Proceedings of Engineering & Technology (PET)
Copyright IPCO-2016
pp. 52-59

FPGA Implementation of Real-Time System for Detection of Human Activity

Kamal Sehairi [#1], Cherrad Benbouchama [*2], Chouireb Fatima[#3], Kobzili El Houari [*4]

[#]*Laboratoire LTSS, Université Amar Telidji Laghouat*
Route de Ghardaia, Laghouat, Algérie 03000
[1]sehairikamel@yahoo.fr, k.sehairi@lagh-univ.dz
[3]chouirebfatima@yahoo.fr

[*]*Laboratoire LMR, École Militaire Polytechnique*
Borj El Bahri, Alger, Algérie
[2]ben_cherrad@yahoo.fr
[4]kobzili_elhouari@yahoo.fr

***Abstract*— This paper proposes a PixelStreams-based FPGA implementation of a real-time system that can detect and recognize human activity using Handel-C, including all the stages, i.e., capture, processing, and display, using DK IDE of Mentor Graphics. The targeted circuit is an XC2V1000 FPGA embedded on Agility's RC200E board. The PixelStreams-based implementation was successfully realized and validated for real-time motion detection and recognition.**

***Keywords*— detection and recognition, FPGA, real-time implementation, High level synthesis, video surveillance.**

I. Introduction

In modern society, there is a growing need for technologies such as video surveillance and access control to detect and identify human and vehicle motion in various situations. Intelligent video surveillance attempts to assist human operators when the number of cameras exceeds the operators' capability to monitor them and alerts the operators when abnormal activity is detected. Most intelligent video surveillance systems are designed to detect and recognize human activity. It is difficult to define abnormal activity because there are many behaviors that can represent such activity. Examples include a person entering a subway channel, abandonment of a package, a car running in the opposite direction, and people fighting or rioting. However, it is possible not only to set criteria to detect abnormal activity but also to zoom in on the relevant area to facilitate the work of the operator.

In general, an intelligent video surveillance system has three major stages: detection, classification, and activity recognition [1]. Over the years, various methods have been developed to deal with issues in each stage.

II. Related Work

Many methods for motion detection have already been proposed. They have been classified [1]–[3] into three major categories: background subtraction, [4],[5] temporal differencing [6] , [7] and optical flow[8],[9]. Further, motion detection methods have been recently classified into matching methods, energy-based methods, and gradient methods. The aim of the motion detection stage is to detect regions corresponding to moving objects such as vehicles and human beings. It is usually linked to the classification stage in order to identify moving objects. There are two main types of approaches for moving object classification:[1],[2],[10] shape-based identification and motion-based classification. Different descriptions of shape information of motion regions such as representations of points, boxes, silhouettes, and blobs are available for classifying moving objects. For example, Lipton *et al.*[11] used the dispersedness and area of image blobs as classification metrics to classify all moving object blobs into human beings, vehicles, and clutter. Further, Ekinci *et al.*[12] used silhouette-based shape representation to distinguish humans from other moving objects, and the skeletonization method to recognize actions. In motion-based identification, we are more interested in detecting periodic, non-rigid articulated human motion . For example, Ran *et al.*[13] examined the periodic gait of pedestrians in order to track and classify it. The final stage of surveillance involves behavior understanding and activity recognition. Various techniques for this purpose have been categorized into seven types: dynamic time warping algorithms, finite state machines, hidden Markov models, time-delay neural networks, syntactic techniques, non-deterministic finite automata, and self-organizing neural networks. Such a wide variety of techniques is attributable to the complexity of the problems and the extensive research conducted in this field. The computational complexity of these methods and the massive amount of information obtained from video streams makes it difficult to achieve real-time performance on a general-purpose CPU or DSP. There are four main architectural approaches for overcoming this challenge: application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs), parallel computing, GPUs, and multiprocessor architectures. Evolving high-density FPGA architectures, such as those with embedded multipliers, memory blocks, and high I/O (input/output) pin counts, are ideal solutions for video processing applications [14]. In the field of image and video processing, there are many FPGA implementations for motion segmentation and tracking. For example, Menezes *et al.* [5] used background subtraction to detect vehicles in motion, targeting Altera's Cyclone II FPGA with Quartus II software. Another similar study on road traffic detection [15] adopted the sum of absolute differences (SAD) algorithm, implemented on Agility's RC300E board using an XC2V6000 FPGA with Handel-C and the PixelStreams library of Agility's DK Design Suite. Other methods for motion detection such as optical flow have been successfully implemented [8],[9] on an FPGA. For example, Ishii *et al.*[8] optimized an optical flow algorithm to process 1000 frames per second. The algorithm was implemented on a Virtex-II Pro FPGA.

