14th European Signal Processing Conference (EUSIPCO 2006), Florence, Italy, September 4-8, 2006, copyright by EURASIP

EMBEDDED IMAGE PROCESSING/COMPRESSION FOR HIGH-SPEED CMOS SENSOR

R.Mosqueron, J.Dubois and M.Paindavoine

Laboratoire Le2i - UMR CNRS 5158, Universite de Bourgogne
Aile des Sciences de l'Ingenieur, BP 47870, 21078 Dijon cedex, France
phone: + (33) 380393606, fax: + (33) 380395910, email: romuald.mosqueron@u-bourgogne.fr
web: www.le2i.com

ABSTRACT

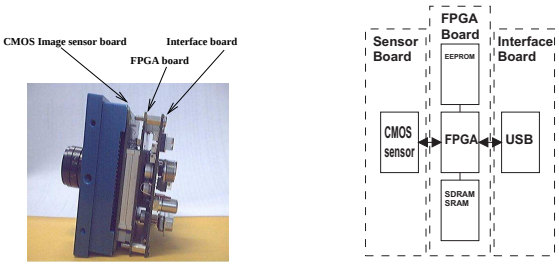
High-speed video cameras are powerful tools for investigating for instance the biomechanics analysis or the movements of mechanical parts in manufacturing processes. In the past years, the use of CMOS sensors instead of CCDs has made possible the development of high-speed video cameras offering digital outputs, readout flexibility and lower manufacturing costs. In this paper, we proposed a high-speed camera based on CMOS sensor with embedded processing. Two types algorithms have been implemented. The compression algorithm represents the first class for our camera and allows to transfer images using serial output link. The second type is dedicated to feature extraction like edge detection, markers extraction, or image analysis, wavelet analysis and object tracking. These image processing algorithms have been implemented into a FPGA embedded inside the camera. This FPGA technology allows us to process in real time 500 images per second with a $1,280H \times 1,024V$ resolution.
Keywords: CMOS Image Sensor, FPGA, Image Compression, High-speed Video.

1. INTRODUCTION

In the past years, the use of CMOS sensors instead of CCDs [1, 2]has made possible the development of industrial high-speed video cameras offering digital outputs, readout flexibility and lower manufacturing costs. Two main limitations of these systems may be discussed. First, the huge data flow provided by the sensor cannot be easily transferred or processed and has generally to be stored temporarily in a local fast RAM memory. This RAM is size limited so the recording time in the camera is only a few seconds long. Using an image compression approach, we developed an alternative solution that allows continuous recording. With this approach, we do not use an embedded RAM memory, and store directly to a connected PC memory. A major advantage is to use the permanent evolution of PC RAM memory.
Second, in order to execute in real-time image processing dedicated to tracking or object recognition, the interesting information must be extracted from huge data flow inside of the FPGA. The image processing algorithms have been implemented inside a FPGA and with technology, we have show that it is possible to process in real-time 500 frames per second (resolution $1,280H \times 1,024V$). In particular, to reach this speed, we implemented on FPGA, fast parallel algorithms dedicated to image compression and image segmentation. This paper is organized as follow. Our high-speed camera is described in Section 2. The studied image compression and image processing algorithms and their implementations inside the FPGA are introduced in Section 3. Finally conclusions and perspectives are drawn in Section 4.

