives (FN) in a particular frame in each of the Wallflower [28] test sequences. The original version of the algorithm, which does not have a maximum number of codewords, is compared with another version in which we will modify the maximum number of codewords allowed. The total number of errors (FP + FN) will be equal to the number of different pixels existing between the binary mask image of the original version and the binary mask image of the version containing a limited number of codewords. Figure 4 shows the percentage of errors from the total number of pixels of the evaluation frame in each of the Wallflower test sequences, depending on the maximum number of codewords. If we limit the number of codewords to two or three in sequences with non-static backgrounds (waving trees), our architecture performs similarly to unimodal models [9]. After an in-depth analysis of accuracy and resources consumption, our choice of optimal number of codewords for a wide range of possible scenes is set to 5.

As stated before, the number of codewords assigned to a pixel will be limited to facilitate hardware implementation, without significantly compromising the accuracy of the system. With this simplification, it may happen that the memory space of certain codebooks is already full (maximum limit reached) when new codewords are created. Then, it is necessary to replace an existing codeword. If this is the case, the replaced codeword will be the one not having been accessed for the longest time period. This condition will be added in step 2(iii) (Fig. 1), when new codewords are created.

3.2 Color distance and brightness computation

As we will detail in the description of the architecture, all the codewords c_m are compared with the incoming pixel

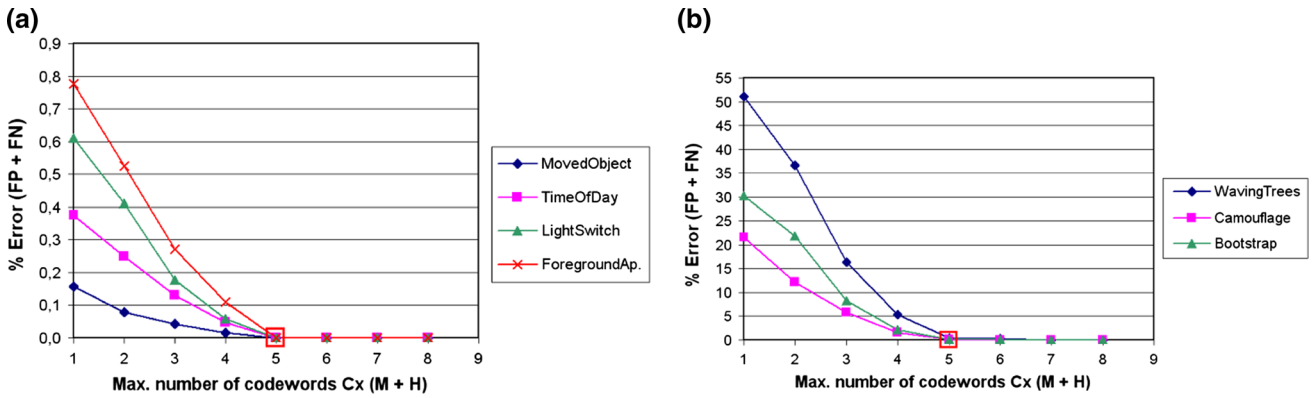

Fig. 4 Percentage of errors rate (total sum of false positives FP and false negatives FN) for a frame taken from the Wallflower test sequences, depending on the maximum number of codewords allowed. For the sake of clarity, **a** and **b** are separated due to the different error ranges. Our choice (set to 5) has been marked with a *big open square*

$x_t = (R, G, B)$ in parallel, using a $colordist(x_t, v_i)$ and $brightness(I, \langle \check{I}, \hat{I} \rangle)$ block for each codeword c_m. Both division and square root operations implemented in these blocks represent a considerable consumption of hardware resources. Therefore, it is desirable to avoid these calculations in the FPGA implementation without significantly affecting the accuracy of the algorithm, using in some cases multipliers which are optimized with the embedded resources of the FPGA DSP48 for a Xilinx Spartan-3A DSP [31].

With respect to color distortion δ, we have implemented the modifications indicated in (6).

$$\begin{aligned} &\|v_i\|^2 p^2 = \langle x_t, v_i \rangle^2 \\ &\|v_i\|^2 \delta^2 = \|v_i\|^2 \|x_t\|^2 - \|v_i\|^2 p^2 = \|v_i\|^2 \|x_t\|^2 - \langle x_t, v_i \rangle^2 \end{aligned} \tag{6}$$

Condition (a) in Fig. 2 will remain as (7).

$$\|v_i\|^2 \|x_t\|^2 - \langle x_t, v_i \rangle^2 \leq \epsilon^2 \|v_i\|^2 \tag{7}$$

With respect to brightness, we have established values $\alpha = 0.5$ and $\beta = 1.25$, so that calculations in (3) can be easily computed by means of bit shifts, as follows in (8).

$$\begin{aligned} &I_{\text{low}} = \alpha \hat{I} = \hat{I} \gg 1 \\ &I_{\text{hi}} = \min\left\{ \beta \hat{I}, \frac{\check{I}}{\alpha} \right\} = \min\{\hat{I} + (\hat{I} \gg 2), \check{I} \ll 1\} \end{aligned} \tag{8}$$

In this way, we have reduced the consumption of hardware resources by avoiding the use of two multipliers and one division.

During the updating process of v_m, there is a division for each color component. To reduce the consumption of resources in the FPGA, we have approximated f, which is the denominator of these fractions, to its nearest power of 2. This approximation allows for the use of shift operations instead of divisions. The implementation details of this modification can be seen in Fig. 5.

3.3 Fixed point arithmetic: bit-width optimization

The software implementation has been developed using double floating point representation, which enables a high accuracy (at the expense of using high cost computing units with high resources consumption). For FPGA hardware implementation with constrained resources, a fixed-point data representation is usually adopted, as it is more suitable for the type of resources in FPGA devices. In addition, specific purpose architectures cannot afford floating point arithmetic when implementing long-datapath pipelined computing architectures that may have a large number of processing elements. In this case, a detailed study is required to optimize the trade-off between accuracy and hardware resources utilization. We must bear in mind that using an insufficient number of bits will produce inaccurate results with a high level of quantization noise. On the other hand, using too many bits leads to a considerable increase

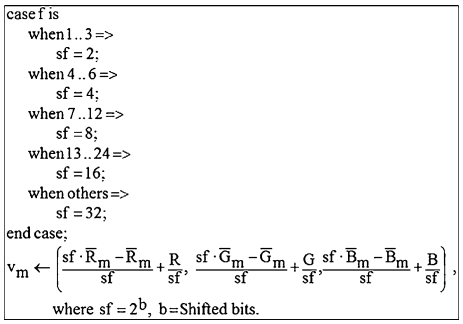

Fig. 5 Updating process of v_m. f is approximated to its nearest power of 2 to reduce hardware resources consumption

in the consumption of hardware resources, making the system implementation in the FPGA unfeasible.

