Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 24163, 16 pages
doi:10.1155/2007/24163

Research Article

High-Speed Smart Camera with High Resolution

R. Mosqueron, J. Dubois, and M. Paindavoine

Laboratoire Le2i, UMR CNRS 5158, Université de Bourgogne, Aile des Sciences de l'Ingenieur, BP 47870, 21078 Dijon Cedex, France

Received 1 May 2006; Revised 27 November 2006; Accepted 10 December 2006

Recommended by Heinrich Garn

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 enabled the development of high-speed video cameras offering digital outputs, readout flexibility, and lower manufacturing costs. In this paper, we propose a high-speed smart camera based on a CMOS sensor with embedded processing. Two types of algorithms have been implemented. A compression algorithm, specific to high-speed imaging constraints, has been implemented. This implementation allows to reduce the large data flow (6.55 Gbps) and to propose a transfer on a serial output link (USB 2.0). The second type of algorithm is dedicated to feature extraction such as edge detection, markers extraction, or image analysis, wavelet analysis, and object tracking. These image processing algorithms have been implemented into an FPGA embedded inside the camera. These implementations are low-cost in terms of hardware resources. This FPGA technology allows us to process in real time 500 images per second with a 1280×1024 resolution. This camera system is a reconfigurable platform, other image processing algorithms can be implemented.

1. INTRODUCTION

The human vision presents high capacities in terms of information acquisition (high image resolution) and information processing (high performance processing). Nevertheless, the human vision is limited because human's reactions to a stimulus are not necessarily instantaneous. The human vision presents spatial and temporal resolution limitations. More precisely, the human vision temporal resolution is close to 100 milliseconds [1]. Moreover, the fast information storage capacity of the human system is difficult to evaluate. On the other hand, the human vision system is very performant in terms of image analysis which extracts relevant information. In the last few years, technical progresses in signal acquisition [2, 3] and processing have allowed the development of new artificial vision system which equals or overpasses human capacities. In this context, we propose to develop a new type of smart camera. The three following constraints have to be considered: a fast image acquisition, images with high resolution, and real-time image analysis which only keeps necessary information.

In the literature, either high-speed cameras without embedded image processing [3] or low-speed smart cameras are presented [4] but we never can find high-speed smart cameras.

Therefore, we propose to develop a new concept of smart camera. In this way, for the last fifteen years, our laboratory has worked in high-speed video system areas [5, 6] and has obtained results for biological applications like real-time cellular contractions analysis [7] and human movement analysis [8]. All these developments were made using CCD (charge-coupled device) imaging technology from Fairchild and FPGA (field-programmable gate array) technologies from Xilinx [9]. The main goal of our system was to provide, at a low price, high-speed cameras (500 images per second) using standard CCD devices in binning mode, with a preprocessing FPGA module connected to a PC-compatible computer. As well as these high-speed camera developments, our laboratory has worked, during these last years, on smart cameras based on standard CMOS (complementary metal-oxide-semiconductor) sensors (25 images per second) and on FPGA technology dedicated to embedded image processing. In the past five years, the use of CMOS sensors instead of CCDs has enabled the development of industrial high-speed video cameras which offer digital outputs, readout flexibility and lower manufacturing costs [10–12].

In our context, fast images present a video data rate close to 6.55 Gbits, this corresponds to 500 images per second with a 1.3 Mpixel resolution. In this high-speed acquisition

context, according to us, the main feature is the huge data flow provided by the sensor output which can represent a major constraint for the processing, the transfer, or the storing of the data. The embedded processings must be adapted to this high-speed data flow and they represent the main core of the high-speed smart camera. The embedded processings enable real-time measurements, such as the fast marker extraction, to be obtained. In the marker extraction application, the output flow is considerably reduced, therefore the transfer and storing of result are simple. On the contrary, if the output flow is not reduced, then an adapted data interface should be selected. In any case, the data are temporarily stored in a fast RAM (random access memory) memory (local or external). The RAM is size-limited, therefore the recording time is only a few seconds long. Our strategy is to propose a compression mode to perform longer sequences and simplify the transfer. The compression can be applied either on the processed images or on the original ones. In this paper, we only present a compression applied on images which have not been processed. The targeted applications are observations of high-speed phenomenon for which fast acquisitions are required. Depending on the compression quality and the precision of the measurement required, an offline processing can be done on the compressed image sequences. In order to optimize the performances of the image compression, we compared different compression algorithms in terms of image quality, time computation, and hardware complexity. First, we compared some algorithms with low hardware complexity like run-length encoding or block coding. Then, as these first compression algorithms are poor in terms of compression ratio, we studied some famous compression algorithms like JPEG, JPEG2000, and MPEG4. This study allowed us to show that a modified and simplified JPEG2000 approach is well adapted to the context of real-time high-speed image compression.