2. HIGH-SPEED CAMERA DESCRIPTION

Nowadays, many image CMOS sensors exist providing a high acquisition flexibility. Moreover, some of them enable to access simultaneously to several pixels at high frequency (up to 66MHz) therefore the input data bandwidth is extremely high. This kind of sensor allows many possibilities in terms of high-speed acquisition and processing. Indeed, in order to process in real-time, simultaneously pixel access is required. For these reasons, we have designed a high-speed camera based CMOS sensor and in order to control this data flow and to process in real-time these informations, a specific unit has been designed based on FPGA component. FPGA features enable to connect a high number of I/O, to design a specific controller adapted to the sensor and finally to achieve real-time processing on large input data flow. Low level image processing are regular, frequently same task can be done simultaneously on several pixels or different regions of the image. The FPGA architecture and hardware ressources are specially adapted to these operations therefore processing time can be significantly reduce and data flow bottleneck remove. In the literature, almost high-speed cameras are developed with embedded memory (optronis CAMRECORD600 [3], motion Blitz cube Eco2 Mikrotron [4]and others) or with parallel output (camera link) but the host PC need a frame grabber (VDS CMC1300 [5] or Basler A504 [6]). To obtain long sequences, the memory is very large and therefore very expensive. We propose a new solution based on the suppression of the embedded memory and direct data transfer into the external memory on the host PC connected to the camera output. By using PC memory, the camera benefits of evolution in terms of size and frequency. The camera output has to be the most simple as possible with low cost like a standard serial interface like firewire or USB.
We used the MT9M413 high-speed CMOS image sensor from Micron in order to design our high-speed camera. The main features of this image sensor are a high frame rate (500 images per second at full size frame ($1,280H \times 1,024V$)), the output with 10-bit digital through 10 parallel ports, a high output data rate at 660Mbs,the shutter exposure time with a minimum of 100ns. In relation to the image data rate and image resolution we have selected the VIRTEX-II XC2V3000 FPGA from Xilinx with 14,336 logical blocks (slices), 96 dedicated 18-bit$\times$ 18-bit multiplier blocks and 18 Kbit dual-port RAM (BRAM) and 720 I/O pads. USB2.0(Universal Serial Bus version 2.0) is a interesting solution for our digital camera because it provides true plug and play installation and it is an hot-plugging material. Our high-speed camera



Internal View Synoptic

Figure 1: High-speed camera system.

system is constituted by three boards as shown in Figure 1, which represents respectively the acquisition board with only CMOS sensor, the acquisition control and processing based on FPGA implementation and finally interfaces board. As any standard CMOS camera, user can define image number and select random region acquisitions with variable region size(Region Of Interest(ROI)). In full resolution, with USB2.0, 17 frames per second can be transferred and simultaneously displayed on the VGA screen.

3. ALGORITHMS AND IMPLEMENTATION

In the context of our fast CMOS camera design, we have distinguished two different types of high-speed image applications. The first class regroups applications that do not require real-time operations, for instance off-line image processing or a visualization of recorded sequences that represent a high-speed phenomenon (Section3.1). The second class regroups applications that require on-line operations like high-speed feature measurements (motion, boundaries, marker extraction)(Section3.2). Therefore, in this context, most of the time, the camera output flow is considerably reduced. With our camera design, FPGA embedded solutions are proposed for the two presented classes. In order to match to requirements of first class applications, an embedded compression is proposed. With this compression, full resolution images can be transferred up to 500 frames per second. To demonstrate on-line capacities of our camera, feature extraction processing has been implemented. As the previous class, the measurement is performed at the highest frequency of sensor data output. In any case, both solutions must deal with the main feature of high-speed imaging : the important sensor output data bandwidth (in our case up to 660 Mpixels per second). Hence, the embedded solutions must achieved real time processing on this large input data flow moreover the hardware ressource should be minimized.
Consequently, some image processing, compression and feature extraction, has been implemented regards to performances and the hardware ressource available. These two different implementations are described in following sections.