Many video surveillance systems have been developed for behavior change detection. For example, in the framework of ADVISOR, a video surveillance system for metro stations, a finite state machine (with scenarios) [16] is used to define suspicious behavior (jumping over a barrier, overcrowding, fighting, etc.). The W4 system [17] is a system for human activity recognition that has been implemented on parallel processors with a resolution of 320×240. This system can detect objects carried by people and track body parts using background detection and silhouettes. Bremond and Morioni [18] extracted the features of moving vehicles to detect their behaviors by setting various scenario states (toward an endpoint, stop point, change in direction, etc.); the application employs aerial grayscale images.

The objective of this study is to implement different applications of behavior change detection and moving object recognition based on motion analysis and the parameters of moving objects. Such applications include velocity change detection, direction change detection, and posture change detection. The results can be displayed in the RGB format using chains of parallelized sub-blocks. We used Handel-C [19] and the PixelStreams library [20] of Agility's DK Design Suite [21] to simplify the acquisition and display stages. An RC200E board with an embedded Virtex-II XC2V1000 FPGA [22] was employed for the implementation.

III. Outline of the Algorithm

A. Detection Algorithm

We choose to implement the delta frame method for three reasons: its adaptability to changes in luminance, its simplicity, and its low consumption of hardware resources. This method determines the absolute difference between two successive images, and it is executed in two stages: temporal difference and segmentation.

1) Temporal difference

In this stage, we determine the absolute difference between the previous frame and the current frame as follows.

$$\zeta(x,y) = \left|\frac{dI(x,y)}{dt}\right| = \left|\Delta_{t,t-1}(x,y)\right| = \left|I_t(x,y) - I_{t-1}(x,y)\right| \quad (1)$$

where $\zeta(x,y)$ is the difference between $I_t(x,y)$ (i.e., the intensity of pixel (x,y) at moment t) and $I_{t-1}(x,y)$ (i.e., the intensity of pixel (x,y) at moment t-1).

2) Segmentation

In this stage, significant temporal changes are detected by means of thresholding:

$$\Psi(x,y) = \begin{cases} 0 & \text{if } \zeta(\text{x,y}) < \text{Th} \\ 1 & \text{otherwise} \end{cases} \quad (2)$$

This operation yields a binary card that indicates zones of significant variations in brightness from one image to the other.

B. Feature Extraction and Behavior Change Detection

In this study, simple behavior change detection refers to motion that can be caused by abrupt movements that might represent suspicious actions. To define these actions, we use the parameters of the objects in motion, such as the center of gravity, width, and length. In general, the actions detected by this method are simple yet useful in video surveillance. For example, velocity change detection is useful for detecting a criminal who is being chased by the police or a car that exceeds the speed limit; direction change detection is useful for detecting a car that is moving in the wrong direction; and posture change detection is useful for detecting a person who bends to place or pick up an object.

Our implementation involves the following stages: acquisition of the video signal, elimination of noise from the input video signal, detection of moving regions, segmentation for separating the moving objects, extraction of the object parameters, classification of the moving objects, and determining whether movements are suspicious.

1) Velocity change detection

We can detect suspicious behavior of a person from his/her gait as well as his/her change in velocity near sensitive locations such as banks, airports, and shopping centers. In such cases, we can calculate the speed (in pixels/s) or acceleration (in pixels/s^2) of the suspect in the image space in real time. There are several ways of representing this anomaly: the most widely adopted method in the literature is the use of a bounding box (a rectangle around the suspect).

It is easy to calculate the speed of a moving object. As soon as the speed or acceleration of the object exceeds a certain threshold of normality (predetermined experimentally or on the basis of statistical studies), a bounding box appears around the suspect. However, the issue that needs to be addressed is the calculation of the speed in real-time circuits owing to the absence of mathematical functions (such as square root), types of data (integer or real values), and the object parameters on which we base our calculation.

In general, the speed and acceleration are calculated as follows:

$$velocity(t) = \left(\sqrt{(x_g(t) - x_g(t-dt))^2 + (y_g(t) - y_g(t-dt))^2}\right)/dt \quad (3)$$

$$acceleration(t) = \left(velocity(t) - velocity(t-dt)\right)/dt \quad (4)$$

where $\text{x}_g(t), \text{y}_g(t)$ and $\text{x}_g(t-dt), \text{y}_g(t-dt)$ are the co-ordinates of the center of gravity of the object at moments t and $t-dt$, respectively, *dt=40ms* in our case, and $velocity(t)$ and $velocity(t-dt)$ are the velocities of the object at moments t and $t-dt$, respectively.

