Computer Science Journal of Moldova, vol.**12**, no.1(34), 2004

Real Time Embedded Moving Objects Detector – Study and Implementation Within Virtex FPGA Technology

Messaoud Mostefai, Samira Djebrani, Youssef Chahir

Abstract

This article details the design and the implementation of an efficient real time moving objects detector in the field of view of a fixed camera. The developed algorithm allows a robust motion detection and a considerable reduction in hardware resources, more particularly memory resources, which remains a powerful criterion in the construction of real time embedded systems. Moreover, the algorithm maps efficiently into a highly pipelined architecture, well suited to an implementation in reconfigurable technology. The algorithm is implemented on a Virtex FPGA architecture and operates in real time on video in CCIR format. The obtained experimental results show the effectiveness and correctness of our approach.

Keywords: image processing, motion detection, fuzzy logic, real time, virtex technology.

1 Introduction

With the aim to satisfy the real time processing requirements, various image processing systems proceed to an important data reduction in order to compensate the insufficient power of current processors. On the other hand, in many applications (tracking, surveillance, guidance...) alone the position and form of objects is necessary, consequently the alone knowledge of moving edges is sufficient. Computer vision is only one of the possible approaches to extract moving objects. Other approaches have been followed based on inductive loops, sound detectors, pneumatic sensors. Nevertheless, none of these sensors is able to extract information as detailed as computer vision is.

Motion detection requires the analysis of image sequences, in order to trace the evolving position of single objects. Most approaches are based on motion extraction from images: on one hand, many analytical methods of motion analysis have been adopted, like the optical flow [1, 2] or the matching of moving features [3]. These methods are able to extract not only moving points, but also information on the motion direction and velocity. However, these methods are generally very expensive in terms of computational time. The approach we follow, extracts moving points from images with simple and efficient image processing techniques, well suited for a real time implementation with low cost hardware.

To proceed to a moving edges analysis, we use an operator which applied to an image sequence, allows to extract from a dynamic scene the edges of objects in movement. For this purpose, different operators have been developed [4, 5, 6, 7]. They often use in their processing classical edges detection techniques, easy to implement but sensitive to noise. The relative failure of the classical detectors has urged us to choose another segmentation approach that uses fuzzy tools to perform a fuzzy edges detection. This approach is actually applied in many areas (process control, expert system ...), fuzzy logic brings also its evidence in image processing, segmentation, image enhancement and edge detection [8, 9, 10, 11].

The system we propose here has been implemented on a low-cost reconfigurable architecture based on Virtex Field Programmable Gate Arrays (FPGAs). These architectures are sufficiently flexible to permit the implementation of new algorithms on existing hardware, and their performance is adequate for real time operation of many image processing problems.

After a brief introduction to moving detection techniques, we present the basic principles of the fuzzy approach and its application in the construction of an edges detector. This detector is validated by comparison with Canny detector. Software motion detection results are presented as well as the details of the implementation within a reconfigurable Virtex FPGA architecture for a real time image processing. A solution based on the search of moving segments instead of moving pixels is proposed to reduce the effect of noise without eliminating the true edges points. Finally, real time results are reported and some conclusions are drawn.

2 Motion detection techniques

Two categories of methods are used for the detection of moving objects in a sequence of images: those that make a comparison of the current image with a reference image which represents the static model of the filmed scene [4, 5] and those which compare the current image with the neighboring images by concentrating directly on zones where changes are observed [6, 7]. The construction of a reference image is an often difficult task, because variations of lighting within the analyzed sequence can make some images incompatible with the reference image. Haynes and Jain [4] suggest an operator that performs the product of the absolute difference of two successive images (I_{n-1} and I_n) with the gradient of the current image I_n. The expression of this operator is described by the following formula:

$$\forall x, y \;\; R_n(x,y) = |I_n(x,y) - I_{n-1}(x,y)| * G_n(x,y) \tag{1}$$

with G_n being the gradient of image I_n and R_n the result of the operation which corresponds to the moving edges. This operator does not perform a coincidence between edges but rather between the gradient of the current image and the absolute value. Another operator proposed by Stelmaszyk [5], consists in a multiplication between the gradient of the difference of two successive images (I_{n-1} and I_n) and the gradient of the current image I_n, allowing thus to have a true coincidence between edges. The formulation of this operator is as follows:

$$\forall x, y \;\; R_n(x,y) = G(|I_n(x,y) - I_{n-1}(x,y)|) * G_n(x,y) \tag{2}$$

In general, operators that operate only on two consecutive images induce false edges. Indeed, in this case, any visible object on the current image which was not found on the one before is considered as mobile, even if it concerns an element of the static bottom that was hidden, and that comes to appear (Figure 1).