3.1 Embedded compression

3.1.1 Analysis for embedded compression

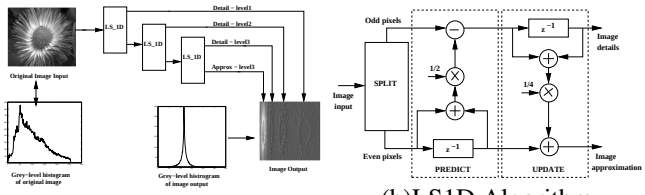
In this section, hardware implementation for image compression are discussed respect to high-speed imaging constraints. We will focuss on recent FPGA implementation. Three hardware implementations of image compression standard (JPEG, JPEG2000, MPEG4) are then presented. These methods are based on spatio-temporal algorithms and use different approaches like predictive coding, transform coding (Fourier transform, Discrete cosinus transform) or wavelet coding.
First at all, these implementations perform compression on video stream at high frequency (Table 1). The JPEG, JPEG2000 and MPEG4 IPs can process respectively 50, 13 and 12 MPixels per second [7, 8, 9]. Nevertheless these performances do not match with high-speed constraints(660 Mpixels per second = 66 MHz x 10 pixels). Our 10 pixel access at each cycle can be a solution to increase performances. Nevertheless, a parallel processing of the 10 pixels is not an easy task. Indeed, the spatio-temporal dependency do not permit to split data flow between several IPs in parallel without modifications on the IP. Once again, the modifications are not easy tasks on such complex IPs, moreover hardware ressource cost would be very high. Indeed, the three standard implementations are require respectively 3034, 10800, and 8300 slices with a serial pixel access. Even partial implementations, like partial JPEG2000 [10], can not match the input flow constraint. In this design entropy encoder has not been implemented, therefore the complexity is reduced to 2200 slices, nevertheless the processing frequency is still not sufficient (33 pixels/s). Moreover, nearly all these IPs require external memory. To solve this problem we propose a trade off between processing performances and hardware ressources by implementing a 1D Discrete Wavelet Transform. Therefore, no external memory is required, indeed a 1D transform can be applied directly on the input data flow. This original implementation permits to process at each cycle 10 pixels in parallel (1D10P-DWT).

Compression IP	input flow	slices /BRAM	freq Mpix/s	external memory
JPEG	S	3034/2	50	no
Part. JPEG2000	S	2200/0	33	yes
JPEG2000	S	10800/41	13	yes
MPEG4	S	8300/21	12	yes
1D10P-DWT +RLE	P	2465/9	130	no
1D10P-DWT +BC+Huff	P	3500/17	660	no

P=Parallel data flow S=Serial data flow
RLE=Run-Length Encoding BC=Block Coding
Huff=Huffman Encoding

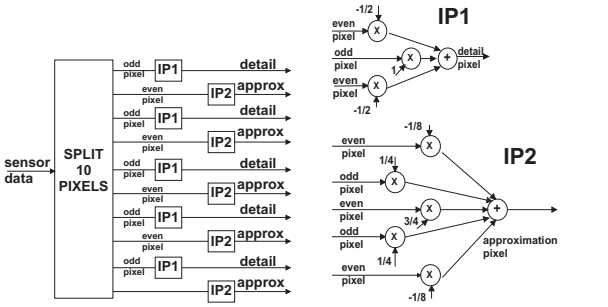
Table 1: Comparison of compression implementation.

The advantage of wavelet transform coding [11, 12] for image compression is that resulting wavelet coefficients decorrelate pixels in the image and thus can be coded more efficiently than the original pixels. Figure 2a is an illustration of a 1D wavelet transformation with 3 levels of decomposition. The original image histogram shows that grey-level distribution is relatively large (range from 0 to 255) while the wavelet coefficients histogram is thinner and centred on the zero value. Using this property, wavelet coefficients can be coded with better efficiency than the pixels in the original image. The 1D10P-DWT implementation and two associated compression are described in the next section. Nevertheless as a comparaison point with standard compression implementations, their performances and hardware requirements are reported into the table 1.



(a)Pyramidal Wavelet (b)LS1D Algorithm

Figure 2: Wavelet Pyramidal Algorithm



(a)Split 10 pixels (b)Update and Predict

Figure 3: Wavelet IPs

3.1.2 Wavelets pre-processing and compression

In order to implement a wavelet transform compatible with hardware constraints, we use the lifting-scheme approach proposed by Sweldens[13] in 1995. This wavelet transform implementation method is described in Figure 2b were we consider the original image pixels in a data-flow mode (in a 1D representation). The one dimension Lifting-scheme (LS_1D) approach is decomposed into three main blocks: Split, Predict and Update. The Split block separates pixels in two signals: odd pixels and even pixels. The Predict and Update blocks are simple digital first order FIR filters which produce two outputs: image details (wavelet coefficients) and image approximation used for the next LS_1D stage. For this camera, the data flow is 10 pixels width and the IPs that we designed is based on it (Figure 3a).