2) *Direction change detection*

To determine the change in direction, we select parameters that distinguish the object of interest, such as its center of gravity, width, and length. In general, the co-ordinates of the center of gravity can be used to determine whether the object has changed its direction, i.e., whether it has moved rightward or leftward depending on the position of the camera.

The change in direction along the x-axis is given by

$$x_g(t) - x_g(t-dt) \begin{cases} > 0 & \text{The object did not change direction} \\ < 0 & \text{The object changed direction} \end{cases} \quad (5)$$

The change in direction along the y-axis is given by

$$y_g(t) - y_g(t-dt) \begin{cases} > 0 & \text{The object did not change direction} \\ < 0 & \text{The object changed direction} \end{cases} \quad (6)$$

These techniques, which are based on the object parameters, can be improved by integrating them with advanced models such as finite state machines (FSMs).

IV. Hardware Implementation

Figure 2 shows the general outline of our FPGA implementation.

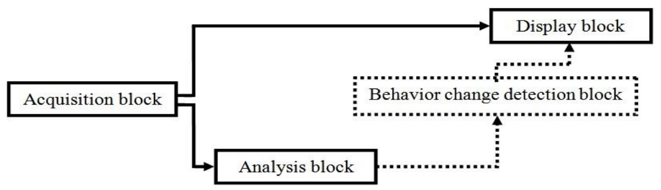


Fig. 2 General outline of behavior change detection.

This general outline consists of four blocks: an acquisition block, an analysis block, a display block, and an intermediate block between the display block and the analysis block.

A. *Acquisition Block*

Acquisition is achieved using the standard camera associated with the RC200E board. The video input processor, Philips SAA7113H, acquires the frames in the PAL format at a rate of 25 fps. The pixels are in the YCbCr format. Using the PixelStreams library of Agility's DK Design Suite, we split the input video signal into two identical streams (see Fig. 3).

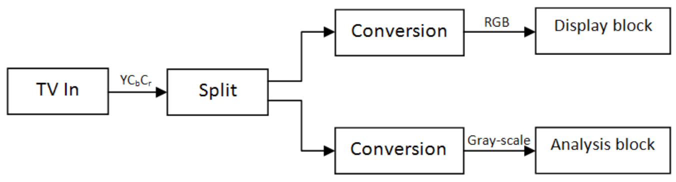


Fig. 3 Acquisition block.

The first stream is fed to the display block and it is converted into the RGB format to display the results on a VGA display. The second stream is fed to the analysis block and it is converted into the grayscale format to reduce (by one-third) the amount of data to be processed.

B. *Analysis Block*

The analysis block consists of several stages. In the first stage, we use inter-image subtraction (delta frames) and apply thresholding to detect moving regions.

To obtain the delta frames, we start by splitting the video signal in three channels (see Fig. 4). The first and second channels are used to save the acquired image, creating a delay cell. The image I(t-1) is recorded in the memory. The third channel is used to acquire the actual frame at moment t. Then, the two image streams are synchronized and fed to the subtraction block. The subtraction block is a modified block that takes the absolute result of subtraction and compares it with a threshold. This function is realized using a macro. The threshold value Th is fixed according to the luminosity of the scene.

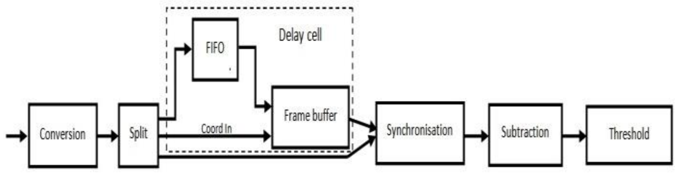


Fig. 4 Motion detection block.

The second stage of the analysis block involves statistical analysis. In this stage, we search for the min and max values along the x- and y-axes of the mobile regions (Fig. 5). In general, this stage must be preceded by a filter for noise reduction. We employed a morphological filter (e.g., alternating sequential filter, opening/closing filter) using the PixelStreams library.

After calculating the min and max values along the two axes, we determine the center of gravity of the detected object. We calculate the sum of the pixel co-ordinates that have non-zero values along the x- and y-axes, and we divide these coordinate values by their sum. However, for our implementation, it is better to avoid this division. Therefore, we use the direct method. We subtract the max from the min and divide the result by 2. Division by 2 is achieved by a simple bit shift (right shift). Once the values minX, MaxX, minY, MaxY, and X_G, Y_G are obtained, we copy these values into the behavior change detection block. Then, we reset these values to zero.