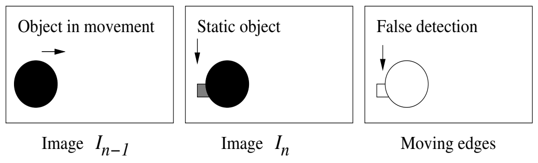


Figure 1. Occultation problem

In order to avoid this problem, the processing must operate on three consecutive images. Vieren [6] has proposed an operator which makes a product of the absolute differences of couples (I_{n-1}, I_n) and (I_n, I_{n+1}). The formulation of this operator is as follows:

$$\begin{aligned}\forall x, y \quad R_n(x,y) = G(|I_n(x,y) - I_{n-1}(x,y)|)*\\ *G(|I_{n+1}(x,y) - I_n(x,y)|)\end{aligned} \tag{3}$$

Another operator using three consecutive images in its processing was also proposed by Orkisz [7]. This operator allows mobile frontiers in the current image to be located by comparison of frontiers of the current image with those of next and preceding images so as to distinguish mobile elements from static ones. The expression of this operator is as follows:

$$\begin{aligned}\forall x, y \quad R_n(x,y) = max(G_n(x,y), G_{n-1}(x,y), G_{n+1}(x,y))-\\ -max(G_{n-1}(x,y), G_{n+1}(x,y))\end{aligned} \tag{4}$$

with G_{n-1}, G_n and G_{n+1} respective gradients of three consecutive images I_{n-1}, I_n and I_{n+1}. Figure 2 shows an example of the application of the Orkisz operator to extract moving objects from three successive images (with a static object and a moving circle).

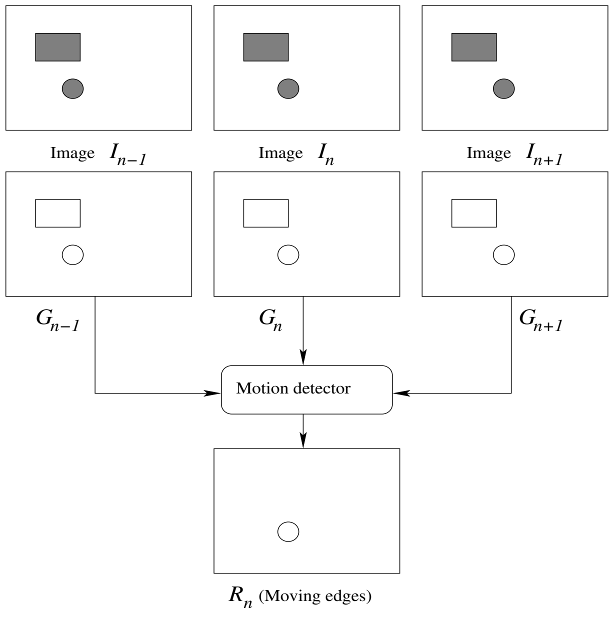


Figure 2. Motion detection based on three successive frames

From all known operators that use varying elements between neighbor images, we have chosen the Orkisz operator; it represents the best compromise between efficiency and simplicity of implementation. Indeed, compared to other operators (e.g., Vieren operator) which consist in performing products of gradients, Orkisz operator requires only comparison and subtraction operations; easy to implement and do not affect the dynamic as does multiplication operation. As all the other operators, this operator uses a classical edge detection technique, such as Sobel filtering, Prewitt, Laplacian or Gaussian filtering, the Canny and Deriche operator, but some common problems of these methods are a large volume of computation, sensitivity to noise, anisotropy and thick lines. Russo and Ramponi [8, 9] designed fuzzy rules for edge detection. Such rules can smooth while sharpening edges, but require a rather large rule set compared to simpler fuzzy methods presented by

Looney in [10]. Neural networks have also been used in [11] to detect edges, but here we use a special fuzzy classifier for edges that does not require training.

3 Fuzzy classification

A fuzzy classifier is a system that accepts inputs that are either feature vectors or vectors of fuzzy truths for the features to belong to various fuzzy set membership functions (FSMFs). It outputs fuzzy truths for the memberships of the input vector in the various classes. The class assigned to an input feature vector is the one with the maximum fuzzy truth given by the FSMFs. We usually require the maximum to exceed the second greatest fuzzy truth by a certain amount in order to yield a unique class membership. Otherwise we can only say that the input feature vector belongs to each class with a particular fuzzy truth. Different types of fuzzy classifiers are used in [12, 13, 14] for other purposes.