To establish the appropriate number of bits of the fractional part of $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ and $(\check{I}, \hat{I})$ variables participating in the calculation of colordist and brightness, we have used a simulator of our architecture with different bit-width configurations and we have compared them with the results obtained from the version using double floating point data representation. This simulator is provided by ImpulseC [13] to check the system degradation due to fixed-point operations. To carry out this comparison, we also use the Wallflower test database. Again, the comparison metric measures the total sum of FP and FN. Figure 6b shows that with regards to $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ variables used in the calculation of colordist. The best representation is 5 bits for the fractional part and 8 bits for the integer part. According to Fig. 6a with regards to the $(\check{I}, \hat{I})$ variables for brightness evaluation, it is appropriate to use 3 bits for the fractional part and 9 bits for the integer part. Furthermore, an in-depth analysis shows how the increase in the $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ variables bit-width leads to an increase in the consumption of hardware resources.

With these choices in Table 1 we see the main algorithm data structures (system registers) and the associated bit-width choices, fixed after the study explained in this section. Along the pipeline datapath, each data structure has a different bit-width which is optimized to the type of performed operations (multiplications, additions, subtractions, etc.). This approach allows researchers to tune resources and accuracy of the system in a much finer way.

The choice of this bit-width is also due to the possibility of maintaining a compact organization of the external memory, as the codeword data size is 4 words of 32 bits.

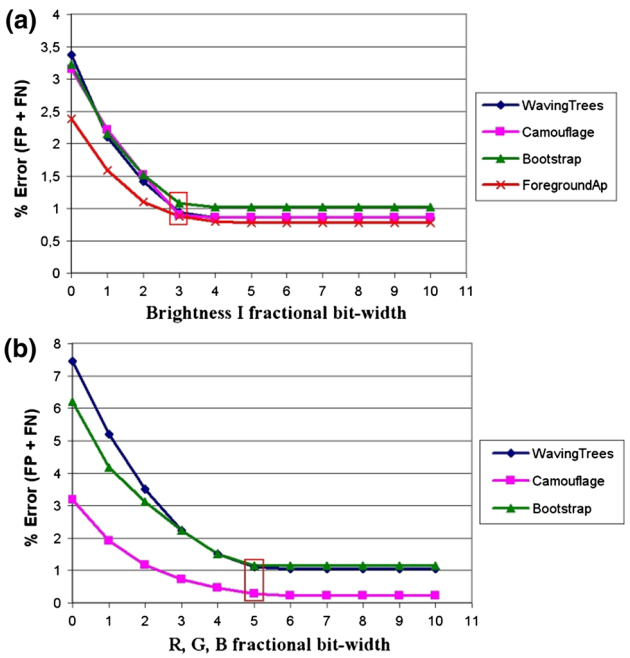

Fig. 6 Percentage of errors considering different bit-width of the fractional part of $(\check{I}, \hat{I})$ and $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$, **a** and **b**, respectively. In the figure, we use only the Wallflower test sequences with a significantly representative error when the fractional part changes (WavingTrees, Camouflage, Bootstrap and Foreground Aperture)

Table 1 Bit-width of each variable taking part in the calculation of colordist and brightness

	Variable		Bits
	R_i, G_i, B_i		[8 0]
	$\bar{R}_i, \bar{G}_i, \bar{B}_i$		[8 5]
$\check{I}, \hat{I}$	$\|x_t\|$		[9 3]
$\langle x_t, v_i \rangle^2$	$\|v_i\|^2 \|x_t\|^2 - \langle x_t, v_i \rangle^2$	$\|v_i\|^2 \|x_t\|^2$	[36 5]
$\|v_i\|^2$	$\|x_t\|^2$		[18 5]
	$\epsilon^2 \|v_i\|^2$		[26 5]

The first value represents the integer part and the second value, the fractional part

Once we have established the bit-width of the fractional part of $v_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ and $(\check{I}, \hat{I})$, we have also evaluated the degradation of our design combining all the modifications required to obtain a hardware-friendly model. These results are shown in the final evaluation (Sect. 5.3).

4 Hardware architecture

The proposed architecture has been developed using EDK (Embedded Development Kit) and ISE Foundation of Xilinx Inc. [31]. The EDK environment facilitates the design of complex and completely modular system-on-chip architectures able to support embedded microprocessors (MicroBlaze, PowerPC), peripheral and memory controllers, and interconnecting buses, whilst IP cores for specific processing can be designed using hardware description language (HDL) in the ISE tool. For a better understanding of the designed architecture using EDK, it is important to highlight the use of a ViSmart video processing board from Seven Solutions S.L. [26], including: two Xilinx XC3SD3400aFG676 FPGAs, two 256 MB DDR2 DIMM memory modules, four independent analog video inputs, two gigabit ethernet connections, 485 connection, a 3G connection module, a 64 MB Flash memory, and two 1 MB $\times$ 36 bits ZBT (Zero-Bus Turnaround) memories. In our case, we have only used one of the FPGAs included in the ViSmart board.

This architecture consists of several modules and interconnecting buses, as shown in Fig. 7. Processing modules, peripherals and a Microblaze processor are

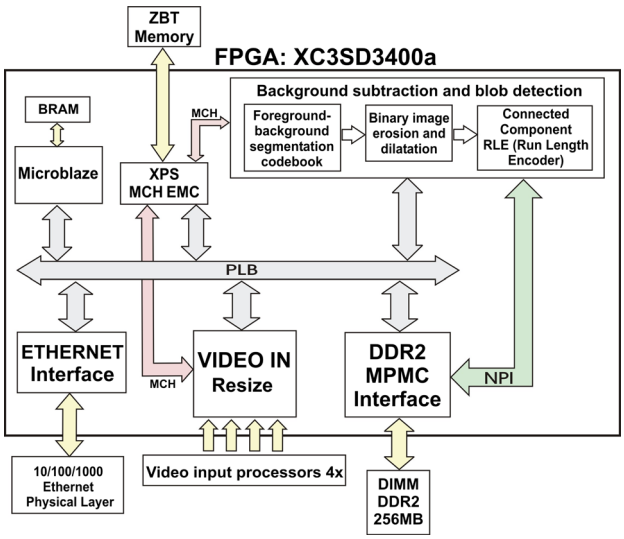

Fig. 7 Scheme of the complete architecture, including connections between the "background subtraction and blob detection" modules, peripherals, memory and processor

connected to the system bus (PLB, Processor Local Bus). Through the PLB bus, Microblaze has access to memory regions, configuration registers of the peripherals and the ethernet interface for data and image sending/receiving. In addition to the construction of codebooks, the BGS and blob (group of connected foreground pixels) detection module, seen in Sect. 4.1, performs an intensive processing on the pixels of each image to separate foreground and background, and then proceeds to the blob extraction of the different objects. The MPMC module, DDR2 memory controller presented in [32], provides an easy access to the external DDR2 memory, which stores all the codebooks and offers efficient high bandwidth access, thus providing a feasible use for applications requiring real-time processing.

In the following subsections, all details regarding the "background subtraction and blob detection" IP core are explained. These details include memory management and access to input images and model information, as well as every computation related to the different stages of the Codebook algorithm [19].

4.1 Memory management

The hardware description language that we have used to implement this IP core is ImpulseC [13], which allows us to work at a high level of abstraction, enabling the construction of a multi-stage pipelined architecture running in parallel. The parallel execution of these stages is the key point for the high performance obtained in our system.