Likewise, in order to implement real-time marker extraction algorithms which are compatible with high-speed image data rate, we used simple image segmentation algorithms. Our camera allows us to record fast image sequences directly on the PC to propose fast real-time processing such as the fast marker extraction.

This paper is organized as follows. Our high-speed camera is described in Section 2. The studied image compression algorithms and their implementations are presented and compared in Section 3. Then, the studied image processing algorithms applied to fast marker extraction are introduced in Section 4. In this section we show a biomechanics application example. In order to compare our system's performances to some other smart camera, we outline our specifications in Section 5. Finally, in Section 6, we conclude our paper and give some perspectives of new developments.

2. HIGH-SPEED SMART CAMERA DESCRIPTION

In order to design our high-speed smart camera, some constraints had to be respected. The first one was of course high-frequency acquisition as well as embedded image processing. Then, some other important specifications had to be taken into account such as low price, laptop, and industrial PC compatiblity. In the literature, most of the high-speed cameras are designed either with embedded memory or using one or several parallel outputs such as a camera link connected to a specific interface board inserted in the PC. In order to record long sequences, the capacity of the embedded memory has to be large, and thus the price is growing. In this paper, we propose a new solution which combines advantages of fast imaging and smart processing without the use of embedded memories. Fast video data output is transferred directly to the PC using a fast standard serial link which avoids the use of a specific board, and thus the full image sequences are stored in the PC memories. This solution makes the most of the continuous progress of PC technologies, in particular memories capacities.

In this section, we describe the different blocks of our high-speed camera and we explain the choices of different components that we used: fast image acquisition using high-speed CMOS sensor from Micron [13] and embedded real-time image processing using FPGA from Xilinx. Finally, at the end of this section, we present the full high-speed smart camera architecture, and in particular the interface choice dedicated to high-speed video data transfer.

2.1. Image acquisition using a high-speed CMOS sensor

Nowadays, in the context of fast imaging, CMOS image sensors present more and more advantages in comparison with CCD image sensors that we will summarize hereafter [14, 15].

(i) *Random access to pixel regions:* in CMOS image sensors, both the detector and the readout amplifier are part of each pixel. This allows the integrated charge to be converted into a voltage inside the pixel, which can then be read out over X-Y wires (instead of using a charge shift register like in CCDs). This column and row adressibility is similar to common RAM and allows region-of-interest (ROI) readout.

(ii) *Intrapixel amplification and on-chip ADC (analogic-digital converter) produce faster frame rates*.

(iii) *No smear and blooming effects:* CCDs are limited by the blooming effect because charge shift registers can leak charge to adjacent pixels when the CCD register overflows, causing bright lights. In CMOS image sensors, the signal charge is converted to a voltage inside the pixel and read out over the column bus, as in a DRAM (dynamic random access memory). With this architecture, it is possible to add an antiblooming protection in each pixel. Smear, caused by charge transfer in a CCD under illumination, is also avoided.

(iv) *Low power:* CMOS pixel sensor architectures consume much less power—up to 100 x less power—than CCDs. This is a great advantage for portable high-speed cameras.

Taking these advantages in consideration, we used the MTM9M413 high-speed CMOS image sensor from Micron

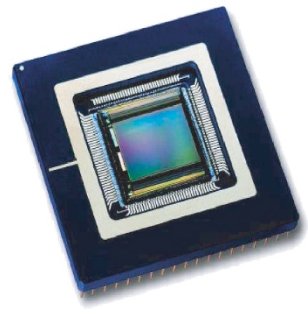


Figure 1: CMOS imager.

in order to design our high-speed camera. The main features of this image sensor are described as follows and illustrated in Figures 1 and 2:

(i) array format: 1280 × 1024 (1.3 megapixels);
(ii) pixel size and type: 12 mm × 12 mm, TrueSNAP (shuttered-node active pixel), monochrome, or color RGB;
(iii) sensor imaging area: H: 15.36 mm, V: 12.29 mm, diagonal: 19.67 mm;
(iv) frame rate: 500 images per second at full-size frame (1280 × 1024), ≥ 10 000 images per second at partial-size frame (1280 × 128);
(v) output: 10-bit digital through 10 parallel ports (ADC: on-chip, 10-bit column-parallel);
(vi) output data rate: 660 Mpixel/s (master clock 66 MHz 500 images per second);
(vii) dynamic range: 59 dB;
(viii) digital responsitivity: monochrome: 1600 bits per lux-second at 550 nm;
(ix) minimum shutter exposure time: 100 nanoseconds;
(x) supply voltage: +3.3 V;
(xi) power consumption: ≺ 500 mW at 500 images per second.

