Design Methodology for Face Detection Acceleration

Laurentiu Acasandrei
Instituto de Microelectronica de Sevilla
IMSE-CNM-CSIC
Sevilla, Spain
laurentiu@imse-cnm.csic.es

Angel Barriga
Instituto de Microelectronica de Sevilla/Univ. Sevilla
IMSE-CNM-CSIC/ Univ. Sevilla
Sevilla, Spain
barriga@imse-cnm.csic.es

***Abstract*—A design methodology to accelerate the face detection for embedded systems is described, starting from high level (algorithm optimization) and ending with low level (software and hardware codesign) by addressing the issues and the design decisions made at each level based on the performance measurements and system limitations. The implemented embedded face detection system consumes very little power compared with the traditional PC software implementations while maintaining the same detection accuracy. The proposed face detection acceleration methodology is suitable for real time applications.**

Keywords—face detection; embedded system; design methodology; hardware & software codesign

I. INTRODUCTION

Face detection is an important aspect for biometrics[1], video surveillance and human computer[2] interaction. Detection systems require huge computational and memory resources due to the complexity of detection algorithms. A software detection realization implemented on a low speed, low resource, low power SoC (System on a Chip) it is not efficient. Instead a Software-Hardware Codesign approach can be used to build hardware accelerators for most computational consuming parts of detection algorithms.

Real time face detection application requires a high amount of multipliers and memory resources running at high speed. Due to high amount of resources and high speed requirements a software implementation of the face detection algorithm is not feasible for a low speed, low resource, low power SoC. Recently there have been several hardware realizations that accelerate the face detection process by parallelizing the detection algorithm. Thus some of the proposals are for specific hardware realizations on ASIC [3] while others are implemented on FPGA [4-6]. Using the Software-Hardware Codesign approach we can accelerate the detection process by replacing, with very fast dedicated hardware, some parts of the software that usually consumes large amount of clock cycles and/or memory during the run time. In [7] several parts of the Viola-Jones detection mechanism have been parallelized in order to run efficiently on a GPU architecture.

The purpose of this communication is to describe a design methodology for accelerating face detection applications. Taking into consideration that the OpenCV library comes with two baseline applications LBP (Local Binary Pattern) and Viola-Jones for video detection, we have chosen Viola-Jones for developing a face detection application for embedded systems. The same design methodology is common to both algorithms. The LBP and Viola-Jones have the same detection mechanism, the only difference being in how to encode/decode the features from an image.

II. FACE DETECTION TECHNIQUE

The face detection technique is based on the face detection framework proposed by Viola-Jones [8]. The proposed framework is capable of processing images extremely rapidly while achieving high detection rates. The speed of the face detection framework relies on three important key components. Firstly, the image is transformed into "Integral Image" which allows the features used by the detector to be computed very quickly. Secondly, the used classifier is simple and efficient which is build using the AdaBoost learning algorithm [9] to select a small number of critical visual features from a very large set of potential features. And thirdly, the classifier is formed by combining weak classifiers in a "cascade" which allows background regions of the image to be quickly discarded while spending more computation on promising face-like regions.

Viola-Jones technique is based on exploring the image by means of a window looking for features. This window is scaled to find faces of different sizes. The system architecture is based on a cascade of detectors. The first stages consist of simple detectors, very fast and low cost, that allows to eliminate those windows that do not contain faces. In the successive stages the complexity of detectors are increased in order to make a more detailed analysis of features. A face is detected only if it makes it through the entire cascade.

The Haar-like features used by the classifier consist of rectangular areas whose processing requires simple arithmetical operations. The calculation is based on the sum of the pixels of each rectangular region weighed by a weight. At all scales, these features form the "raw material" that will be used by the detector. The set of rectangle features in the image is quite large and overcomplete, so to reduce that number applies the AdaBoost learning algorithm [9]. The Viola-Jones classifier employs AdaBoost at each node in the cascade to learn a high detection rate at the cost of low rejection rate multitree (mostly multistump) classifier at each node of the cascade.

To facilitate the processing of the features the operations are not made on the original image but on an integral image. Therefore the detection algorithm requires a preprocessing step

that calculates this integral image. The integration of the image consists of adding for each pixel the values of the previous pixels.