Figure 8 shows the proposed architecture for this IP core. Before getting into the details of this architecture, it is important to remark that memory has a key role in the performance of the system and requires an efficient memory accessing scheme. The codebook model requires an intensive utilization of memory resources and poor system memory architectures drastically reduce the system performance. This has motivated the utilization of high-performance multi-port memory controllers (Xilinx MPMC for DDR2 and XPS_EMC_MCH for SSRAM) as well as very specific and optimized memory ports such as NPI (Native Port Interface) for DDR2 and MCH (Xilinx Multi-Channel) for SSRAM.

To integrate this IP core into the EDK environment, we have developed low level and optimized VHDL interfaces

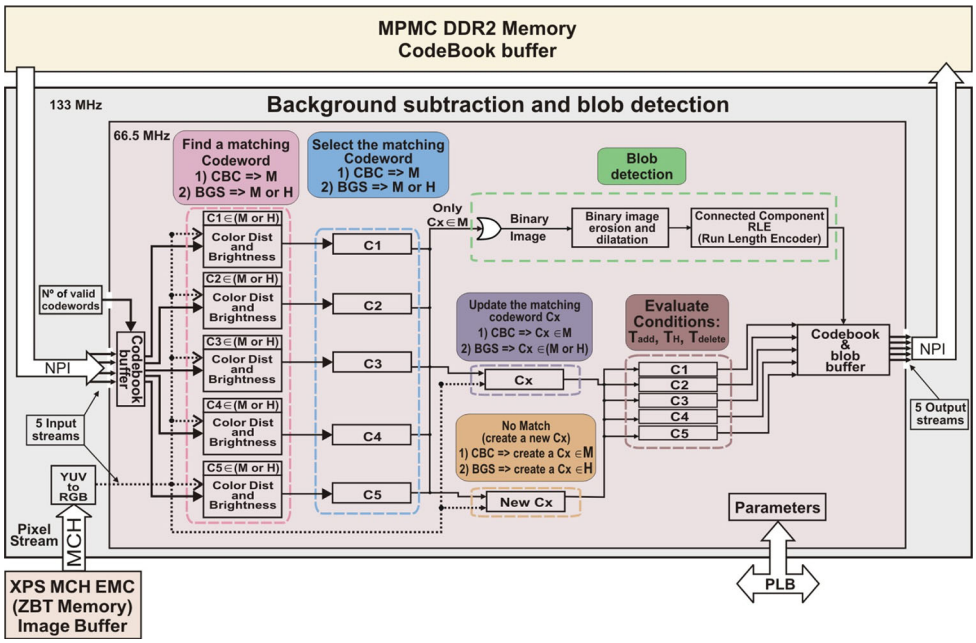

Fig. 8 Simplified datapath architecture for background subtraction and blob detection core with five scalar units (*C1*, *C2*, *C3*, *C4*, *C5*). The architecture works in two different modes: CBC (codebook construction) and BGS (background subtraction)

between the NPI and MCH ports on the one hand, and the IP core input and output streams, on the other hand (ImpulseC description was not adequate for these modules). This interface allows us to read and write efficiently on the codebooks and image memory map, as well as creating separate blocks running at different clock frequencies. The input image pixels, which are stored in the ZBT memory in YUV format, are obtained through the XPS EMC memory controller MCH port. These pixels are converted into RGB format before introducing them into the IP core input pixel stream.

On the other hand, to obtain the codewords for each pixel which are stored in the DDR2 memory, we will use the DDR2 MPMC memory controller NPI ports. For an efficient use of the NPI port, which makes use of all the available bandwidth, we have configured its interface to allow for reading and writing data bursts at the same frequency (133 MHz) as the DDR2 memory. In addition, the NPI port bit-width has been set at 64 bits and the burst size at 64 words. With this NPI port configuration, Xilinx states [32] that 923 MB/s can be obtained as maximum total data throughput. We check this data later on, in the performance evaluation section.

Sequential and burst access to the DDR2 memory provides high bandwidth access. However, there is the disadvantage of inefficient access to random positions. In Sect. 3, we explain that a key simplification for our hardware architecture is the assignment of a maximum number of codewords per pixel, thus facilitating the efficient use of memory and caching techniques. As explained in Sect. 3 we limit the number of codewords to 5, distributed all along the background model M and cache H. Figures 8 and 9 show the datapath and memory map configuration where the list of codebooks is stored. These figures illustrate the processing elements of the architecture as well as memory interfaces required, showing multiple parallel modules to point up the superscalar units used in the architecture. The different elements represent the computation modules described in Sects. 2 and 3. In Fig. 8 ,we can see that an image pixel is read from ZBT memory using the MCH–EMC interface and at the same time the associated codeword is read from DDR2 using the NPI interface. Each pixel has a minimum of one codeword (C1) and a maximum of five codewords (C1, C2, C3, C4, C5), which may belong to M or H. According to this, up to five comparison units run in parallel using the *colordist* and *brightness* to determine if the current pixel matches any of the stored codewords (this is represented by the pink box on the left side of the figure). Purple, orange and brown boxes represent the computing units that update codewords, after which this information is stored in the DDR2 memory using the NPI interface. Finally, the green module is the one that generates the binary foreground image. These data are also stored by the NPI interface. Finally, information about the different clock domain required for the architecture is also included in this figure.

The amount of memory required to store the codebooks may change, as the total number of codewords may increase or decrease dynamically; in order to facilitate the use of this memory, we have to make sure that all the codewords are lined up consecutively in memory, depending on the pixel they belong to, always bearing in mind that both data reading and writing are performed in bursts and in consecutive addresses. To maintain this order without moving large amounts of data nor compromising the use of the DDR2, we have used two different memory regions to store the codebooks. During the processing of frame n, region 1 will be the read buffer and region 2 will be the write buffer; at the same time, the codewords meeting condition in Sect. 2.3 will be eliminated and new codewords will be created and stored in region 2; they will modify the size of the memory in use. For the next frame $n+1$, the two memory regions in use will swap, repeating the same process: region 2 will be the read buffer and region 1 will be the write buffer.

As it will be detailed in Sect. 5.1, the maximum performance frequency for the IP core has been set at 66.5 MHz. In order to introduce codewords data into the IP core at the same speed as they are read by the DDR2 (133 MHz) without data input becoming a bottleneck in the IP core input, we have used four input streams in parallel, with a maximum of 32 bits of width for each of them. As Fig. 9 shows, each codeword has a size of 4×32 bits words; therefore, thanks to the four input streams configuration, it is possible to introduce a codeword on each clock cycle (66.5 MHz) with a 1,064 MB/s IP core input bandwith. All the codewords will enter our IP core

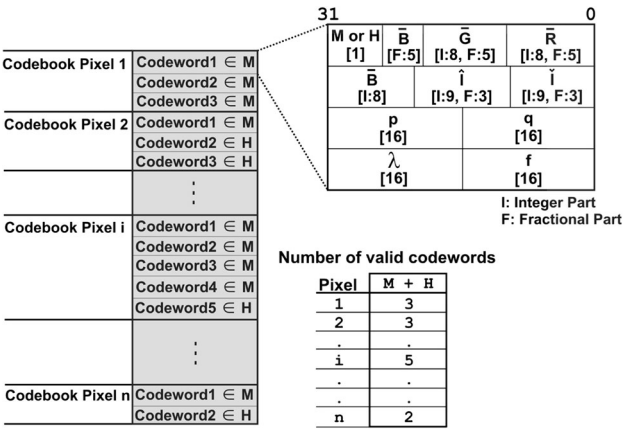

Fig. 9 Memory map configuration. The set of codebooks is stored in the external memory, and its size may change dynamically depending on the total numbers of codewords. We can also see the variables for each codeword and the way in which the fixed point representation enables an efficient use of memory

following the same order as they are read from the DDR2 memory. In addition to this, the number of valid codewords for each codebook will be used as a control parameter for this data input. There is also a fifth input stream with 24 bits of width, containing the pixels of the current image, already converted from YUV format to RGB. The final processing output also consists of four parallel streams providing an updated codeword on each clock cycle. Finally, there is an additional output stream containing the blobs present in the binary image after background/foreground segmentation.