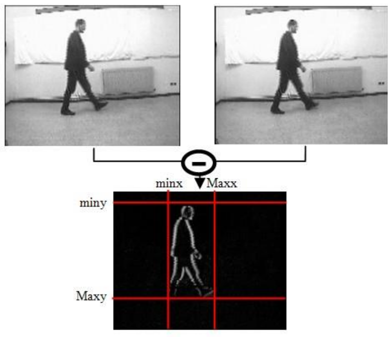


Fig. 5 frame difference and calculation of min and max values.

C. *Behavior Change Detection Block*

As stated in the previous section, the analysis block provides the behavior change detection block with the parameters of the moving objects. In this stage, we save the values extracted from the first delta frame (xg(t-1), yg(t-1), minx(t-1), MaxX(t-1), miny(t-1), MaxY(t-1)), and from the second delta frame, we obtain the current values xg(t), yg(t), minx(t), MaxX(t), miny(t), and MaxY(t). From these latter results, we can calculate the width and length of the moving object to classify the object as human, vehicle, or others, as in our previous work [23]. Using the values extracted in two different instants (t-1, t), we define the changes in behavior.

For velocity change detection, the speed and acceleration are calculated using the two equations presented in Sec. 3.B.1. However, we simplify these equations by calculating the absolute differences between two moments (the previous and current values). If the absolute difference exceeds a certain threshold V_{th}, we assume that the velocity has changed, and we copy the values of the center of gravity in the display block in order to draw a rectangle around the object. Then, the current values are saved as previous values.

Figure 7 shows this implementation and represents all the stages realized.

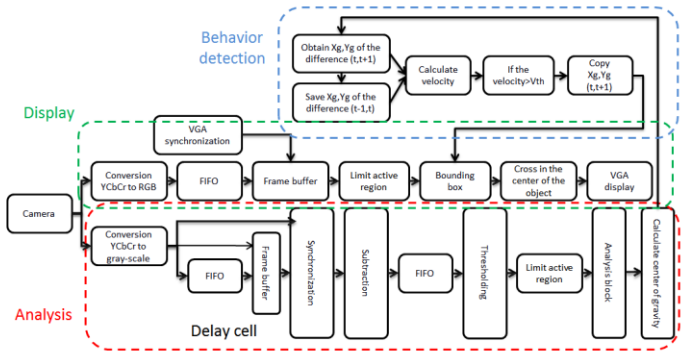


Fig. 7 Hardware architecture for velocity change detection.

Direction change detection: To implement this application, we follow the same stages as those used in velocity change detection, except that the condition changes. We use the same parameters, minX, MaxX, minY, and MaxY, in order to guarantee that the object is entirely entered the scene. We calculate the difference between minX1 and minX2, and MaxX1 and MaxX2. If there is a change in sign, we assume that the object has changed its direction. Otherwise, we assume that the object has not changed its direction.

We can easily determine the direction of motion of an object by applying the same concept as that described above. However, in this case, it is impractical to compare the differences between the previous values and the current values with zero because the presence of a small or non-significant movement (such as that of the arms) can cause false detection. Therefore, to overcome this problem, we compare the difference with a threshold Th_d, which should not be very large. Then, the values minX, MaxX, minY, and MaxY are copied to the block that draws the bounding box.

We use two blocks for detection in two directions (a different color for each direction of motion). In order to minimize resource consumption, we used only one block for drawing the bounding box by changing the parameters of entry in our macro. In this macro, we added a parameter that changes the color according to the direction of detected motion (Fig. 8).

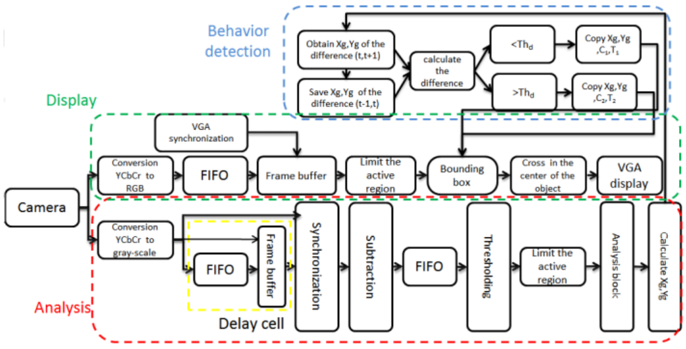


Fig. 8 Hardware architecture for direction change detection.

Posture change detection: We are interested in such an application to detect a person who leans (bends) to place or pick up something, especially in sensitive locations (e.g.,the subways). In this case, we are interested in movements along the y-axis of the image (up/down motion), and we use the same architecture as that used in velocity change detection. We calculate the difference between the previous and current values of miny(t-1), MaxY(t-1), miny(t), MaxY(t).