III. DESIGN METHODOLOGY

The starting point of the design methodology of the embedded system is the OpenCV's Viola-Jones baseline face detection application. OpenCV (Open Source Computer Vision), started by Intel in 1999, is a library of programming functions for real time computer vision [10]. OpenCV is released under a BSD license and hence it is free for both academic and commercial use. It is written in C/C++ and was designed for computational efficiency and with a strong focus on real-time applications.

The host target for the proposed face detection system is an embedded environment based on LEON3 AMBA Bus processor. The LEON3 is a synthesizable VHDL soft core of a 32-bit processor compliant with the SPARC V8 architecture [11]. The processor is highly configurable, and particularly suitable for system-on-a-chip (SOC) designs. The full source code is available under the GNU GPL license. The processor controls and executes the majority of software application tasks while a specific IP (Intellectual Property) module accelerate only those tasks that require a high number of clock cycles.

The design flow is based on four stages, as shown in Figure 1. In the first stage an adaptation of the software application to execute on the embedded system has been made. In the next stage an analysis of the new embedded software application is performed, in order to detect "bottlenecks" and those tasks that are suitable to accelerate through hardware implementation. In the third phase, as result of the previous analysis, solutions to optimize and accelerate some of the tasks of the face detection process will be proposed. The fourth stage consists in the hardware implementation of those parts that consume a many resources and have large run times. The idea is to offload to the hardware the functions with a high degree of processing and to parallelize the execution of the detection algorithm. With this the face detection process can be drastically accelerated.

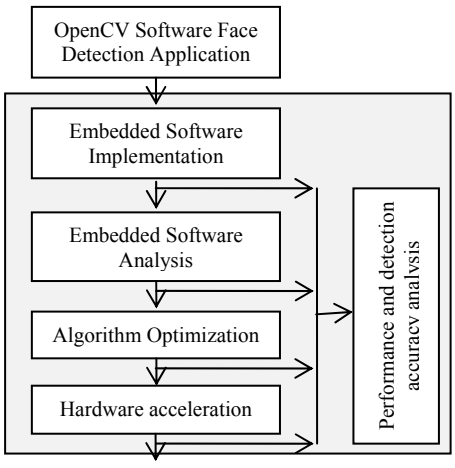


Fig. 1. Design methodology

At each stage of the design process the performance of the face detection system is analyzed, in terms of speed and detection accuracy. The accuracy of face detection process is analyzed using ROC (Receiver Operating Characteristic) curves. The ROC curve of a given face detector shows its performance as a trade-off between the false acceptance rate and the face detection rate by varying its discrimination criterion (e.g. a threshold parameter).

IV. EMBEDDED SOFTWARE FACE DETECTION IMPLEMENTATION

Viola and Jones organized each boosted classifier group into nodes of a rejection cascade. Each of the nodes contains an entire boosted cascade of groups of decision stumps (or trees) trained on the Haar-like features from faces and nonfaces (or other objects the user has chosen to train on). Typically, the nodes are ordered from least to most complex so that computations are minimized (simple nodes are tried first) when rejecting easy regions of the image. Typically, the boosting in each node is tuned to have a very high detection rate (at the usual cost of many false positives). When training on faces, for example, almost all (99.9%) of the faces are found [12] but many (about 50%) of the nonfaces are erroneously "detected" at each node. But this is satisfactory because using, as an example, 20 nodes will still yield a face detection rate (through the whole cascade) of $0.999^{20} \approx 98\%$ with a false positive rate of only $0.5^{20} \approx 0.000001\%$!

During algorithm execution, a search window of different sizes is swept over the original image. In practice, 70–80% of nonfaces are rejected in the first two nodes of the rejection cascade, where each node uses about ten decision stumps. This quick and early "attentional reject" vastly speeds up face detection.

The Haar-like features are trained to be applied for a evaluating rectangular window of 20x20 pixel. For other dimensions of the evaluating window the Haar-like features must be scaled correspondingly. The OpenCV software implementation for face detection consists of 22 cascade detectors, containing 2135 Haar like features.

The first task of Software-Hardware Codesign was adapting and optimizing the OpenCV baseline application for an embedded environment. We have considered that the majority of embedded environments are capable of running C/C++ applications with or without Operating System (OS) support. This means that the resulting application code has to be compatible for both C, C++ compilers and in the same time platform independent.