This architecture is based on a multi-stage pipelined structure running in parallel (Fig. 8). In addition, it is controlled by an embedded Microblaze processor, with two different operating modes: (1) CBC and (2) BGS. Microblaze is in charge of controlling and commuting between both modes through the PLB interface. During the construction of the initial codebook, CBC mode, only the following stages will remain active: match, selection, updating and creation of new codewords. Once the training frame is reached, Microblaze will perform step III of the algorithm for CBC and then will filter the background model M according to (5). The last two operations are done only once, after the construction of the initial codebooks. Since they are not expensive in terms of computation resources, Microblaze is able to perform them easily, with no increase in the hardware resources consumption. Finally, once these tasks are performed, Microblaze will commute to BGS operating mode.

4.2 Matching codeword selection

The first processing step (Fig. 10) consists of the parallel evaluation of *colordist* and *brightness* for each codeword and input pixel. Each codeword has its own datapath running in parallel with the rest, with five scalar units (C1, C2, C3, C4, C5). The output of each unit consists of a bit which indicates if codeword x with pixel i meets conditions *colordist* and *brightness*. The number of evaluated codewords changes in each pixel, based on the variations in the values of the RGB components. A scene with little change will have many pixels with only one codeword assigned to them. In this case, our architecture will use just one evaluation scalar unit. On the other hand, scenes with a high degree of changes will have pixels experimenting variations and, therefore, all its available memory may be used, thus using all the evaluation scalar units. Figure 10 shows the fine grain pipelined datapath for evaluating a codeword. Value $\|x_t\|^2$ is intrinsic to every pixel and, therefore, it is common to obtain it for each evaluation scalar unit. The multipliers used to obtain vectors $\|v_i\|^2, \|x_t\|^2, \langle x_t, v_i\rangle^2$ are optimized with the embedded resources (DSP48) of the FPGA Spartan3A DSP. To calculate the square root of $\|x_t\|^2$, representing the brightness of the current pixel, we have used a Xilinx IP core generated by the Core Generator tool and based on the CORDIC algorithm [31] (parallel architectural configuration).

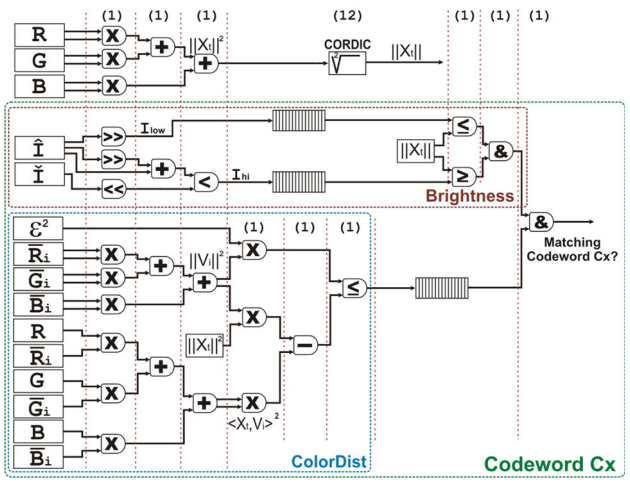

Fig. 10 Fine grain pipelined datapath for evaluating a codeword Cx. This stage corresponds to the implementation of *colordist* and *brightness*. All these operations are computed using 18 pipelined stages. The number of clock cycles is indicated in *brackets*

We use a fixed-point representation for each one of the scalar units; after assessing accuracy versus consumption of resources (Sect. 3), we decided the following representation for input components: (R, G, B) 8 bits integer part and 0 bits fractional part; $(\bar{R}_i R + \bar{G}_i G + \bar{B}_i B)$ 8 bits integer part and 5 bits fractional part, and for $(\check{I}, \hat{I})$, 9 bits integer part and 3 bits fractional part.

The total number of stages (latency) for each scalar unit is 18, with a rate (cycles/result) of 1. It must be highlighted that the parallel CORDIC core has a total number of 12 pipelined stages, and is able to produce a new output data in each clock cycle.

It might be the case that an input pixel meets the two conditions (*colordist* and *brightness*) with different codewords. In order to avoid competition between codewords, a selection stage has been implemented to chose the most likely codeword (largest f), which will be updated in the next stage with the new R, G, B values.

4.3 Update of the matching codeword

The previously selected codeword (Cx) is updated in this stage following step II (stage iv of Fig. 1) of the algorithm for CBC. In the case of v_m, the update will be done by using shift operations instead of divisions, allowing for a low consumption of hardware resources. All variables are updated in parallel, requiring four pipelined stages with a rate of 1 cycle/result. During the construction of codebooks

(CBC), only the codewords in the background model M will be updated. Once BGS starts, codewords within the background model M and cache H are updated.

4.4 Creation of new codewords

If no codeword meets conditions *colordist* and *brightness*, a new codeword will be created by repeating the process from step II(iii) (Fig. 1) of the algorithm for CBC. Since we have limited the number of codewords, in the case of reaching the maximum number of codewords permitted, it will be necessary to replace the codeword Cx not accessed for the longest period of time. This new codeword will be created in the background model M during the CBC or, otherwise, in cache H during BGS.

4.5 Update layered model

The update of the background model M with new codewords, as well as the evaluation of conditions associated with the parameters $(T_H, T_{\text{add}}, T_{\text{delete}})$ are described in Sect. 2.3. The evaluation of these conditions is performed in parallel for each codeword. Parameter T_H is used for evaluation only with cache H codewords, whereas parameter T_{delete} is used for evaluation only with background codewords in M. At a later stage, after evaluating the above-mentioned parameters in parallel, the codewords in cache H which fulfill the condition in parameter T_{add} will be added to the background model M.

4.6 Blob detection

After conducting BGS, the system generates a binary mask image in which 0 and 1 represent background and foreground, respectively. This binary mask image might include noise and individual objects decomposed in multiple units; this is due to the moving object having some similar colors to the background. To remove noise and connect the decomposed objects again, morphological operations (erosion–dilation) are applied to the binary mask image, making use of the resource-optimized architecture described by Hedberg et al. [8], where a low complexity architecture using structuring element decomposition is proposed.