3.1 Methodology

Consider an image I_n having gray levels in the interval [0,G-1]. It is assumed that each processed pixel $I_0(x, y)$ has eight closest neighbors (I_i, i=1,8). The partitioning process consists on computing for each central pixel I_0 the difference luminance set: $D = \{d_i\}$, with ($d_i = I_i - I_0$). These values will be used as input variables by the fuzzy edges detector to compute the new central value. The input variable d_i can be member of several classes with membership degrees varying between 0 and 1. The membership functions can have several forms, however triangular and trapezoidal forms are the most used, allowing thus to have easy algorithms to implement. In our case three linear membership functions are used (Figure 3). The data partitioning operation allows to allocate each input variable to one of the three classes (class0, class1 or class2) by using the function $Max(\mu_p, \mu_g, \mu_c)$. Where μ_p, μ_g and μ_c correspond to the membership functions of class0, class1, and class2 respectively. The d_i values belonging to the class2 are called dynamic values and will be arranged in one of the two static classes "0" or "1" only if their two close neighbors belong to the same class "0" or "1"; otherwise, all configuration including these values is not taken into account, and the correspondent central pixel is considered as a noisy pixel. After the data partitioning operation the operator will use the previously generated fuzzy rules and the input variables allocations to compute the new central pixel value (mapped to black if the central pixel is considered as a contour point and mapped to white if not). To elaborate our fuzzy

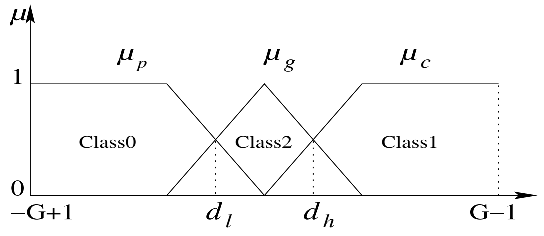


Figure 3. Used membership functions

rules, we have used the popular IF-THEN approach well described in [15]. This approach uses a group of N THEN rules (ending to the same result) and one ELSE rule. The fuzzy rules generation is undertaken from a 3x3 size binary mask. The eight examined contour types are presented in figure 4. An operator is used to examine the considered point neighborhood. In case one of the eight configurations is met, the considered central point is taken for a contour point, otherwise it is considered as belonging to a uniform zone.

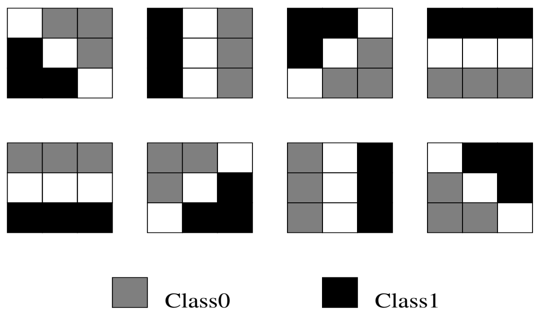


Figure 4. Examined contour types

4 Simulation results

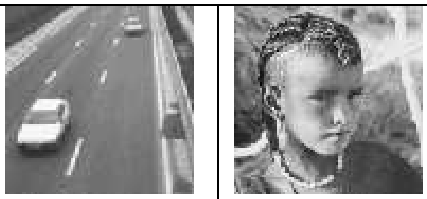
4.1 Fuzzy approach validation

In this study we are interested to real world scenes which have the reputation to be noisily and with weak dynamic. To validate our approach, the developed fuzzy edges detector is compared with Canny edges detector. All of our results were obtained by using a 3x3 neighborhood centered in turn on each interior pixel. The center parameters d_h and d_l must be provided to achieve good results. In practice, different people may look for different details in the same image, so those parameters should be input by the user to obtain the desired type of edges. For example, we could put $d_l = -4$, $d_h = 4$. A smaller $[d_l, d_h]$ interval yields more sensitivity to edges (and displays more noise), whereas a larger interval maps more of the weak edges to the background.

To obtain results for the Canny edge detector we used Matlab 6 (Release 12). The results are white edges on black background, so we take the negative for comparison with our fuzzy operator. The Matlab command for Canny edge detection is: Output Image = edge(Input Image ,'Canny', T, σ); where "Output Image" and "Input Image" are the respective output edge image and input image. The Canny parameters that must be input by the user are the upper threshold T (upper edge sensitivity) and sigma σ (the Gaussian parameter). The default value is 1.0. We found that a smaller threshold T gives more detail (and noise). Making σ smaller also gives more detail but without the noise. T is the most sensitive.