Another consideration made is the fact that most of the SoC have no floating point support. For it, the resulting application uses integer operations instead of floating point operations in order to preserve the generality of the application for the embedded system world. An important moment in this step was finding an acceptable scaling coefficient of the floating point variables and data to integer variables and data. After trying different values and comparing the resulted integer application with the floating point application we found that by scaling with 20 bits (precision of 20 bits for the floating point decimals) the integer and floating point applications obtain

identical results. Also the floating point squared root function necessary to calculate variance of the evaluating window was replaced with a fast integer squared root version.

In the end it was obtained a face detection stand-alone application compatible with C/C++, using only integer type operations and data.

V. Embedded Software Face Detection Analysis

The next step in application development was analyzing different modes of detection in order to find the run time bottlenecks and optimize the detection. For the embedded target we have decided to use the detection mode where the detector (Haar-like feature) is scaled and the biggest regions containing faces are searched within an image. In this mode the detections starts with the biggest evaluation window and biggest evaluation step and progressively, the window together with the evaluation step are decreased until a region containing a face is detected. In the case that a region with face or multiple faces is detected, the attention of the detector concentrates in that region.

The trained classifier cascade (Haar-like feature) is provided by OpenCV in an XML file format. Using this XML format in an embedded system will produce memory and run time overhead. In order to avoid the overhead we have developed an application that receives a XML file, interprets the data and saves it in a simpler format to a C header file. The resulted embedded application can be compiled with the cascade of classifier or the data can be transmitted during the execution of the application via an appropriate interface.

In order to obtain relevant insight about which parts (or functions) of the face detection program are taking most of the execution time we enable *-gp* flag in the Eclipse project compilation options in order to generate profiling information that can be interpreted with GNU *gprof* tool. This tool permits one to learn where the program spends its time and the function calling tree during the execution. It can also tell which functions are being called more or less than are expected.

Table I shows the obtained results. As we can see the function SetMatZero() uses 24.64% of the executing time even surpassing the time spent applying the cascade Haar like-features (20.16%) for the entire image. The function SetMatZero() is used to set to zero all the elements of a temporary matrix having the same dimension of the image. In this matrix the top left coordinates of a detected face are flagged with value 1. In this mode the detections starts with the biggest evaluation window and biggest step and progressively, the window and the step are decreased until a region containing a face is detected. The detection is done in two steps:

- First Step: The image is scanned with the evaluation window by applying only the first two Haar-like feature stages in order to rapidly detect regions containing potential faces. If a region is found to have a potential face then the coordinates are set to value one in the temporary matrix.
- Second step: Each potential face (starting with their coordinates) from the temporary matrix is evaluated with the remaining Haar-like feature stages. If a true face is detected then the coordinates, width and height are stored in a list. At the end of the second step, the temporary matrix is set to zero in preparation for the next image where the evaluating window has smaller dimensions.

We can improve the speed by not using the function SetMatZero() at the end of the second step and instead each time after we have detected a face during the second step and that face is stored into a list, we set to zero the coordinates inside the temporary matrix. After applying this change, the detection time is improved with 24.64 %.

TABLE I. Analysis of Detection System Bottlenecks

Time %	# calls	Function name
24.64	17	SetMatZero()
20.16	3201	RunHaarClassifierCascadeEmbedd()
14.81	16	SetImagesForHaarClassifierCascadeEmbedd()
13.39	1	Integral()
11.35	1	LinkDataToEmbeddClassifierCascade()
4.22	511	HResizeLinear()
3.11	262144	saturate_uchar()
2.62	512	VResizeLinear()
1.80	296384	sum_elem_ptr()
1.10	3201	isqrt64()

VI. Embedded Face Detection Optimization

The OpenCV face detection baseline application implements detection in two distinct modes:

Mode 1: Face detection by scaling the image. In this mode the image is scaled using interpolation until it reaches a predefined minimal dimension. Each time the image is scaled the two integral images (normal=$\sum x$ and squared=$\sum x^2$), needed for variance, are recalculated for the scaled image. The search window has fixed dimension during the detection process.