Finally, once morphological operations have been conducted, the binary mask image contains groups of connected pixels representing different relevant objects (blobs). To separate and differentiate these groups (labeling stage), we will use the efficient run-length encoder (RLE) algorithm described in Appiah et al. [2], which associates each pixel to one label placing it into a particular group. Both of these post-processing stages are included in the approach described in this paper and therefore in the evaluation of accuracy described in the next section.

5 Results

This section shows the results obtained by the proposed implementation. Since this approach is a hardware implementation which targets low power devices, we have analyzed the performance of the algorithm regarding system resource consumption and real-time constraints, as well as an objective accuracy evaluation based on the Wallflower dataset [28].

5.1 System resources and performance

For the sake of hardware feasibility, hardware resources constraints must be taken into account. In Sect. 3, we have established the optimal number of codeword processing units so that the accuracy of the model is not compromised and the consumption of hardware resources is affordable (see Fig. 4). In Fig. 11, we can see the consumption of resources in the FPGA due to the maximum number of codewords per codebook. In this case, the maximum number of codewords per codebook is five. In addition, input variables have been configured with their optimal representation: [I:8, F:5] for $(\bar{R}_i R + \bar{G}_i G + \bar{B}_i B)$ and [I:9, F:3] for $(\check{I}, \hat{I})$. The number of slices grows generally in an exponential way. This is mainly caused by the non-linear stages whose expansion does not follow a linear trend. Note that we have used dedicated DSP48 multipliers on the system. Each processing codeword unit uses 16 DSP48 multipliers and, therefore, the BGS module has 80 DSP48 for all the Cx and 3 DSP48 which are used to calculate $\|x_t\|^2$, thus having a total of 83 DSP48. Note that we have used automatic inference of DSP48 multipliers on the

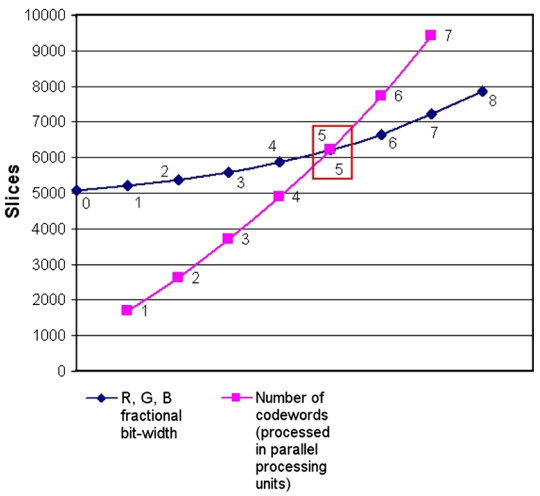

Fig. 11 Consumption of resources in the FPGA due to the maximum number of codewords per codebook. Our choice has been marked with a *red rectangle*, which corresponds to the configuration described in Sect. 3

system, which is the best option, avoiding the use of look-up table (LUT), as DSP48 inference usually achieves higher clock rates than LUT inference with considerably less logic.

Figure 11 also shows the impact of different data bit-width chosen for the $v_i = (\bar{R}_i R + \bar{G}_i G + \bar{B}_i B)$ and $\|x_t\|^2$ on the consumption of resources in the FPGA.

ViSmart video processing board has a DIMM DDR2 memory module whose configuration (detailed in Sect. 4) enables a large bandwidth (920 MB/s, empirically tested), which is equally distributed for the tasks of codeword reading and writing (460 MB/s for each task). Since the system performance in terms of frames per second depends directly on the access to the memory module, enabling large bandwidth is a key feature to reach a high frame rate. With the proposed organization of dynamic memory, the total amount of memory required for a particular scene may vary significantly, depending on the number of necessary codewords. If the required amount of memory changes, the maximum frame rate will change too. Bearing all this in mind, the total number of codewords is an important factor to determine the processing performance of the system. A scene with multiple backgrounds (waving trees) will have more codewords than a scene with a static background with hardly any change. Thus, the computational capability of the system can be determined from the average number of codewords per pixel. For example, with a 460 MB/s reading bandwidth for the DDR2 memory, each codeword requiring 16 B, an image resolution of 1,024 × 1,024, and being aware that the average of codewords is 1.20, the frame rate is computed as follows (9).

$$\frac{\text{Bandwidth DDR2}}{\text{Image resolution} \times \frac{\text{bytes}}{\text{codeword}} \times (\text{Avg of codewords/pixel})} = \frac{460\,\text{MB/s}}{1{,}024 \times 1{,}024 \times 16 \times 1.20} = 23.958\ \text{fps} \approx 24\ \text{fps} \tag{9}$$

With image resolution of 768 × 576, the architecture reaches a frame rate between 50 and 60 fps depending on the average of codewords. The tested image sequences belong to the performance evaluation of tracking and surveillance (PETS) [23] test and Wallflower [28] databases.

Note that the BGS IP core is modular and scalable, enabling the reduction of the system's parallelism (and performance) to fit onto smaller devices. Xilinx XC3SD3400aFG676 is a low-cost FPGA [3], but we could use cheaper FPGAs with less logic resources (XC3SD1800A-4CSG484C). For example, if we reduce the codewords processing units to only one without reducing the maximum number of possible codewords (5 in our case), we would have to process sequentially each one of them to determine which is the appropriate one. This loss of performance will be steeper in sequences whose pixels are associated with a higher number of codewords. Thus, considering the worst case scenario, we have a factor 5 performance loss.

The whole system has different operating frequencies (clock domains). Microblaze and the system buses operate at a frequency of 66.5 MHz, ethernet at 25 MHz, the input video modules at 13.5 MHz and, finally, the DDR2 interface at 133 MHz. The maximum processing frequency of our IP core *background subtraction* is 67.1 MHz; however, we have configured it at 66.5 MHz (the same as Microblaze's) to avoid a higher complexity of the FPGA clock distribution networks.

The system has been implemented and tested in the ViSmart video processing board of Seven Solutions S.L. [26], using the Xilinx XC3SD3400aFG676 FPGA. Resource utilization of the entire implementation is summarized in Table 2, taking into account that this total resource consumption is determined based on the design decisions taken earlier (bit-width optimization and maximum number of codewords per pixel).

We have also analyzed power consumption with the Xpower tool [31] and reported the results in Table 2. According to this value, the architecture can be used to implement a stand-alone system and work on embedded applications with reduced space occupation and low power consumption compared to other approaches as high-performance processors (software solutions) and GPUs. Our system can afford at the same time three advantages: low power consumption, high performance and physical size.

We have also addressed the evaluation of the real hardware in comparison with software simulation to check the final system degradation.