2.2. Image processing with an FPGA device

The used high-speed image sensor delivers, in a pipeline dataflow mode, 500 images per second with a 6.55 Gbits per second data rate. In order to manage this dataflow, it is necessary to add inside our camera a processor able to treat in real time these informations. Some solutions are conceivable and one of them is the use of FPGA.

2.2.1. FPGA advantages for real-time image processing

The bulk of low-level image processing can be split into two types of operations. The first type of operation is where one fixed-coefficient operation is performed identically on each pixel in the image. The second type of operation is neighborhood processing, such as convolution. In this case, the result that is created for each pixel location is related to a window of pixels centered at that location. These operations show that there is a high degree of processing repetition across the entire image. This kind of processing is ideally suited to a hardware pipeline implemented in FPGA, that is able to perform the same fixed mathematical operation over a stream of data.

FPGAs, such as the Xilinx Virtex-II series, provide a large two-dimensional array of logic blocks where each block contains several flip-flops and lookup tables capable of implementing many logic functions. In addition, there are also resources dedicated to multiplication and memory storage that can be used to further improve performance. Through the use of Virtex-II FPGAs, we can implement image-processing tasks at very high data flow rates. This allows images to be processed from the sensor with full resolution (1280 × 1024) at 500 images per second. These functions can be directly performed on a stream of camera data rate as it arrives without introducing any extra processing delay, significantly reducing and, in some cases, removing the performance bottleneck that currently exists. In particular, the more complex functions such as convolution can be mapped very successfully onto FPGAs. The whole convolution process is a matrix-multiplication and as such requires several multiplications to be performed for each pixel. The exact number of multipliers that are required is dependent on the size of the kernels (window) used for convolution. For a 3 × 3 kernel, 9 multipliers are required and for a 5 × 5 kernel, 25 are required. FPGAs can implement these multipliers. For example, with the one-million-gate Virtex-II, 40 multipliers are available and in the eight-million-gate part, this number increases to 168.

2.2.2. Main features of the used FPGA

Taking into account the image data rate and image resolution we selected a VIRTEX-II XC2V3000 FPGA from Xilinx which has the following summarized specifications:

(i) 3 000 000 system gates organized in 14 336 slices (15 000 000 transistors);
(ii) 96 dedicated 18-bit × 18-bit multipliers blocks;
(iii) 1728 Kbits of dual-port RAM in 18 Kbit SelectRAM resources, 96 BRAMs (block RAM);
(iv) 720 I/O pads.

2.3. High-speed camera system

Our high-speed camera system is composed of three boards as shown in Figure 3.

As illustrated in Figure 4, the first board contains the CMOS image sensor and is connected to the FPGA board. This second board has three functions. The first function is the CMOS sensor control, the second function is the real-time image compression, and the third function corresponds to real-time image processing such as edge detection, tracking, and so on. The role of the third board (interface board) is to control, using the USB 2.0 [16] protocol, the image real-time transfer between the FPGA board and the PC computer.

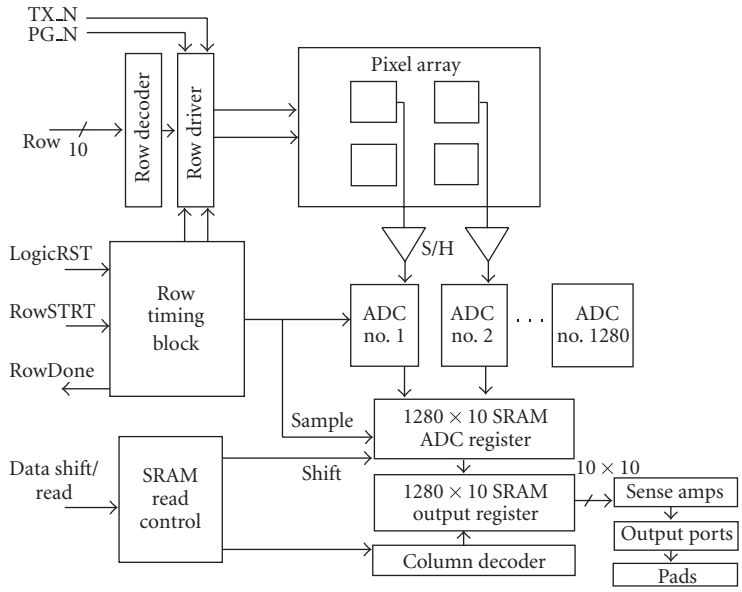


Figure 2: CMOS imager.