Mode 2: Face detection by scaling the classifiers. In this mode the integral images(normal=$\sum x$ and squared=$\sum x^2$), needed for variance, are calculated only once for the original image but the Haar-Like features form the classifier are scaled progressively until their dimensions are close to the dimension of the original window. This mode lacks the interpolator used in mode 1. The search window has a variable dimension during detection process.

In both detection mode the Haar-like features components (weights and dimensions) are scaled proportionally with the dimensions of search window. That means for a search window of dimension *WxH*, the weight of each rectangle forming the Haar-like features are scaled with *WxH*. In the search window, the sum of each applied Haar-like feature is calculated using:

$$HaarFeature^{Sum} = \sum_{I=1}^{3} Area_I \cdot Weight_I^{scaled} \quad (1)$$

Area represents the sum of all pixels inside a component and *I*=1, 2 or 3 represents the number of components for that Haar-like feature. In order to determine the next weight value

for the stage sum, each $HaarFeature^{Sum}$ is compared with each normalized threshold of the respective Haar-like feature as:

$$StageSum = \begin{cases} StageSum + Stage_J^{Weight2}, if\ HaarFeature_J^{Sum} > Thres_J^{norm} \\ StageSum + Stage_J^{Weight1}, if\ HaarFeature_J^{Sum} < Thres_J^{norm} \end{cases} \quad (2)$$

where J=[1…2135] represents the Haar-like feature indexes in a stage and $Thres_J^{norm}=\sigma Thres_J^{HaarFeature}$ (σ is standard deviation of the windows search area).

If we do not scale the Haar-like feature weights and adjust the variance computation by using the formula $\sigma^2_{adjusted} = W \cdot H \cdot (\sum x^2 - (\sum x)^2)$, it results that the number of arithmetic operations (division, multiplication) and memory accesses are decreased substantially. This will make the algorithm perform faster due to a reduced number of operations needed for computation of the adjusted variance for the search window [13]. Figure 2 shows the proposed optimization of the detection algorithms in the two detection modes (scaling the image and scaling the classifiers).

Fig. 2. Proposed face detection acceleration algorithm

In order to compare the performance of the OpenCV 2.2 baseline face detection and the accelerated Viola-Jones algorithm, and to analyze the influence of the configuration parameters, both implementations have been compiled and speed optimized for 64 bit Win7 OS using Visual Studio 2010 Profesional edition. The verification PC has a Pentium Dual-Core CPU T4300, with L1 cache of 128KB, L2 unified cache 1024KB and the OS is Windows 7 Home Edition 64 bits.

Both implementation (OpenCV's implementation and accelerated Viola-Jones version) have received the same test VGA (640x480) images and the same configuration parameters. In Table II the configuration have different scale factor (sf) and minimum search window dimensions (swd): sf=1.1 and swd=20x20 (Conf 1); sf=1.2 and swd=20x20 (Conf 2); sf=1.1 and swd=30x30 (Conf 3).

TABLE II. PERFORMANCES OF OPENCV AND ACCELERATED VIOLA-JONES IMPLEMENTATION FOR DIFFERENT PARAMETERS

		Mode 1. Img scaling		Mode 2. Haar scaling	
		OpenCV Baseline	Optimized version	OpenCV Baseline	Optimized version
Conf 1	speed	708.8 ms 1.41 FPS	185.7ms 5.38FPS	843.9 ms 1.18FPS	188.1ms 5.3FPS
	Search windows	602348	599816	697582	631343
Conf 2	speed	409.5 ms 2.44FPS	164.8ms 6.06FPS	479.7ms 2.08FPS	140.1ms 7.1FPS
	Search windows	354321	353935	381474	364635
Conf 3	speed	348.1ms 2.87FPS	99.4ms 10FPS	456.7 ms 2.18FPS	130.6ms 7.6FPS
	Search windows	352718	351184	402519	332917

As the proposed implementation has kept the control mechanism for search windows identical with the one form the OpenCV baseline it is difficult to make a comparison with previous work done in accelerating detection due to the lack of setup information and how many search windows are evaluated. There is one exception Cho et al [6] where they use mode1 with a scaling factor of 1.2, minimal search window dimension of 20x20 and the search window is applied with a vertical horizontal step of 1. Table III shows the comparison results.