Test images presented in Figures 5.a and 5.b are used to compare the contours detection results of the two operators. The first test figure shows an outdoor traffic scene with moving vehicles. The sequence is shot in daylight conditions from a camera placed on a bridge passing above a highway. The second test figure shows an image of a south girl acquired in daylight conditions. Test images are 256x256 in size and 8 bits in quantization.

The results obtained with Canny edges detector are shown for both high and low sensitivity, respectively, in figures 6.a and 6.b for the traffic image and figures 6.e and 6.f for south girl image with σ fixed



(a) Traffic scene (b) South girl

Figure 5. Test images

at 0.5. Decreasing σ would show more details without increasing the noise significantly. The results for our fuzzy edges detector for the lower and higher sensitivity to edges are presented in figures 6.c and 6.d for traffic image and figures 6.g and 6.h for south girl image. In our judgement, which is a subjective decision, our fuzzy edges detector yields moderately thin black lines even when the edge in the input image is diffuse.

4.2 Motion Detection

Three successive images I_{n-1}, I_n and I_{n+1} are used to extract the edges images G_{n-1}, G_n and G_{n+1} using the fuzzy edges detector. These last are used by the motion detection operator to extract moving edges. Different results depending on d_l and d_h values are presented in figure 7. A smaller $[d_l, d_h]$ interval yields more sensitivity to edges and displays more noise, whereas a larger interval maps more of the weak edges to the background. In addition, a same scene behaved to different results in natural light and in artificial light; shade of objects induce wrong contours, notably for objects in movement. From a theoretical point of view, this is a correct decision since the motion detection operator is expected to detect changes that are not due to noise. In a surveillance system, the detection of shadows is annoying, since it modifies the coherence of the shape of the object.

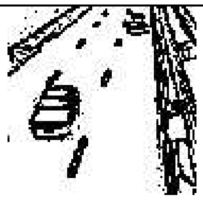
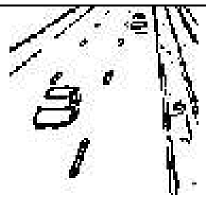
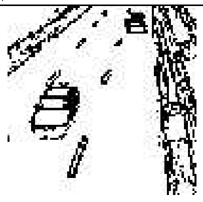
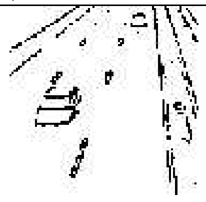
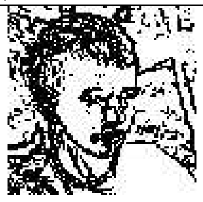
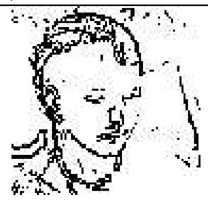
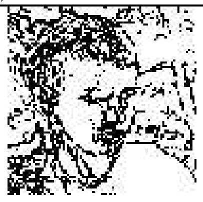
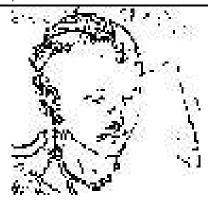
 (a) T=0.05, $\sigma = 0.5$	 (b) T=0.2, $\sigma = 0.5$
 (c) $d_l = -2$, $d_h = 2$	 (d) $d_l = -4$, $d_h = 4$
 (e) T=0.05, $\sigma = 0.5$	 (f) T=0.2, $\sigma = 0.5$
 (g) $d_l = -2$, $d_h = 2$	 (h) $d_l = -4$, $d_h = 4$

Figure 6. Simulation results

4.3 Moving segments detection

To reduce the shadow and noise effects without eliminating true edges points we propose to associate to the motion detection operator an intra image filter based on the search of moving segments instead of moving pixels. We define a edge segment as a whole of 3 adjacent edge pixels (figure 8.a). The procedure to extract a moving segment is as follows: A candidate segment is considered as a edge segment if it only exist in the current image and not in preceding and following images. An example of case where a candidate segment is validated as a moving segment is presented in figure 8.b. As the processing is on 3x3 size masks, two line delays and six data registers are therefore necessary for each image gradient, allowing thus a simultaneous access to the nine data of each processed mask. It is important to note that this operator cannot be used in case of perfectly rectilinear displacements, which fortunately, are extremely rare in reality. Figure 9 shows the obtained results with the two methods on the traffic road sequence.