The USB 2.0 has the main advantage of being present on any standard PC and also permits the connection of the camera to a PC without any frame grabber. The USB 2.0 features are fully compatible with our results on the targeted applications.

2.4. High-speed camera implementation

We propose to separate high-speed imaging applications into two classes. The first class regroups applications that do not require real-time operations, for instance offline image processing or a visualization of recorded sequences that represents a high-speed phenomenon. The second class regroups applications that require online operations like high-speed feature measurements (motion, boundaries, marker extraction). Therefore, for this second class, most of the time, the camera output flow is considerably reduced.

With our camera design, FPGA embedded solutions are proposed to match with the two presented classes' features. In any case, both solutions must deal with the main feature of high-speed imaging: the important data bandwidth of the sensor's output (in our case, up to 660 Mpixels per second), which corresponds to the FPGA's input data flow.

For the first class application, we propose a solution based on an embedded compression (Section 2). With a data compression, the output bandwidth can be obviously reduced, therefore the data can be easily transferred. The compression choice should be defined to match to output feature (Section 3.1). To demonstrate online capacities of our camera, feature extraction processing has been implemented (Section 4). As the first class is, the measurement is performed at the highest frequency of sensor data output. Hence, the embedded solutions must achieve real-time processing on this large input data flow, moreover the hardware resource should be minimized.

Consequently, some image processing, compression and feature extraction, has been implemented taking into account the required performances and the hardware resource available. In the following sections, two implementation examples are described, one for each class of applications: an embedded compression and a real-time marker extraction. The algorithm selection has been done using hardware considerations.

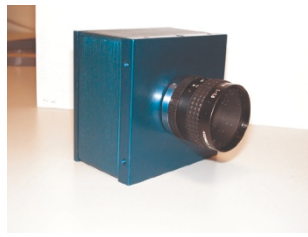
3. EMBEDDED IMAGE COMPRESSION

The compression type, lossless or lossy, is connected to the application's features. For instance, the observation of animal movements can require lossless or lossy compression. A biologist, who focuses on a mouse's behavior does not need images showing the animal's precise movement. On the contrary, if the biologist needs precise measurement on the movement of the mouse's legs, he may need to track precisely the markers. Therefore, an image sequence with a lossless compression may be more relevant. A selection of the lossy compression would limit the number of applications, nevertheless it would have advantage in terms of compression rate than in words of data flow.

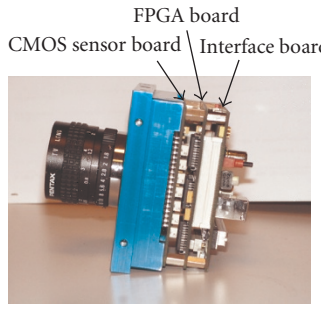
With our proposed compression, full-resolution images can be transferred up to 500 frames per second using a simple USB 2.0 connection.

3.1. Compression algorithms

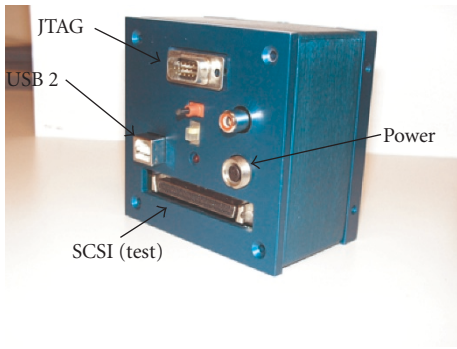
The high-speed CMOS image sensor delivers images with a pixel data rate of 1280×1024 pixels $\times$ 500 images per second $= 655$ Mpixels/s. Each pixel is coded on 10 bits, thus the bit data rate is $655 \times 10 = 6.55$ Gbits/s.



(a) Camera view



(b) Internal camera view



(c) Interface view

Figure 3: High-speed camera system.