TABLE III. PERFORMANCE OF THE OPENCV BASELINE AND ACCELERATED VIOLA-JONES USING THE CHO ET AL.[6] SETUP

Implementation	Mode1-Image scaling	
	Speed	Search Windows
OpenCV	974.3 ms 1.02FPS	881484
Accelerate Viola-Jones	451.4 ms 2.21 FPS	880585

As shown in Tables II and III the number of search windows depends heavily on the configuration setup and also of the control mechanism. Also measuring the number of searched windows performed by the detection system gives more realistic information about the detection performance. The speeds obtained by the accelerated Viola-Jones implementation in some configuration are faster that any single GPU acceleration [3], [7] and for the highest number of search windows (Table III) it has closer performances to [3] and [7].

VII. HARDWARE ACCELERATION OF THE EMBEDDED FACE DETECTION SYSTEM

After a careful analysis of the face detection application it was found that the software bottleneck resided in the huge amount of memory read access, multiplication, and squared root operations done by all the search window evaluations. In order to detect a face from an image, hundreds of thousands of search windows are evaluated and this represents the most time consuming part of the application. Therefore it was decided to accelerate the evaluation of search windows by employing a harware IP module [14]. The proposed IP module and all internal components are clocked by the system clock (80 Mhz).

In order to keep a high degree of flexibly and share the hardware resources with the rest of the LEON3 system it was decided for the *IMSE_OBJECT_DETECTION* IP to have two operating modes: the free mode in which LEON3 processor can use the IP hardware resources to implement other functionalities, and the face detection mode. Figure 3 shows the block diagram of the IP module.

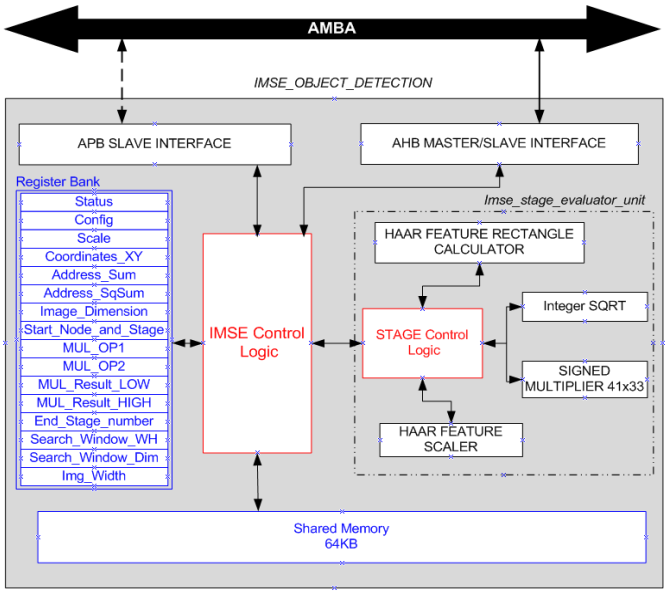


Fig. 3. IMSE_OBJECT_DETECTION IP block diagram

As it was previously mentioned the IP module implements the search window algorithm. The software application will load the compressed form of Haar-like features into the Shared Memory before any detection operation. Before starting any detection operation the configuration registers (scale, x-y coordinates, image dimension, etc.) must be configured with the desired configuration values. When the start command is given the face detection procedure is fully controlled by the component *Imse_stage_evaluator_unit* (see Figure 3). This unit is the core engine for accelerating face detection. At the end of the detection the component signalize if a face is present, the *Status* register is updated with the detection result and an interrupt is generated.

The evaluator unit has a multiple state machine control in order to deal with variable memory access latencies. Beside the multiple state machine control this unit contains specialized modules. *Haar_feature_rect_calc* module is used to calculate the area of integral rectangles using only the corners data. *Haar_feature_scaler* is a pipelined module for Haar-like feature scaling and search window address computation. *Sqrt64_array_pipe16* is 64 bit pipelined integer square root unit that has data output latency of 16 clocks. *Mul41x33signed* is a 41x33 signed multiplier. The *Register Bank* contains APB bus accessible registers that are used for the core configuration and control. The *APB Slave Interface* connects the IP module to the APB bus and enables the LEON3 processor to access the registers from Register Bank. The *AHB Master/Slave Interface* is a simple DMA interface.