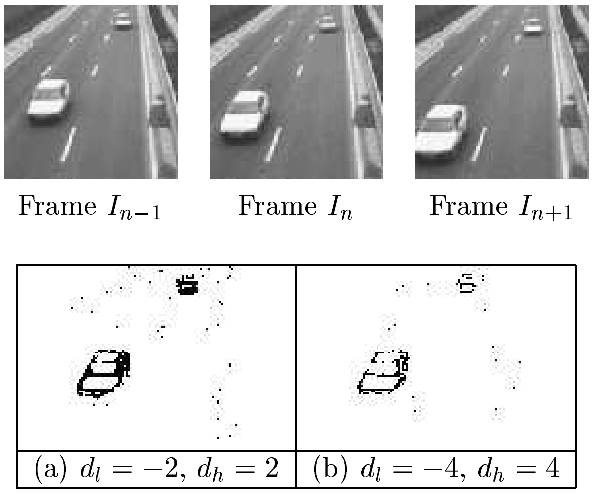


Figure 7. Moving edges detection

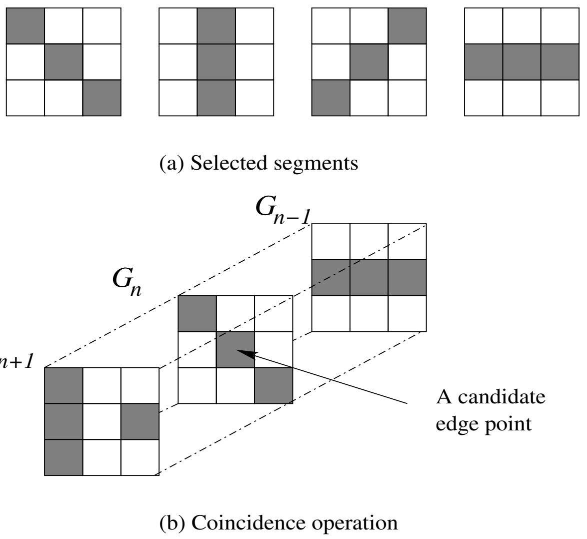
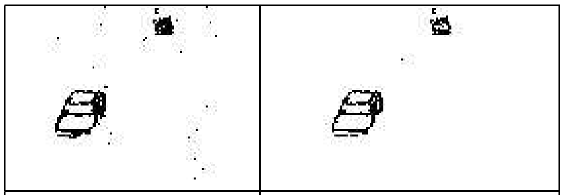


Figure 8. Moving segments detection

5 Hardware Implementation

A system designer is constantly faced with a tradeoff between performance and generality: a digital system designed to handle different types of tasks is usually slower than a system tailored to a specific application. This is the reason why computationally intensive tasks are handled by dedicated system architectures usually implemented in Application Specific Integrated Circuits (ASIC's). Although they offer more than adequate performance, ASIC's often lack the ability to adapt themselves to a changing environment. Moreover, ASIC design has high non-recurring costs and requires large engineering effort. A new methodology for the implementation of digital logic circuits, based on Field Programmable Gate Arrays (FPGA), emerged during the mid 80's.

The basic architecture of a FPGA consists of a large number of Configurable Logic Blocks (CLB) and a programmable mesh of inter-



(a) Moving pixels (b) Moving segments

Figure 9. Moving pixels and segments detection

connections. Both the function performed by the logic blocks and the interconnection pattern can be specified by the circuit designer. FPGA allow large-scale parallel processing and pipelining of data flow. Latest FPGA provide enormous processing resources, significant on-chip RAM and support very high clock speeds [16, 17, 18]. FPGA are therefore suitable for implementing real time image processing algorithms.

5.1 Implementation details

The complexity of the system dictated the use of modern logic synthesis tools throughout all design phases. Synopsys FPGA Compiler [19] has been used for most part of the logic design starting from descriptions specified in the VHDL hardware description language. Design of FPGA based computing machines is carried out through the following steps :
1. Definition of data processing resources;
2. Definition of memory resources;
3. Design of the control logic.

The result of the first step depends on a tradeoff between the complexity of the algorithm and the available logic and communication resources, e.g. the number of FPGA devices, their capacity, and the number of interconnection lines between programmable devices. The processing resources required by this application (adders, subtractors, comparators) are automatically inferred by the synthesis tool from VHDL code and mapped to elements of the Xilinx XBLOX library.