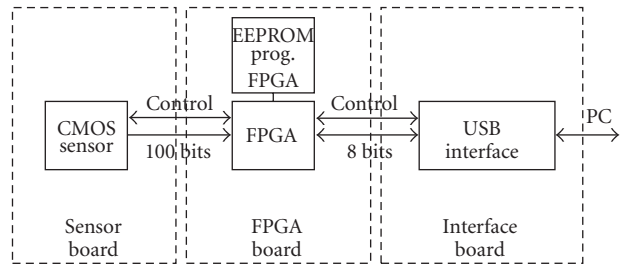


Figure 4: High-speed camera system principle.

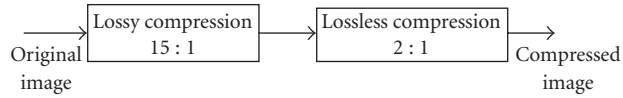


Figure 5: Compression synoptic.

As described previously, information is sent from our high-speed camera to a PC computer through a USB 2.0 link, and this with a peak data rate of 480 Mbits/s. We have obtained an average transfer rate equal to 250 Mbits/s. In our case, to transfer data at the sensor's full speed, the data must be compressed with a compression ratio at least equal to (6.55 Gbits/s)/(250 Mbits/s) = 26.2.

Two main approaches are used in compression algorithms: lossless compression and lossy compression. The main goal of lossless compression is to minimize the number of bits required to represent the original image samples without any loss of information. All bits of each sample must be reconstructed perfectly during decompression. Some famous lossless algorithms based on error-free compression have been introduced like the Huffman coding [17] or LZW coding [18]. These algorithms are particularly useful in image archiving, for instance in the storage of legal or medical records. These methods allow an image to be compressed and decompressed without losing information. In this case, the compression ratio is low (ranges from 2 : 1 to 3 : 1).

Lossy compression algorithms result in a higher compression ratio, typically from 10 : 1 to 100 : 1 and even more. In general the more the compression ratio is high, the more the image quality is low. Some famous methods, designed for multimedia applications, have been also introduced like JPEG, JPEG2000, MPEG2, MPEG4, and so forth. These methods are based on spatiotemporal algorithms and use different approaches like predictive coding, transform coding (Fourier transform, discrete cosine transform, or wavelet coding).

Our compression method choice is based on two main constraints. The first one concerns the real time consideration: how to compress 500 images/s in real-time? The second constraint is related to the image quality, our goal in this case is to compress and to decompress images in order to obtain a PSNR[1] greater than 30 dB. For real-time consideration, the compression process is implemented into the FPGA device and the decompression process is operated using a PC after the image sequence has been recorded.

As shown in Figure 5, we combine lossless compression and lossy compression. The used lossless compression, based on the Huffman algorithm, gives a compression ratio close to 2 : 1. So the lossy compression has to obtain a compression ratio close to 15 : 1. In order to accomplish this target, we studied five lossy compression algorithms: block coding, one-dimensional run-length coding, JPEG, JPEG2000,

[1] PSNR means peak signal-to-noise ratio and is calculated as PSNR = $10\log_{10}((2^B - 1)^2/\text{MSE})$, where B represents the number of bits in the original image. MSE means mean square error and is calculated as MSE = $(1/\text{Npixels})\sum_{x,y}(f(x,y)-g(x,y))^2$, where Npixels is the number of pixels in the image, $f(x,y)$ and $g(x,y)$ are, respectively, the grey levels of original and processed images at x, y coordinates. Reconstructed images are obtained after a compression-decompression process.

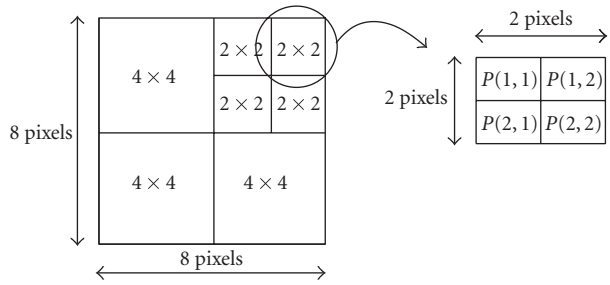
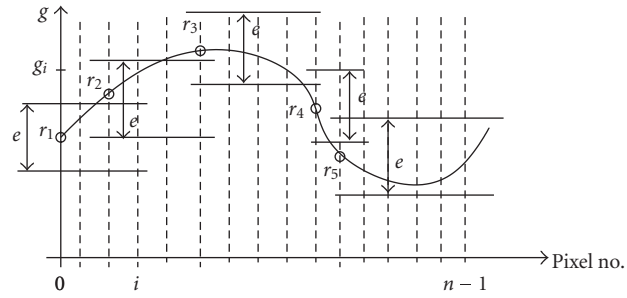


Figure 6: Block coding principle.