Table 3 shows the total errors obtained by the proposed architecture, tested with the simulator and the real board, as

Table 2 Complete hardware resources required on a Xilinx XC3SD3400aFG676 FPGA after place and route

	Total system	Background subtraction
Slices	17,337 (73 %)	6,215 (26 %)
DSP48s	95 (75 %)	83 (66 %)
BRAM	44 (35 %)	0 (0 %)
DDR2 interface frequency (MHz)	133	
Microblaze/PLB frequency (MHz)	66.5	66.5 (Max. 67.1)
Power (W)	5.13	

The whole system includes processing modules (background subtraction and blob detection core), peripherals (ethernet, SSRAM ZBT, DDR2), interconnect buses (PLB, NPI, MCH) and a Microblaze processor

Table 3 Total error differences between simulation and real hardware results, in number of pixels and percentage

Test	Simulation	Hardware	Diff	%
B	2,100	2,203	103	0.54
C	3,069	3,179	110	0.57
FA	2,806	2,848	42	0.22
LS	6,391	6,428	37	0.19
MO	4	10	6	0.03
TD	1,192	1,185	7	0.04
WT	682	917	235	1.22

well as the difference between them and the percentage of different pixels. From these results, it can be seen that the degradation is really small, <0.6 % in every test except for waving trees. These differences are due to restriction of the software simulator to emulate fixed-point arithmetic of the hardware system.

5.2 Performance comparison with other approaches

The implementation of BGS algorithms has been addressed by several authors in recent years, using different approximations. Jiang et al. [15] proposed an architecture to implement the MoG model. This architecture provides a calculation capacity allowing real-time processing of relatively large RGB images 1,024 × 1,024 at a frame rate of 38 fps. To reduce the large amount of required memory, a compression scheme was proposed, using similarities for Gaussian distributions in adjacent areas. In this way, they have managed to save up to 60 % of the required memory bandwidth. Although that approximation achieved high processing speed, there is the disadvantage of a considerable loss of accuracy, due to the compression scheme in use. Other relevant factors, such as the consumption of resources, cannot be compared with our approach, since the author does not provide these data.

Another architecture proposed by Appiah et al. [1] uses a single-chip FPGA for the segmentation of moving objects in a video sequence. The system performs 209 fps for 640 × 480 frame size and 145 fps for 768 × 576 frame size in RGB. In Sect. 5.3), we have evaluated (software simulator) this simplification of MoG and it got worse results than the approach described here.

Our system has been experimentally tested on real hardware and, although it has a lower frame rate, it is able to segment objects in complex sequences with resolution 768 × 576 at 50 fps or at 24 fps with resolution 1,024 × 1,024 as shown in (9), along with much higher accuracy.

5.3 Evaluation of the background subtraction method

In this work, a quantitative evaluation of the proposed algorithm has been performed so that it can be compared with other approaches in the literature. To perform a quantitative analysis instead of qualitative, the use of a benchmark database with information about the ground truth is required. For that reason, the Wallflower dataset has been used to evaluate the algorithms.

The algorithms which have been used for this comparison are mixture of Gaussians (MoG) [27], a segmentation method based on Bayes decision rules [21], the original codebook model [19] and two hardware-friendly approaches, a simplification of MoG for FPGAs [1] and a static algorithm based on Horprasert et al. [9] and Karaman et al. [18]. These models have been selected since they represent different kinds of algorithms and they are among the most frequently used.

The implementations of MoG and the Bayesian algorithm that have been used are versions from the OpenCV library [22], while the hardware-oriented approaches have been developed from the information shown in their respective papers. Each algorithm parameter has been fixed for the entire benchmark to avoid over-fitting this specific dataset.

To evaluate and compare various BGS algorithms, relative measures have been calculated based on true and false positives and negatives (TP, FP, TN and FN): *Recall*, *Precision*, F_1 and *Similarity* [10, 18].

Recall is the true positive rate $R = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$, and it evaluates the capability of the algorithm to detect true positives. *Precision* is the ratio between the number of correctly detected pixels and the total number of pixels marked as foreground $P = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$, being an estimation of the capability to avoid false positives. These metrics, despite offering objective evaluation regarding the sensitivity of the algorithm to true positives and false positives, respectively, are not reliable separately. For example, an algorithm classifying every pixel as foreground would have maximum *Recall*, although with many false positives. For that reason, there are two accuracy metrics, F_1 and *Similarity*, which combine *Precision* and *Recall* to evaluate an overall quality of the segmentation.

$$F_1 = 2\frac{PR}{P+R} \tag{10}$$

$$Similarity = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{11}$$

These measures offer a balance between the ability of an algorithm to detect relevant and non-relevant pixels and have been widely used in the literature.

The Wallflower benchmark [28] consists of seven sequences which test different capabilities from BGS algorithms: “Bootstrap”, “Camouflage”, “Foreground Aperture”, “Light Switch”, “Moved Object”, “Time of Day”

and "Waving Trees". Each of these sequences contains one evaluation frame with ground truth information. This frame is chosen so that it evaluates the performance against one specific difficulty. For example, "Time of Day" sequence tests adaptation of the algorithm to gradual lighting changes, while "Waving Trees" contains quasi-periodic movements in the background, which tests the multi-modal behavior of the algorithm.

The test "Moved Object" cannot be evaluated using these metrics, since the ground truth does not have any foreground pixel. As a result, there are not true positives (TP) or false negatives (FN), being impossible to compute *Precision* or to get useful results from *Similarity*. For that reason, the performance in this test is only studied in a qualitative manner, by observing the resultant images (Fig. 14).

Figure 12 shows the F_1 and *Similarity* values obtained by each algorithm in the dataset. From this figure, we can see that the Codebook algorithm gets very good results, being the best algorithm in two of the tests, and obtaining very high marks in the others. Concerning hardware approach presented in this paper, the accuracy decreases minimally so that it gets slightly worse results than the original software one (even though we have performed many modifications for the sake of hardware resources reduction). However, these results are in a similar order with respect to the software implementation, and are far better than the results offered by any other hardware-oriented solutions available in the literature, such as the simplification of MoG or the Horprasert implementation.

Considering the comparison with the hardware-oriented algorithms [1, 9] our approach is better in all tests except for "Light Switch", since this test requires fast adaptation to sudden changes, and the adaptation rate has been fixed to do this analysis. Thus, the implementation presented in this work represents an improvement in accuracy over the other state-of-the-art approaches.

Besides the general comparison, it is interesting to see the accuracy of these algorithms in outdoor and indoor situations. Instead of constructing figures for all the tests of the benchmark, we have grouped the sequences in two groups according to the characteristics of each one, and the results are obtained as a weighted average of the results of each test. In the indoor group, we have taken into account the sequences "Bootstrap", "Foreground Aperture" and "Light Switch". In the outdoor group, the sequences are "Camouflage", "Time of Day" and "Waving Trees".