The basic memory resource required by any FPGA based image processing system operating on pixel neighborhoods is the pixel line delay, that is usually implemented either with external RAM memory devices or with FPGA internal memory. The Xilinx Virtex FPGA architecture, in fact, allows to configure every CLB as a 32 x 1 bit shift register, and several registers can be cascaded in order to build FIFO memories of various widths and depths. The integration of the FIFO on the same piece of silicon does not only mean increased performance but also a decrease in the number of external components [20, 21, 22].

This application has been validated on a real time image development system (figure 10) based on series 150/151 boards, compatible with the VME bus [23]. FPGAs which are cabled on our specific applications card (SAC) use various initialization and framing signals in order to operate in the field of validity of the video signal. The general structure of the moving segments detector is presented in figure 11. In order to perform a real time processing on 3x3 size masks, two delay lines and six data registers are used, allowing thus a simultaneous access to the nine video data of the processed mask. Three memory plans (F_1, F_2 and F_3) of the frame buffer card (FB) are used in cascade to store the binary edges images G_{n-1}, G_n and G_{n+1} required by the motion detection operator.

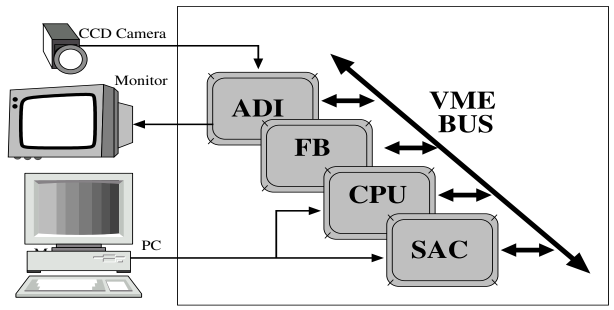


Figure 10. Development system

In order to fit the 2x8 bit wide delay lines in a single Virtex XCV400 device, we had to downsample the CCIR 512 X 512 image by a factor of 2 along the x axis. Due to interlacing, the video decoder sends each field one at a time, so the system is actually processing 256 x 256 pixel images.

The fuzzy edges detector module allows to extract binary edges. The obtained three consecutive binary edges pictures (G_{n-1}, G_n, G_{n+1}) are used to perform the moving segment detection operation. In order to perform a real time processing on 3x3 size masks, two delay lines and six data registers are necessary for each output memory plan, allowing thus a simultaneous access to the nine binary edges data of the processed masks necessary for a moving segment detection.

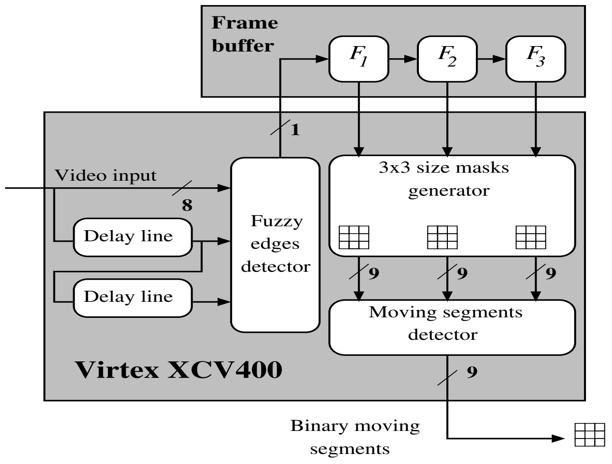


Figure 11. Moving segments detector

The fuzzy motion edges detector has been implemented within FPGA Virtex XCV400. The cost in surface evaluated by the number of CLBs used depends directly on the input pixels format (Figure 12).

Pixels Format	Gray Levels	Surface Cost (Clbs)
6 bits	64	1012
7 bits	128	1346
8 bits	256	1727

Figure 12. Implementation costs

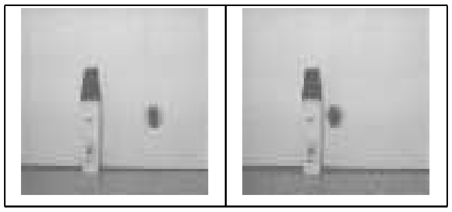
5.2 Real Time tests

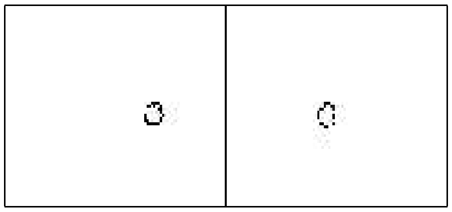
Some real time results obtained with our system are presented in figure 13. First tests are carried out on a pendulum (metal part suspended by a genuine silver wire) (figure 13.a). The use of the pendulum allows to have a periodic movement in the field of view of the camera (figure 13.b), and makes adjustment easiest than with walking persons. Second tests are carried out on a car propelled by a spring (figure 13.c). It is important to note that the magnitude of the additive noise is largely linked to the type of chosen lighting. As mentioned before, shade of objects (figure 13.d) induce wrong contours notably for objects in movement. A uniform lighting of the scene and a smoothing operation of acquired images are therefore necessary to obtain noiseless contours and more representative of objects in movement.