If the difference between the previous and current values is positive (negative), we assume that the person leans (rises), and we copy the values minX, MaxX, minY, and MaxY to the block that draws the bounding box and fix the color parameter of the rectangle. We can add a warning message using the PxsConsole filter of PixelStreams. As in the case of direction change detection, it is better to use a threshold Th_p to reduce the occurrence of false detection due to small movements along the y-axis. For such detections, we require a camera whose front sight faces the scene.

Motion analysis: Here, we tried to collect all the above-mentioned behaviors using a single program in order to practically validate the system. To minimize resource consumption, we considered our problem as a finite state machine with several scenarios. The thresholds of detection for each case were used to define and manage these various scenarios, these values were obtained by tests. The differences between the values of minX, MaxX, minY, and MaxY at moments t and t-1 are denoted by Δminx, ΔMaxX, Δminy, and ΔMaxY, respectively.

In the first state, all the values are initialized (State 0); they represent the initial state of each new inter-image difference. In the second state (State 1), if the absolute values of Δminx and ΔMaxX are higher than V_{Th}, we assume that the velocity changes and we return to the initial state after copying the values of the block to the bounding box filter. In the opposite case, we go to the third state (State 2) and compare Δminx and ΔMaxX with the threshold Th_d. According to the result of this comparison, we assume that a leftward or rightward movement has occurred. Then, we return to the initial state. Starting from this state, if the moving object accelerates, we return to the second state of velocity change. For posture change detection, the condition is related to the values of Δminy and ΔMaxY (State 3). We can detect this behavior from any state (e.g., a person runs and leans to collect something). The following figure summarizes these states and the possible scenarios.

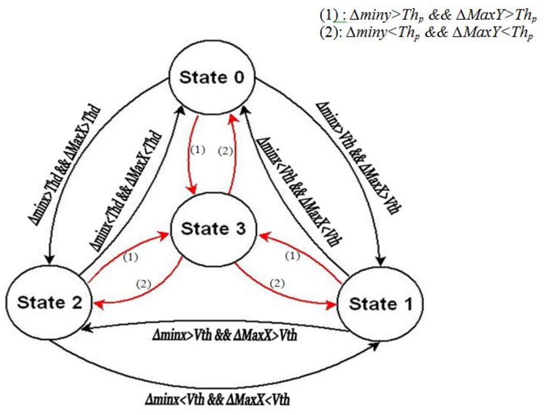


Fig. 9 FSM of motion analysis.

D. Display Block

In this block, we call the macro PxsAnalyseAwaitUpdate, which allows us to pause the display until an update occurs in the analysis block. We obtain the values minX, MaxX, minY, and MaxY; if there is a motion, we copy these values to the bounding box filter to draw the rectangle. The values of the center of gravity, Xg,Yg, are also copied to the PxsCursor filter in order to draw a cross at the center of the moving object. We can add a warning message, e.g., "Warning: velocity change detection", by using the PxsConsole filter of the PixelStreams library. Finally, the results are displayed in the RGB format on a VGA display.

V. Experimental Results

An RC200E board with an embedded Virtex-II XC2V1000 FPGA was used for our implementation. The language used was Handel-C. The results for each behavior are summarized in Tables 2–5:

TABLE 2 RESOURCE CONSUMPTION AND MAXIMUM FREQUENCY OF IMPLEMENTATION FOR VELOCITY CHANGE DETECTION.

Resources	Total	One object	Two objects
I/O	324	179 (55%)	179 (55%)
LUTs	10240	2286 (22%)	3484 (34%)
Slice Flip/Flops	10240	3046 (29%)	3738 (36%)
CLB slices	5120	3092 (60%)	4040 (78%)
Block RAM	40	9 (22%)	9 (22%)
Frequency	/	67.21 MHz 6.17 ms/image	56.85 MHz 7.29 ms/image

These tables specify the resource consumption and maximal frequency of each implemented detection case for PAL video with a resolution of 720×576.

In all these implementations, the results show that the two main constraints, i.e., the resource limit of our FPGA and the real-time aspect (40 ms/image), are well respected. We note that the consumption of the CLB blocks increases in the case of detection of multiple objects; this is caused by the algorithm used to identify the number of objects in the scene. We also note that the algorithm for motion analysis that collects all the previous behaviors have been implemented on our FPGA in real time, but it consumes nearly all of the CLB resources (88%).

TABLE 3 RESOURCE CONSUMPTION AND MAXIMUM FREQUENCY OF IMPLEMENTATION FOR DIRECTION CHANGE DETECTION.