The CMOS image sensor 10 pixels simultaneously and therefore a real-time parallel processing is necessary. For this, the 10 pixels are split in five odd pixels and five even pixels (Figure 3a). For the odd pixel we designed the IP1 and for the even the IP2. This 2 IPs are based on the same principle of LS_1D [14, 15]. For the IP1 the central pixel is the odd pixel and we use the 2 neighbor even pixel with the appropriate coefficients. For the IP2 the central pixel is the even pixel and we use the 2 neighbor odd pixel and the 2 neighbor even pixels with the appropriate coefficients (Figure 3b). For each process, we have 5 detail pixels and 5 approximation pixels in the same time. In our case, a pyramidal algorithm is described where three LS_1D block are cascaded, and this gives a wavelet transform with three coefficients levels. The same operation is operated for each level. The approximation pixels are processed 5 by 5, and then 10 pixels word is formed to be used at next level.

In implementation, we have 4 outputs, three for detail level and 1 for approximation level. This output can be extract at the same time and we cannot build the wavelet image. To build this image, the coefficients are stored in 4 FIFOs, one for each coefficient. For a fast processing, we create 8 FIFOs: 4 for odd line and 4 for even line. In term of slices for the 3 levels is 1465 slices(10%), and we use 8 BRAMs(8%). To accelerate the data flow, a solution is to reduce information from the wavelet coefficients. One method is to use a threshold and a RLE coding for detail pixels and transfer the approximation pixels. In this case we have a 5:1 compression rate with an acceptable PSNR. This wavelet and compression has been implemented (1D10P-DWT+RLE) in the FPGA and requires 2465 slices(17%) and 9 BRAMS(9%). Another type of compression is to apply a threshold to the wavelet coefficients in order to eliminate low values, and then code these coefficients using a block coding method. This compression method consists of processing the image with an $n \times n$ pixel window. In relation to the above FPGA specifications, it is possible to store inside this chip 8 lines of 1280 pixels each so we chose $n = 8$. For each $n \times n$ pixel window, we test the uniformity of the pixels. If the pixels are not uniform, we divide this window into 4×4 and 2×2 pixels sub-windows and we re-test the uniformities inside these sub-windows. When we apply this compression after a wavelet coding we obtain a compression ratio of 15:1, and the image quality is quite good (PSNR greater than 30dB). To attempt a compression ratio of 30:1, we use a Huffman algorithm after this processing. This algorithm of block coding (BC) is currently implemented. The association of wavelet and block coding (1D10P-DWT+BC) requires around 3500 slices(24%) and 17 BRAMs(17%). To attempt a compression ratio of 30:1, we use a Huffman algorithm after this processing.

3.2 Low level image processing

A lot of fast vision applications do not need to store the full image because only pertinent informations in the image are necessary like object position detection. To illustrate this approach, we present in this section a real-time markers extraction in the context of biomechanics analysis. In the first part of this section we describe the preprocessing allowing objects segmentation and in the second part the marker extraction.