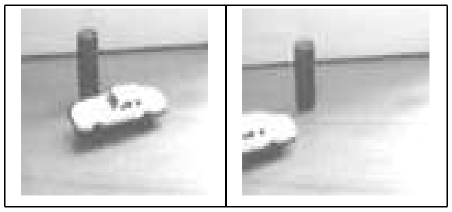
Other tests are made with daylight scene involving humans walking naturally in a straight line (figure 13.e). Some best snapshot results obtained with different people are presented in figure 13.f. The moving segment detection method gives good results, because even if the human is walking in a straight line, the body movement is not.

6 Migration Towards a fully embedded system

In order to have an autonomous system (principal embedded systems characteristic), it is necessary to re-examine the application components in terms of memory and I/O resources which remains integral part of the real time development system. Indeed, acquisition and dis-


(a) Real time snapshot frames (pendulum scene)


(b) Real time snapshot moving edges


(c) Real time snapshot frames (car scene)

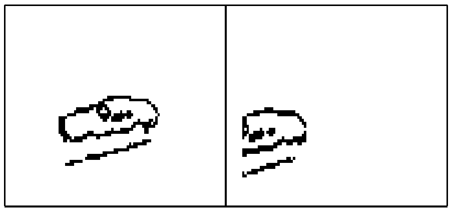
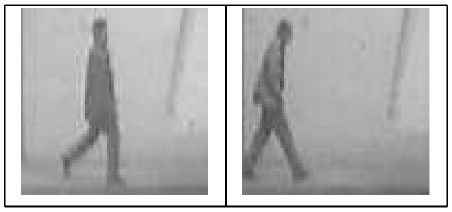

(d) Real time snapshot moving edges

Figure 13. Real time tracking results


(e) Real time snapshot frames (humans walking)

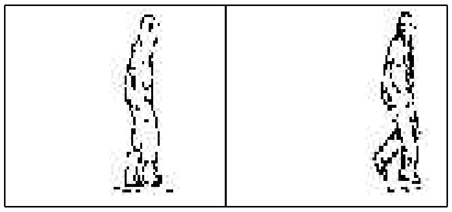

(f) Real time snapshot moving edges

Figure 13. Real time tracking results (continuation)

play is assumed by the ADI card, while storage of binary edges images is made by the Frame Buffer card (FB). However, even the significant on-chip RAM provided in Xilinx Virtex FPGAs [21] (81920 bits in the case of XCV400) is not sufficient to support a useful level of internal RAM frame buffering in the object detection application described.

For the first fully embedded prototype system, we have used for video acquisition and display the high speed A/D-D/A converter (UVC 3130) from ITT Semiconductor [24]. Four high speed dual ports memories (256x256x1bit) [25] are used to store binary edges images required by the moving segments detector. The necessary control logic used to manage the UVC 3130 and memories is mapped on the Virtex XCV400.

7 Conclusion and perspectives

In this paper we have presented the design and implementation of a real time fuzzy moving objects detector within a Virtex FPGA. This detector is based on the integrated use of both spatial and temporal information of the video sequence. Experimental results indicate that the proposed fuzzy operator and refinement based on moving segments detection significantly reduce the noise effect without eliminating the true moving edges. The reconfigurable aspect of FPGA circuits offers a great flexibility in the adjustment of the architecture and the algorithm to implement. Technology advances in the area of embedded memory on FPGAs is particularly attractive to video and image processing applications. Indeed, the new features offered by Virtex-II Pro in term of capacity of integration, and diversity of implemented modules (ADC, DAC, memory banks etc) enables us to implement the whole application on the same chip. We are currently working on the migration towards this new family. Finally, we can say that the development of robust motion detectors can not be validated only by software. Real time tests with real dynamic scenes must be done. FPGA technology allowed us to make a quick prototyping to validate our approach and can even support the final application in case where the migration toward ASICs is not envisaged.

References

[1] K. Nakajima, A. Osa, T. Maekawa, H. Mike, Evaluation of body motion by optical flow analysis, Japanese Journal of Applied Physics, Part 1, 36 (5A), pp. 2929-2937, 1997.