Resources	Total	One object	Two objects
I/O	324	179 (55%)	179 (55%)
LUTs	10240	2052(20%)	2991 (29%)
Slice Flip/Flops	10240	2908(28%)	3491 (34%)
CLB slices	5120	2895(56%)	3698 (72%)
Block RAM	40	9(22%)	9 (22%)
Frequency	/	66.69 MHz 6.22 ms/image	55.12 MHz 7.26 ms/image

TABLE 4 RESOURCE CONSUMPTION AND MAXIMUM FREQUENCY OF IMPLEMENTATION FOR UP/DOWN MOTION DETECTION.

Resources	Total	One object	Two objects
I/O	324	179 (55%)	179 (55%)
LUTs	10240	2100 (20%)	3233 (31%)
Slice Flip/Flops	10240	2927 (28%)	3595 (35%)
CLB slices	5120	2961 (57%)	3904 (76%)
Block RAM	40	9 (22%)	9 (22%)
Frequency	/	67.14 MHz 6.17 ms/image	58.97 MHz 7.03 ms/image

TABLE 5 RESOURCE CONSUMPTION AND MAXIMUM FREQUENCY OF IMPLEMENTATION FOR MOTION ANALYSIS.

Resources	Total	Two objects
I/O	324	179 (55%)
LUTs	10240	3640 (35%)
Slice Flip/Flops	10240	4173 (40%)
CLB slices	5120	4537 (88%)
Block RAM	40	9 (22%)
Frequency	/	53.18 MHz 7.80 ms/image

The following figures show the results of all these implementations. Each behavior is represented by a different color, and a warning message is added below the scenes.

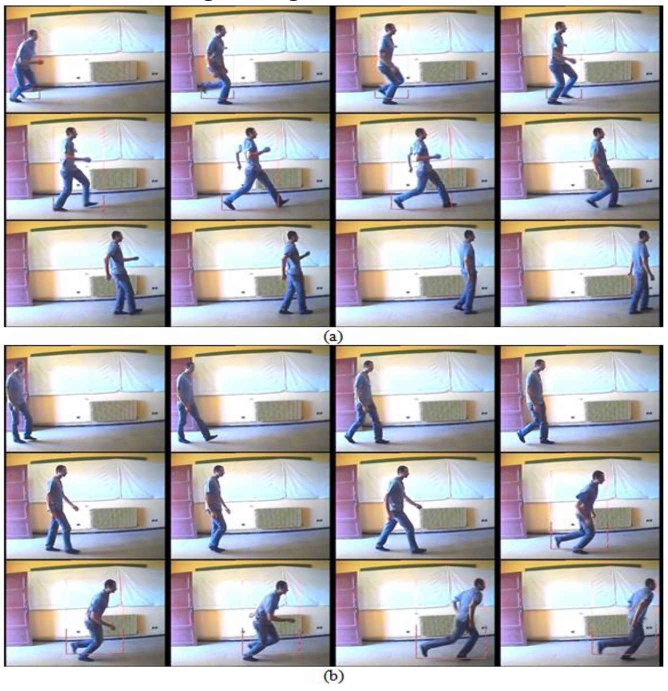


Fig. 10 Results of velocity change detection in the case of one object.

Figure 10 shows the results of velocity change detection in the case of one object. In Fig. 10(a), as soon as the object decreases its speed, the rectangle disappears. In Fig. 10(b), as soon as the object starts to run, a rectangle appears around it.

Figure 11 shows the results of velocity change detection in the case of two objects. As soon as the objects start running, a rectangle appears. We note that in the case of occlusion, the algorithm considers both objects as a single object.

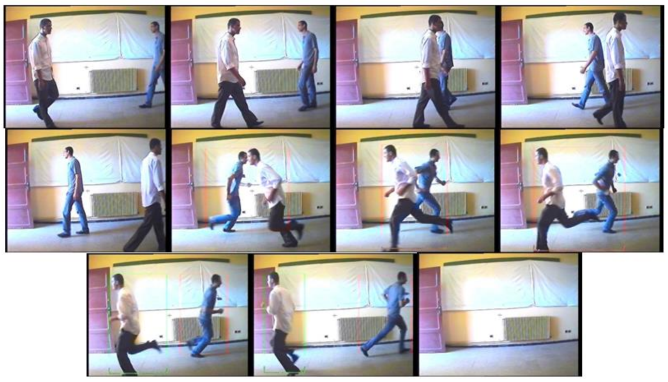


Fig. 11 Results of velocity change detection in the case of two objects.

Figure 12 shows the results of direction change detection. Right to left movement, represented by the blue rectangle, and left to right movement, represented by the red rectangle. The figures also show warning messages below the images.