3.2.1 Pre-processing

In many applications, objects in the image are extracted from the background; the image is then binarised: objects are coded at high logic level and background in low logic level. Many segmentation methods exist[16] to extract the object from the background, we have retained a very simple one in term of implementation: binarise with a threshold. In our implementation the threshold is defined by the user and transfer via USB2.0 link to the processing block. The threshold can be processed off-line on the PC in view of the image nature. The user can apply its own segmentation method, this solution is then very flexible. The threshold determination can be also implemented into the FPGA, some local adaptative binarisation based on robust Niblack algorithm [17](using 8x8 or 16x16 neighborhood) can be implemented with a moderate hardware cost: 1286 slices(9%) and 5 BRAMs(5%). In aimed application, the number of the segmented regions are

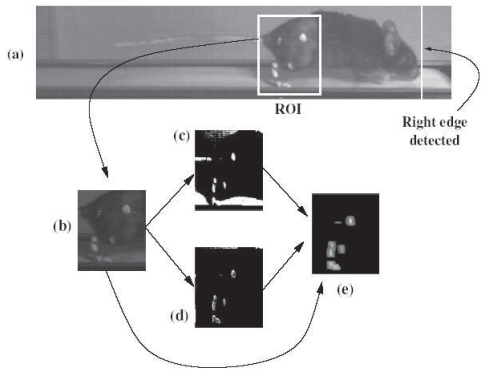


Figure 4: Mouse tracking

low and the resulting high-level regions very homogenous. The information can then be compressed. The transfer is obviously done row by row, therefore, on each row, only the beginning, the end of each high level regions is transferred. For many applications, the obtained compression rate is over 30, that enables to reach a 500 images per second for a full-frame resolution ($1,280H \times 1,024V$). The image frequency increases proportionally with a reduction of the image size. This solution is very economic in terms of hardware implementation only one embedded memory block and less than 330 slices(2%) are required. Most of the logic is mainly due to the large input data flow (10 pixels) simultaneously.

3.2.2 Markers extraction

In this part, we present a specific application concerning the tracking of a mouse running on a moving pavement with a speed of 12, 20 or 37cm/s. For this application, biologists have a simple commercial video camera with a frame rate of 25 images per second and no embedded processes. After discussions and tests with biologists, the conclusion was that standard video acquisition is too slow. A minimum speed of 250 images per second is necessary for a good observation of the leg movement. Only the movement of the leg markers interest the biologists [18]. It is in this perspective that we proposed a embedded markers extraction. In this case, our camera works with a 500 images per second mode. We have implemented inside the FPGA device of our camera some basic real time image processing operators like: ROI detection, image thresholding, image edge detection, image merging, erosion, dilation and centres of gravity calculation. In particular we have previously studied the implementation of real-time centres of gravity calculation in the context of sub-pixel metrology[19]. We used this result for our application. Figure 4 illustrates this application and simple image processing which can extract the X and Y positions of specific markers placed on the mouse. Thus, Figure 4-a shows the mouse running on the moving pavement. On this mouse, 7 markers have been placed. A first step of our algorithm consists to extract the ROI (where the markers are) and for this, using a local edge detector, we obtain the right edge of the mouse (located near its nose). As we know the average length of a mouse, we can easily determine the position of the ROI. From this ROI (shown in Figure 4-b), two image processing are executed in parallel: Image thresholding (Figure 4-c) and Edge detection using Sobel filter