[2] Beauchemin. S, Barron. J, The computation of optical flow, ACM Computing Surveys, 27(3), pp. 433-466, 1995.

[3] S. Smith, J. Brady, Real time motion segmentation and shape tracking, IEEE Trans PAMI, 17(8), pp. 814-820, 1995.

[4] S. Haynes, R. Jain, Detection of moving edges, Computer Graphics Image Process, Vol 21, pp. 345-367, 1983.

[5] P. Stelmaszyk, P. Bonnet, J.G. Postaire, Analyse de scenes dynamiques par recherche des contours en mouvement, congres Reconnaissance de Formes et Intelligence Artificielle AFCET / ADI / INRIA, Grenoble (France), pp. 1181-1189, 1985.

[6] Ch. Vieren, J.G. Postaire, P. Bonnet, P. Deparis, Detection du contour exterieur d'objets en mouvement sur fond non uniforme, Colloque GRETSI Traitement du Signal et ses applications, Juan les pains, pp. 621-624, juin 1989.

[7] M. Orkisz, Localisation d'objets mobiles dans des scenes filmées par une caméra fixe, Revue traitement du Signal, Volume 9, N 4, pp. 325-346, 1992.

[8] F. Russo, A user-friendly research tool for image processing with fuzzy rules, Proc. First IEEE Int. Conf. Fuzzy Systems, San Diego, pp. 561-568, 1992.

[9] F. Russo, G. Ramponi, Fuzzy operator for sharpening of noisy images, IEE Electronic. Letters, 28, pp. 1715-1717, 1992.

[10] C. G. Looney, Nonlinear rule-based convolution for refocusing, Real-Time Imaging 6, pp. 29-37, 2000.

[11] C. Anagnostopoulos, J. Anagnostopoulos, D. Vergados, E. Kayafas, V. Loumos, V. Stassinopoulos, A neural network and fuzzy logic system for face detection on RGB images, Proc. ISCA Int. Conf. Computers and Their Applications, pp. 233-236, 2001.

[12] L. R. Liang, E. Basallo, C. G. Looney, Image edge detection with fuzzy classifier, Proc Of the ISCA 14'h International Conference, Las Vegas, pp. 279-283, 2000.

[13] T. L. Huntsgerger, C. L. Jacobs, R. L. Cannon, Iterative Fuzzy Image Segmentation, Pattern recognition, Vol 18, N2, pp. 131-138, 1985.

[14] H. M. Lozoya, D. M. Rodriguez, F. J. Romero, H. Tawfik, Handoff algorithms based on fuzzy classifiers, IEEE Trans. Vehicular Tech, vol. 49, no. 6, pp. 2286 -2294, 2000.

[15] F. Russo, A New Class of Fuzzy Operators for Image Processing, Design and Implementation, Proc Of Intl Conf on Fuzzy Systems, San Francisco, pp. 494-501, 1993.

[16] A. Dehon, The Density advantage of configurable computing, IEEE on computer, pp. 41- 49, 2000.

[17] J. Valls, M. Kuhlmann, K. K. Parhi, Efficient mapping of CORDIC algorithms on FPGA, Proc of the 2000 IEEE Workshop on Signal Processing Systems (SiPS) Design and Implementation, Lafayette, LA, pp. 336-345, 2000.

[18] L. Nozal, G. Aranguren, J. Martin, J. Ezquerra, Moving images time gradient implementation using RAM based FPGA, SPIE Proceedings, Vol. 3028, San Jos, California, pp. 108-116, 1997.

[19] Synopsys, FPGA Compiler User Guide v3.5, Mountain View, CA, 1996.

[20] Xilinx Data Book, 2001.

[21] Xilinx XAPP 151 (v1.5),September, 2000.

[22] Sawyer. N, Self addressing fifo, XAPP 291 (v 1.1), 2002.

[23] Imaging Technology Series 150/151 Boards, 1990.

[24] UVC 3130 Converter, ITT Semiconductors Data Book, 1990.

[25] Cypress Data Book, San Jose, 1996.

Messaoud Mostefai, Samira Djebrani, Youssef Chahir Received November 25, 2003

Messaoud Mostefai,
Computer Science Department,
UFR Setif, Algeria
E–mail: *bbamos@wissal.dz*

Samira Djebrani
Computer Science Department,
UFR Setif, Algeria
E–mail: *samirad76@yahoo.fr*

Youssef Chahir
University of Caen,
BP 5186, 14032, France
E–mail: *chahir@info.unicaen.fr*