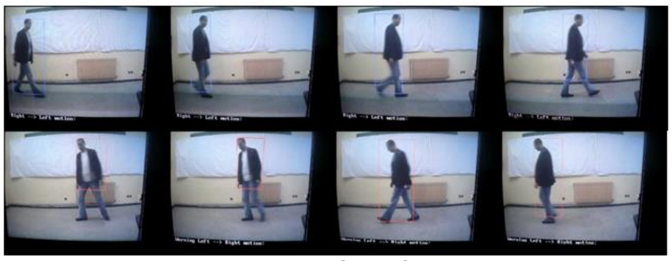


Fig. 12 Direction change detection.

Figure 13 shows the results of posture change detection. When the object leans to pick up something, it will be detected. Up/down and down/up motion are represented in different colors. A warning message is added in each case.

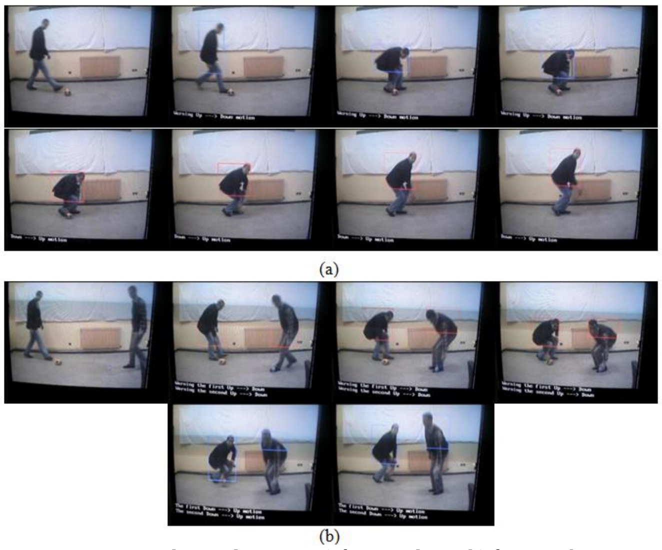
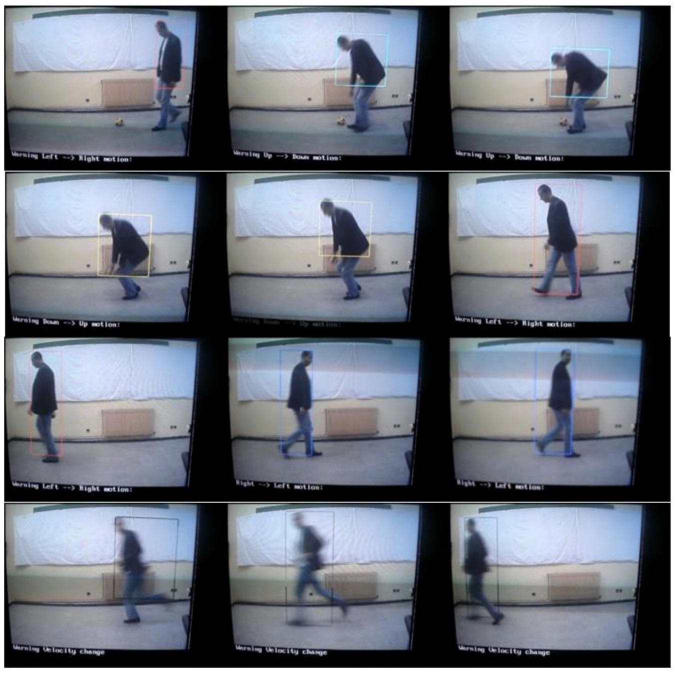


Fig. 13 Posture change detection: a) for one object, b) for two objects.

Figure 14 shows the results of collecting all the behaviors using a single program. Motion to the right and left are represented by red and blue rectangles, respectively. Further, up/down and down/up motion are represented by turquoise and yellow rectangles, respectively. Finally, velocity change is represented by a black rectangle. In every case, a warning message is displayed.


Warning Left --> Right motion!
Warning Up --> Down motion!
Warning Up --> Down motion!
Warning Down --> Up motion!
Warning Left --> Right motion!
Warning Left --> Right motion!
Right --> Left motion!
Right --> Left motion!
Warning Velocity change
Warning Velocity change
Warning Velocity change
Fig. 14 Motion analysis.