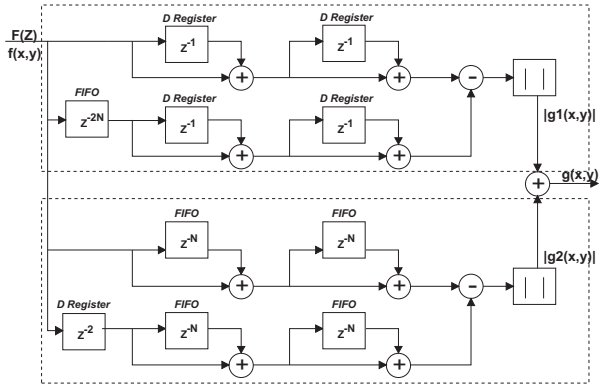


Figure 5: Schematics of a fast parallel Sobel calculation

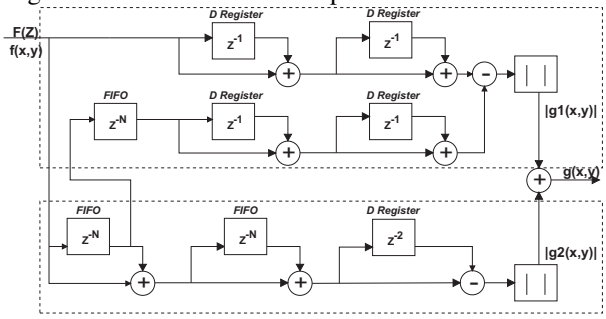


Figure 6: Schematics of a fast parallel Sobel implementation

(Figure 4-d). A logic combinaison between the 3 images (Figures 4-b, 4-c, 4-d) followed by an erosion produce the final image shown in figure 4-e. Erosion allows to suppress isolated white pixels. The final image corresponds to the markers extraction. The final step consists to determine, with a sub-pixel resolution, the coordinates of the centres of gravity of each detected marker[19]. The resulting regions can be then transferred or only centres of gravity to reduce the output flow. In order to illustrate our approach, we only present in this section, the description of real time image processing implementation for Edge detection using Sobel filter. If we consider $f(x,y)$ the grey value of the current pixel, where x and y are the pixel coordinates, the output $g(x,y)$ of the Sobel filter edge detector is given by the following equation: $g(x,y) = |(g_1(x,y))| + |(g_2(x,y))|$ where:
$g_1(x,y) = (f(x,y) + 2f(x-1,y) + f(x-2,y)) - (f(x,y-2) + 2f(x-1,y-2) + f(x-2,y-2))$ and
$g_2(x,y) = (f(x,y) + 2f(x,y-1) + f(x,y-2)) - (f(x-2,y) + 2f(x-2,y-1) + f(x-2,y-2))$.
Using the Z transform respectively to $g_1(x,y)$ and $g_2(x,y)$, we obtain $G_1(z)$ and $G_2(z)$.
Thus, $G_1(z) = (F(z) + 2Z^{-1}F(Z) + Z^{-2}F(Z)) - Z^{-2N}(F(z) + 2Z^{-1}F(Z) + Z^{-2}F(Z))$
where $F(z)$ is the Z transform of $f(x,y)$ and N is the number of pixels per line.
$G_1(z)$ can be factorized as:
$G_1(z) = (F(z).(1 + Z^{-1}).(1 + Z^{-1})) - Z^{-2N}(F(z).(1 + Z^{-1}).(1 + Z^{-1}))$.
Similarely, we obtain g2(x,y),
$G_2(z) = (F(z) + 2Z^{-N}F(Z) + Z^{-2N}F(Z)) - Z^{-2}(F(z) + 2Z^{-N}F(Z) + Z^{-2N}F(Z))$
$G_2(z)$ can be factorized as:

$G_2(z) = (F(z).(1 + Z^{-N}).(1 + Z^{-N})) - Z^{-2}(F(z).(1 + Z^{-N}).(1+Z^{-N}))$.
These last equations show that the $g_1(x,y)$ and $g_2(x,y)$ components can be calculated using simple first order digital FIR filters. The full schematics of the Sobel filter operator is given in Figure 5.
For the implementation, we factorize $G_2(z)$ like this: $G_2(z) = (F(z).(1+Z^{-N}).(1+Z^{-N}))(1-Z^{-2})$.
With this factoring, we see that we can eliminate 3 FIFO from the original architecture shown in figure 5. The new simplified schematic is given in figure 6.
This basic block described has been adapted to the parallel pixel access. The major of the original structure is repeated as adders, but all the delays are obtained with large 100 bits fifo based on BRAM embedded memory. This implementation required 1560 slices(9%) and 9 BRAMs(9%). The erosion implementation has been done with the same methodology. The marker extraction implementation is then requiring 2103 slices and 18 BRAMs and enables to reach 500 frames per second with a full resolution($1,280H \times 1,024V$). The processing is less than 20 ns for 10 pixels, the edge filter is then reaching the performance of 500 MPixels/s (10-bits/pixel).