Figure 13 shows that the quality of the segmentation provided by the proposed implementation is similar to the one provided by the original algorithm, with results similar to the obtained by Bayesian [21] and MoG [27] models, and that this approach offers a great improvement against other hardware solutions. In general, a loss of accuracy is shown in the "Indoor" situation, due to bad results of the algorithms

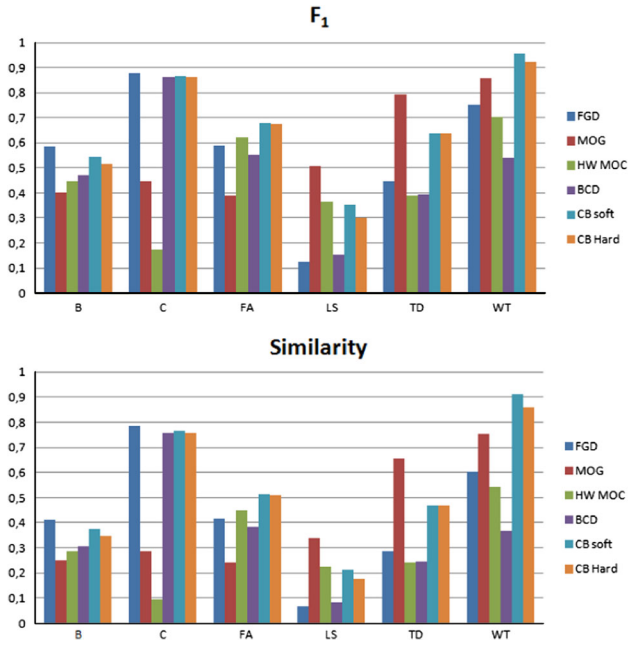

Fig. 12 The overall performance evaluated using F_1 and *Similarity*. FGD is the Bayesian algorithm [21], MOG the mixture of Gaussians [27], HWMOC the FPGA implementation [1], BCD is the approach by Horprasert et al.[9], and CB soft [19] and CB hard the original and proposed implementations of codebook

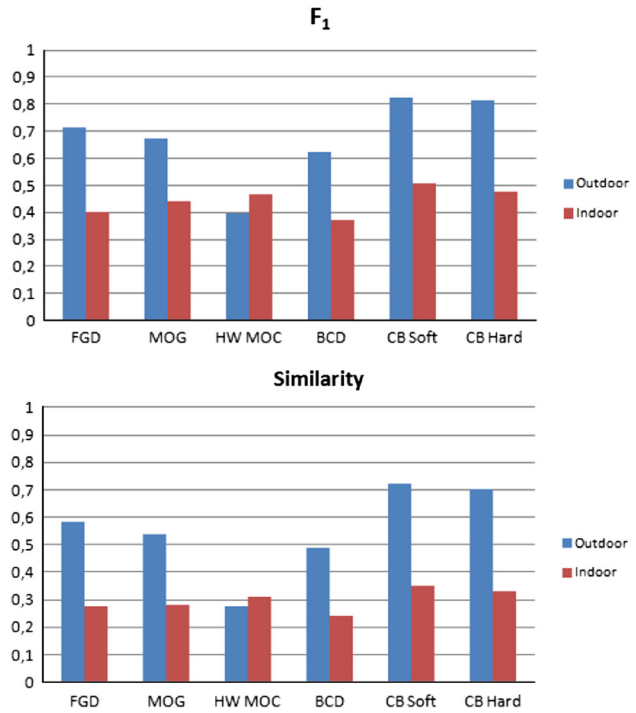

Fig. 13 Performance in outdoor and indoor circumstances

during the "Light Switch" test. However, the results obtained by our approach are more accurate than the rest.

Finally, it is also important to remark that the final hardware implementation has a very low degradation compared with the original software approach [19]. From Fig. 12 it can be seen that "Bootstrap", "Light Switch" and "Waving Trees" sequences show bigger difference between both approaches, since those tests require a higher number of codewords than average. Nevertheless, even with that degradation, the results obtained by the proposed architecture represent an improvement against the other hardware-oriented implementations and are similar to the results offered by the more advanced algorithms.

In Fig. 14, the evaluation images resultant from each algorithm on the Wallflower dataset are displayed, as well as the original frame and the ground truth to evaluate the quality of the segmentation. This comparison allows us to see in a qualitative manner the foreground–background segmentation quality of the different approaches. This subjective evaluation supports the conclusions of the quantitative analysis. Regarding the "Moved Object" test, only a small region of <10 pixels has been misclassified by the hardware approach, being an unimportant mistake since this small area could be removed by subsequent morphological filtering.

6 Conclusions

In this work, we have designed and analyzed an architecture to carry out the BGS in video sequences. Our approach is based on the Codebook algorithm, which allows us to model dynamic and multimodal backgrounds and is well known because of its robustness and good balance between accuracy and efficiency.

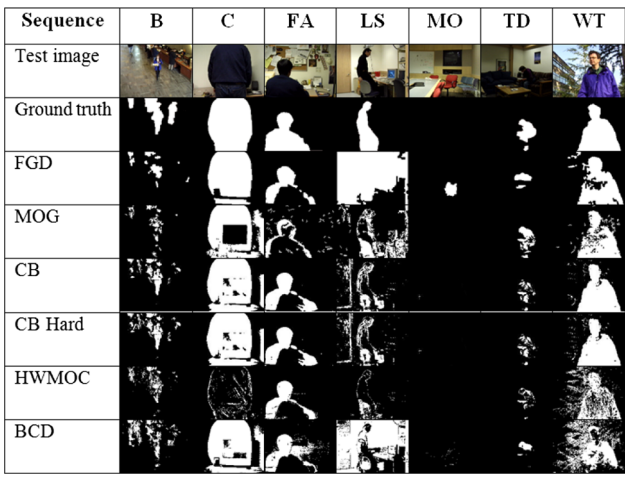

Fig. 14 Wallflower evaluation frames, ground truth and resultant images from tested algorithms

The design techniques presented in this paper are of general use and valid to many resource-constrained hardware implementations. A detailed study has been performed to improve the trade-off between accuracy and computing resources, including model simplifications and fixed-point operations.

An FPGA implementation of this algorithm has been developed which offers very low degradation in comparison with the original software solution. Since the hardware environment has limited resources, we have optimized memory access, low-level interfaces with external memory and storage of the background model. Thus, the proposed architecture can use the hardware resources more efficiently without a relevant decrease of accuracy. We have also addressed the evaluation of the real hardware in comparison with software simulation to check the final system accuracy degradation.

We have evaluated the approach with the benchmark Wallflower to test the quality of the segmentation. The results are excellent in comparison with other hardware-oriented approaches found in the literature, being similar to the ones offered by advanced software algorithms such as Bayesian and MoG. The implementation is able to segment objects in complex sequences with resolution 768×576 at 50 fps or at a higher speed with less resolution sources. Concerning the cost of the system, the architecture has been designed for low-cost FPGAs Spartan-3 by Xilinx, with an estimated power consumption of 5.13 W.

Future work will include improvements over the codebook model with the incorporation of spatio-temporal context described in [30], as well as the refinement of the blob detection stage by using the architecture from [5]. Furthermore, the use of faster memory (DDR3, QDR-II) or FPGAs with on-chip memory, such as Virtex-6, could remove limitations related to bandwidth, allowing further processing with the same board.

Acknowledgments The authors would like to thank Seven Solutions S.L. [26] for their valuable help with the design of the FPGA platform. This work was supported by the projects of excellence from Junta de Andalucía (TIC-3873, TIC-5060), the national projects DINAM-VISION (DPI2007-61683) and ARC-VISION (TEC2010-15396) and the EU Project TOMSY (FP7-270436).