VI. Conclusions

We presented in this paper an implementation approach for object detection and behavior recognition based on motion analysis and sudden movements. We exploited the hardware part, which offers the possibility of handling large amounts of data and performing calculations for image processing via parallel processing, guaranteed by the use of the PixelStreams library of Agility's DK Design Suite. Further, we tried to improve our architecture by collecting all the different behaviors using a single program. In addition, we added warning messages using the PxsConsole filter. Thus, we successfully implemented different algorithms that can recognize objects in motion and detect changes in velocity, direction, and posture in real time. The results showed that our approach achieves good recognition and detection of these behaviors, in indoor areas. However, in outdoor areas, the results are less promising owing to the simple motion detection algorithm used; this problem is aggravated by occlusion due to overlapping movements of different persons. Therefore, in the future, we will try to use improved motion detection algorithm and learning methods to detect behavior changes in crowded environments, using a newer architecture.

Acknowledgment

We would like to thank Pr. Larbes Cherif and Dr. Benkouider Fatiha for their insightful comments.

REFERENCES

[1] L.Wang, W. Hu, and T. Tan, “Recent developments in human motion analysis”, Pattern Recogn, 36 (3), 585-601, (2003).

[2] W. Hu, T. Tan, L. Wang, and S. Maybank, “A survey on visual surveillance of object motion and behaviors”, IEEE T Syst Man Cyb, 34 (3), (2004).

[3] T. Ko, “A survey on behavior analysis in video surveillance for homeland security applications”, AIPR 2008, Washington, DC, USA, 2008.

[4] M. Piccardi “Background subtraction techniques: a review”, IEEE SMC, 4, 3099-3104, (2004).

[5] G.G.S. Menezes and A.G. Silva-Filho, “Motion detection of vehicles based on FPGA”, SPL VI Southern, 151-154, (2010).

[6] W.Shuigen, C.Zhen, L.Ming, and Z.Liang, “An improved method of motion detection based on temporal difference”, ISA 2009, 1-4, (2009).

[7] Widyawan, M.I. Zul, and L.E. Nugroho, “Adaptive motion detection algorithm using frame differences and dynamic template matching method”, URAI 2012, 236-239, (2012).

[8] I. Ishii, T. Taniguchi, K. Yamamoto, and T. Takaki, “1000 fps real-time optical flow detection system”, Proc. SPIE 7538, 75380M (2010).

[9] J. Diaz, E. Ros, F. Pelayo, E.M. Ortigosa, and S. Mota, “FPGA-based real-time optical-flow system”, IEEE T Circ Syst Vid, 16, (2), 274-279, (2006).

[10] M. Paul, S. Haque, and S. Chakraborty, “Human detection in surveillance videos and its applications - a review”, EURASIP JASP, Springer International Publishing, (2013).

[11] A.J. Lipton, H. Fujiyoshi, and R.S. Patil, “Moving target classification and tracking from real-time video”, WACV 98, 8-14, (1998).

[12] M. Ekinci and E. Gedikli, “Silhouette based human motion detection and analysis for real-time automated video surveillance”, Turk. J. Elec. Eng. & Comp. Sci., 13, 199-229 (2005).

[13] Y. Ran, I. Weiss, Q. Zheng, and L. S. Davis, “Pedestrian detection via periodic motion analysis”, Int J Comput Vision, 71 (2), 143-160, (2007).

[14] K. Ratnayake and A. Amer, “An FPGA-based implementation of spatio-temporal object segmentation”, Proc. ICIP, 3265-3268, (2006).

[15] M.Gorgon, P.Pawlik, M. Jablonski, and J. Przybylo, “FPGA-based road traffic videodetector”, DSD 2007.

[16] F. Cupillard , A. Avanzi , F. Bremond, and M. Thonnat, “Video understanding for metro surveillance”, ICNSC 2004.

[17] I. Haritaoglu, D. Harwood, and L. S. Davis, “W4: Real-time surveillance of people and their activities”, IEEE T Pattern Anal, 22 (8), 809-830 (2000).

[18] F. Bremond and G. Medioni, “Scenario recognition in airborne video imagery”, IUW 1998, 211-216, (1998).

[19] “DK5 Handel-C language reference manual”, Agility 2007.

[20] “PixelStreams Manual”, Mentor Graphics Agility (2015), http://www.mentor.com/products/fpga/handel-c/pixelstreams/

[21] “Agility DK User Manual”, Mentor Graphics Agility (2015), http://www.mentor.com/products/fpga/handel-c/dk-design-suite/

[22] “Virtex II 1.5v Field-Programmable Gate Arrays”, Data sheet, Xilinx Corporation, 2001.

[23] K. Sehairi, C. Benbouchama, and F. Chouireb, "Real Time Implementation on FPGA of Moving Objects Detection and Classification," International Journal of Circuits, Systems and Signal Processing, 9, 160-167, 2015.