The IP module has a shared memory based on a dual-port RAM with AHB interface. The *Shared Memory* is used by the IP module to store the compressed Haar-like features. When the module works in "Free Mode" LEON3 can use the Shared Memory as additional on chip RAM memory.

VIII. RESULTS

The proposed LEON3 face detection system works with images (colored or grey) having a resolution smaller than 1024x1024 pixels. It fully uses the OpenCV cascade of classifiers for frontal faces and it can store into the IP Shared Memory approximately 2730 Haar-like classifiers. It also works with a greater number of Haar-like classifiers but the extra classifiers must be store into the program memory and then loaded into the Shared Memory at the appropriate moment.

The system was implemented on a Xilinx XC5VLX50 FPGA. The entire LEON3 face detection system uses 6,435 slices (up to 89% of the device utilization) and 10,962 of flip-flops (up to 38% of the device utilization). The estimated static power consumption (measured with Xpower Analyzer from Xilinx) for the LEON3 core is 603 mW. The most power consuming components are the DDR2 memory controller (216 mW), DVI interface (136.06 mW) and the clock generators. The LEON3 processor consumes 32.39 mW and even though the IMSE_OBJECT_DETECTION IP uses more flip flops and has approximately the same amount of logic, its power consumption is 6 times less (5.39 mW) than the LEON3 processor.

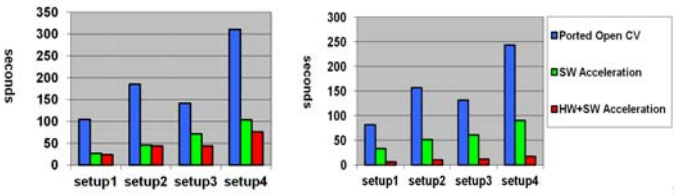
A. Performance

In order to measure the detection performances of the LEON3 embedded detection system three distinct software implementations for face detection were compared:

- Ported OpenCV software for embedded systems.
- Software accelerated version of the ported software.
- Hardware + Software accelerated version of the ported software.

The measured performances metrics are the execution time and the number of searched windows performed. For the first two implementations the performances were measured for two distinct modes of detection (mode 1 and mode 2). For each mode, four different set-up parameters were used (setup 1 to 4) for the minimum size search window (S) and the scale step (step): 1) S=30x30, step=1.2; 2) S=30x30, step=1.1; 3) S=20x20, step=1.2; 4) S=20x20, step=1.1.

From Figure 4 it results that the accelerated face detection application is 3-4 times faster than the ported OpenCV application for both modes. Using the hardware acceleration IP the face detection is 10-12 times faster than the ported face detection application running exclusively on LEON3 processor core.



a) b)

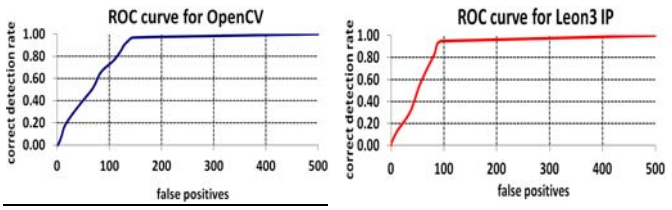
Fig. 4. Detection times of three distinct implementations for VGA image. a) Scale Image mode (Mode 1), b) Scale Haar-like features mode (Mode 2)

B. Detection Accuracy

In order to analyze the detection accuracy an specific software has been developed. The PC based test bench software configures LEON3 based detection system and send the test images. Then it receives the detection results for further analysis. The test setup is based on 2409 frontal face images from the color FERET database [15].

The accuracy of face detection can be described using receiver operating characteristic (ROC), which is a curve widely adopted in signal-detection theory. An ROC is essentially a scatterplot that shows the relationship between the false acceptance rate and the true acceptance rate. Because the number of evaluated negative regions is very high for detection algorithms, for analysis, it was adopted a modified version of ROC curve introduced in FDDB[16] in which the horizontal axis contains only false positive for the entire test image set.