References

1. Appiah, K., Hunter, A.: A single-chip FPGA implementation of real-time adaptive background model. In: Proceedings of the IEEE International Conference on Field-Programmable Technology, pp. 95–102 (2005)
2. Appiah, K., Hunter, A., Dickinson, P., Owens, J.: A run-length based connected component algorithm for FPGA implementation. In: International Conference on ICECE Technology, FPT 2008, pp. 177–184 (2008)

3. Avnet (FPGA Distributor) http://www.avnet.com (2011). Accessed 14 Oct 2011
4. Cao, T., Elton, D., Deng, G.: Fast buffering for FPGA implementation of vision-based object recognition systems. J. Real Time Image Process. (2011). doi:10.1007/s11554-011-0201-1
5. Déforges, O., Normand, N., Babel, M.: Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture. J. Real Time Image Process. 1–10. doi:10.1007/s11554-010-0171-8
6. Elgammal, A., Harwood, D., Davis., L.: Non-parametric model for background subtraction. In: Vernon, D. (ed.) Computer Vision ECCV 2000. Lecture Notes in Computer Science, vol. 1843, pp. 751–767. Springer, Berlin (2000)
7. Happe, M., Lnbbers, E., Platzner, M.: A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. J. Real Time Image Process. (2011). doi:10.1007/s11554-011-0212-y
8. Hedberg, H., Kristensen, F., Nilsson, P., Owall, V.: A low complexity architecture for binary image erosion and dilation using structuring element decomposition. In: IEEE International Symposium on Circuits and Systems, ISCAS 2005, vol. 4, pp. 3431–3434 (2005)
9. Horprasert, T., Harwood, D., Davis, L.S.: A statistical approach for real-time robust background subtraction and shadow detection. In: IEEE Frame-Rate Applications Workshop, Kerkyra, Greece (1999)
10. Huang, S.C.: An advanced motion detection algorithm with video quality analysis for video surveillance systems. IEEE Trans. Circuits Syst. Video Technol. **21**(1), 1–14 (2011)
11. Ierodiaconou, S., Dahnoun, N., Xu, L.: Implementation and optimisation of a video object segmentation algorithm on an embedded dsp platform. In: The Institution of Engineering and Technology Conference on Crime and Security, pp. 432–437 (2006)
12. Ilyas, A., Scuturici, M., Miguet, S.: Real time foreground–background segmentation using a modified codebook model. In: Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS '09, pp. 454–459 (2009)
13. Impulse Accelerated Technologies. http://www.impulseaccelerated.com (2011). Accessed 14 Oct 2011
14. Javed, O., Shafique, K., Shah, M.: A hierarchical approach to robust background subtraction using color and gradient information. IEEE Workshop Motion Video Comput. pp. 22–27 (2002)
15. Jiang, H., Ardo, H., Owall, V.: Hardware accelerator design for video segmentation with multi-modal background modelling. In: IEEE International Symposium on Circuits and Systems, ISCAS 2005, vol. 2, pp. 1142–1145 (2005)
16. Jiang, H., Ardo, H., Owall, V.: A hardware architecture for real-time video segmentation utilizing memory reduction techniques. IEEE Trans. Circuits Syst. Video Technol. **19**(2), 226–236 (2009)
17. Jodoin, P.M., Mignotte, M., Konrad, J.: Statistical background subtraction using spatial cues. IEEE Trans. Circuits Syst. Video Technol. **17**(12), 1758–1763 (2007)
18. Karaman, M., Goldmann, L., Yu, D., Sikora, T.: Comparison of static background segmentation methods. In: Proceedings of the SPIE 5960, 596069, vol. 5960 (2005)
19. Kim, K., Chalidabhongse, T.H., Harwood, D., Davis, L.: Real-time foreground–background segmentation using codebook model. Real Time Imaging **11**(3), 172–185 (2005) (special Issue on Video Object Processing)
20. Kohonen, T.: Learning vector quantization. Neural Netw. **1**, 3–16 (1988)
21. Li, L., Huang, W., Gu, I.Y.H., Tian, Q.: Foreground object detection from videos containing complex background. In: Proceedings of the Eleventh ACM International Conference on Multimedia, ACM, New York, NY, USA, MULTIMEDIA '03, pp. 2–10 (2003)
22. OpenCV Library. http://opencv.willowgarage.com/wiki/ (2011). Accessed 14 Oct 2011
23. PETS. http://www.cvg.rdg.ac.uk/slides/pets.html (2007). Accessed 14 Oct 2011
24. Ripley, B.D.: Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge (1996)
25. Rodriguez-Gomez, R., Fernandez-Sanchez, E.J., Diaz, J., Ros, E.: FPGA implementation for real-time background subtraction based on Horprasert model. Sensors **12**(1), 585–611 (2012)
26. Seven Solutions S.L. http://www.sevensols.com (2011). Accessed 14 Oct 2011
27. Stauffer, C., Grimson, W.: Adaptive background mixture models for real-time tracking. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vols. 2, xxiii+637+663, pp. 2 (1999)
28. Toyama, K., Krumm, J., Brumitt, B., Meyers, B.: Wallflower: principles and practice of background maintenance. IEEE Int. Conf. Comput. Vis. **1**, 255 (1999)
29. Trieu, D., Maruyama, T.: Real-time image segmentation based on a parallel and pipelined watershed algorithm. J. Real Time Image Process. **2**, 319–329 (2007)
30. Wu, M., Peng, X.: Spatio-temporal context for codebook-based dynamic background subtraction. AEU Int. J. Electron. Commun. **64**(8), 739–747 (2010)
31. Xilinx. http://www.xilinx.com (2011). Accessed 14 Oct 2011
32. Xilinx MPMC. http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf (2011). Accessed 14 Oct 2011
33. Zhong, B., Hong, X., Yao, H., Shan, S., Chen, X., Gao, W.: Texture and motion pattern fusion for background subtraction. In: Proceedings of the 11th Joint Conference on Information Sciences (2008). Accessed 14 Oct 2011

Author Biographies

Rafael Rodriguez-Gomez received his B.Sc. degree in 2005 and his M.Sc. degree in 2007 from the University of Granada, Spain. Currently, he is a Ph.D. student at the University of Granada. His main research interests include video surveillance, high-performance image processing architectures and embedded systems based on reconfigurable devices.

Enrique J. Fernandez-Sanchez received his B.Sc. degree in 2006 and his M.Sc. degree in 2009 from the University of Granada, Spain. Currently, he is a Ph.D. student at the University of Granada. His main research interests include video analytics, high-performance computer vision, video segmentation and multi-camera networks.

Javier Diaz received his M.S. in electronics engineering in 2002 and a Ph.D. in electronics in 2006, both from the University of Granada. Currently, he is assistant professor at the Department of Computer Architecture and Technology at the same university. His main research interests are cognitive vision systems, high-performance image processing architectures, and embedded systems based on reconfigurable devices. He is also interested in safety-critical systems, biomedical/bioinspired devices and robotics.

Eduardo Ros received his Ph.D. degree in 1997 from the University of Granada. He is currently Full Professor at the Department of Computer Architecture and Technology at the same University. He is currently a responsible researcher at the University of Granada of two European projects related to bio-inspired processing schemes and real-time image processing. His research interests include hardware implementation of digital circuits for real-time processing in embedded systems and high-performance computer vision.