4. CONCLUSION AND PERSPECTIVES

In this article we have shown that it is possible to implement into a FPGA device several real-time image coding and processing algorithms dedicated to fast vision applications like long fast image recording or real time movement analysis. Long fast image recording is possible using embedded compression approach based on wavelet coding into a FPGA device. Thus, it is possible to capture fast image with $1,280H \times 1,024V$ pixels resolution running at 500 images per second, and transmitting in real-time coded images with a 30:1 compression ration with a PSNR higher than 30dB. We describe also real-time image segmentation implementation dedicated to fast movement analysis. In particular, we have described an application concerning tracking of a mouse running on a moving pavement with a speed of 37 cm/s. For this application, basic image processing have been implemented into the FPGA device like edge detection, erosion and centres of gravity calculation.
In perspectives, we wish to design and to implement new algorithms for Fast Motion capture without markers using pattern recognition approaches like Neural Networks and for this, we will use previous results that we already obtained for real-time face tracking[20].

REFERENCES

[1] E. Fossum, "Active pixel sensors: are ccds dinosaurs?," in *Charge-Coupled Devices and Solid State Optical Sensors III*, Morley M. Blouke, Ed. 1993, pp. 2–14, Proc. SPIE.

[2] D. Litwiller, "Ccd vs cmos:fact and fiction," *Photonics Spectra*, jan 2001.

[3] "http://www.optronis.com," .

[4] "http://www.mikrotron.de," .

[5] "http://www.vdsvossk.de," .

[6] "http://www.baslerweb.com," .

[7] "http://www.cast-inc.com/cores/jpeg-e/," .

[8] P. Schumacher, W. Paluszkiewicz, and R. Turney, "Analysis of a jpeg2000 encoder implemented on a platform fpga," in *image processing*, GSPX, Ed., sep 2003.

[9] P. Schumacher and W. Chung, "Fpga-based mpeg-4 codec," *DSP Magazine*, 2005.

[10] A. Staller, P. Dillinger, and R. Männer, "Implementation of the jpeg 2000 standard on a virtex 1000 fpga.," in *FPL*, 2002, pp. 503–512.

[11] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 7, pp. 674–693, 1989.

[12] A. Grossmann and J. Morlet, "Decomposition of hardy functions into square integrable wavelets of constant shape," *SIAM J. of Math. An.*, vol. 15, pp. 723–736, 1984.

[13] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in *Wavelet Applications in Signal and Image Processing III*, A. F. Laine and M. Unser, Eds. 1995, pp. 68–79, Proc. SPIE 2569.

[14] A. Cohen, Ingrid Daubechies, and J.-C. Feauveau, "Biorthogonal bases of compactly supported wavelets," *Comm. Pure Appl. Math.*, vol. 45, no. 5, pp. 485–560, 1992.

[15] Camille Diou, Lionel Torres, and Michel Robert, "Implementation of a wavelet transform architecture for image processing.," in *VLSI*, 1999, pp. 101–112.

[16] O. D. Trier and A. K. Jain, "Goal-directed evaluation of binarization methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 12, pp. 1191–1201, 1995.

[17] J. Dubois and M. Mattavelli, "Embedded co-processor architecture for cmos based image acquisition," in *ICIP (2)*, 2003, pp. 591–594.

[18] Minerva Gimnez y Ribotta and al., "Activation of locomotion in adult chronic spinal rats is achieved by transplantation of embryonic raphe cells reinnervating a precise lumbar level," *Journal of neurosience*, vol. 20, no. 13, pp. 5144 5152 and 944 947, 2000.

[19] D. Riveo, M. Paindavoine, and S. Petit, "Real time subpixel cross bar position metrology," *Real-time imaging journal*, vol. 8, pp. 105–113, 2002.

[20] F. Yang and M. Paindavoine, "Implementation of a rbf neural network on embedded systems:real time face tracking and identity verification," *IEEE Transactions Neural Network*, vol. 14, no. 5, pp. 1162–1275, 2003.