Figure 5 shows the ROC curves for OpenCV software and the IP module. Both have very similar results. There is a small difference between the two ROC curves because the data result (i.e detected faces) aggregation mechanism is slightly different in the analysis tools.



(a) (b)

Fig. 5. ROC curves: a) OpenCV face detection software, b) IMSE_OBJECT_DETECTION IP based system

IX. CONCLUSIONS

This communication presents a design methodology for face detection embedded implementation. The starting point was OpenCV library resources for video face detection. For it some modifications has been make adapting and optimizing the OpenCV baseline application for an embedded environment. This modifications can be summarized in changing the floating point operations by integer ones, analyzing the performance in order to detect the system bottlenecks, and algorithmic speed-up by not scaling the Haar-like feature weights and adjusting the computation of variance. Finally, those tasks requiring a greater computation are implemented in hardware, accelerating the face detection process, and enabling its application in embedded systems that require real time.

ACKNOWLEDGEMENT

This work was supported in part by Spanish Ministerio de Ciencia y Tecnología under the Project TEC2011-24319, and by Junta de Andalucía under the Project P08-TIC-03674. Co-financed by FEDER.

REFERENCES

[1] K. Suzuki, J. Kobayashi, T. Takeshima; K. Yamada, "Detection of unusual facial expression for human support systems," 34th Annual Conference of IEEE Industrial Electronics. IECON 2008, pp.3414,3418, 10-13 Nov. 2008.

[2] M. Paschero, G. Del Vescovo, L. Benucci, A. Rizzi, M. Santello, G. Fabbri, F.M.F. Mascioli, "A real time classifier for emotion and stress recognition in a vehicle driver," IEEE International Symposium on Industrial Electronics (ISIE), pp.1690,1695, 28-31 May 2012.

[3] T. Teocharides, N. Vijaykrishnam, M.J. Irwin, "A Parallel Architecture for Hardware Face Detection", Proc. IEEE Computer Society Annual Symposium Emerging VLSI Technologies and Architectures, pp. 452-453, 2006.

[4] V. Nair, P.O. Laprise, J.J. Clark, "An FPGA-Based People Detection System", EURASIP Journal on Applied Signal Processing, pp. 1047-1061, 2005.

[5] C. Gao, S.L. Lu, "Novel FPGA Based Haar Classifier Face Detection Algorithm Acceleration", Proc. International Conference on Field Programmable Logic and Applications, 2008.

[6] J.Cho, S. Mirzaei, J. Oberg, R. Kastner, "FPGA-Based Face Detection System Using Haar Classifiers", Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'09), pp. 103-112, 2009.

[7] D. Hefenbrock, J. Oberg, N.T.N. Thanh, R. Kastner, S.B. Baden, "Accelerating Viola-Jones Face Detection to FPGA-Level using GPUs", Proc. IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, 2010.

[8] P. Viola, M.J. Jones, "Robust Real-Time Face Detection", International Journal of Computer Vision, v.57 n.2, pp.137-154, May 2004.

[9] R.E. Schapire, Y. Freund, P. Bartlett, W.S. Lee, "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods", The Annals of Statistics, pp. 1651-1686, 1998.

[10] OpenCV, link: http://sourceforge.net/projects/opencvlibrary/

[11] LEON3, link: http:// www.gaisler.com/

[12] G. Bradski, A. Kaehler, "Learning OpenCV", O'Reilly Media, pp.506-516, September 2008.

[13] L. Acasandrei, A. Barriga: "Accelerating Viola-Jones Face Detection for Embedded and SoC Environments", Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'2011), Ghent, Belgium, Aug. 2011.

[14] L. Acasandrei and A. Barriga: 'FPGA implementation of an embedded face detection system based on LEON3', Int. Conf. on Image Processing, Computer Vision, and Pattern Recognition, Jul. 2012.

[15] P.J. Phillips, H. Wechsler, J. Huang, P. Rauss: "The FERET database and evaluation procedure for face recognition algorithms," Image and Vision Computing J, Vol. 16, No. 5, pp. 295-306, 1998.

[16] J. Vidit and E.L. Miller: "FDDB: A Benchmark for Face Detection in Unconstrained Settings", Technical Report UM-CS-2010-009, Dept. of Computer Science, University of Massachusetts, 2010.