--- Original sampling
○ Obtained new sampling
g_i: Grey level of the pixel no.i
r_j: Grey level of the reference pixel no.j
e: Error range
n: Number of pixels per line (1280 here)

Figure 7: One-dimensional run-length coding principle.

MPEG4 and we present our low-cost compression implementation based on wavelet coding using lifting scheme. We describe and compare these algorithms hereafter.

3.1.1. Block coding

This compression method consists of processing the image with an $n \times n$ pixels window. For each $n \times n$ pixels window, we test the uniformity of the pixels. Considering the algorithm's results, an 8×8 window has been selected. If the uniformity is not verified, we divide this window into 4×4 and 2×2 pixels subwindows and we test again the uniformities inside these subwindows. In Figure 6, we give an example of this method. If we consider the 4 pixels $P(1,1)$, $P(1,2)$, $P(2,1)$ and $P(2,2)$ in the 2×2 pixels window, we can compute the following operations:

$$P_{\text{moy}} = \frac{P(1,1) + P(1,2) + P(2,1) + P(2,2)}{4},$$

$$\text{if } P(i,j) \le P_{\text{moy}}, \quad \text{then} \begin{cases} \text{Diff}(i,j) = P_{\text{moy}} - P(i,j), \\ \text{Sign} = 0, \end{cases} \tag{1}$$

$$\text{if } P(i,j) > P_{\text{moy}}, \quad \text{then} \begin{cases} \text{Diff}(i,j) = P(i,j) - P_{\text{moy}}, \\ \text{Sign} = 1, \end{cases} \tag{2}$$

with $i, j = 1, \ldots, 2$.

The obtained code is P_{moy}, $\text{Diff}(1,1)$, $\text{Diff}(1,2)$, $\text{Diff}(2,1)$, $\text{Diff}(2,2)$, $\text{Sign}(1,1)$, $\text{Sign}(1,2)$, $\text{Sign}(2,1)$, $\text{Sign}(2,2)$.

As each of the original pixels $P(1,1)$, $P(1,2)$, $P(2,1)$, and $P(2,2)$ is coded on 10 bits, the 2×2 original pixels subwindow contains 40 bits. If we code P_{moy} on 10 bits, Diff on 2 bits, and Sign on 1 bit, the size of the obtained code is $(1 \times 10) + (4 \times 2) + (4 \times 1) = 22$ bits and the theoretical compression ratio is $40/22 = 1.81$.

3.1.2. One-dimensional run-length coding

In this method, we consider the variations between neighbor pixels on the same image line. If the variations between neighbor pixels are small, we merge these pixels into the same segment with a unique reference grey-level value. Figure 7 is an illustration of this method. For each pixel, we execute the following tests:

$$\text{if } r_j - e \le g_i \le r_j + e, \quad \text{then} \begin{cases} \text{pixel(i) merges with } r_j, \\ \text{else } r_j = g_i, \end{cases} \tag{3}$$

with g_i the current grey-level pixel, r_j the grey-level reference of the jth segment, and e the error range.

The obtained code is $r_1, n_1, r_2, n_2, \ldots, r_{N_{\text{seg}}}, n_{N_{\text{seg}}}$ with n_j the jth segment size and N_{seg} the segments number detected on the current image line.

If we code the reference pixels (r_j) on 10 bits and the segment size (n_j) on 5 bits, the compression ratio for an n-pixel image line (here 1280) is $(10 \times n)/\sum_{j=1}^{N_{\text{seg}}}(10+5)$. This ratio is variable in function of the image's content; the more the image contains high variations, the more this compression ratio is low.

3.1.3. JPEG compression

The principle of the JPEG algorithm [19, 20] for grey-level images is described (for color image, a similar algorithm is applied on each chrominance components) in the following section. The image is split into blocks of 8×8 pixels. A linear transform, DCT (discrete cosine transform) is applied on each block. The transform coefficients are quantified with a quantization table defined in JPEG normalization [20]. The quantization step (or truncation step) is equal to a bit reduction of the samples. This is the main lossy operation of the whole process (see Figure 8).

The entropy coding is the next processing step. The entropy coding is a special form of lossless data compression. It involves arranging the image components in a "zigzag" order employing a run-length encoding (RLE) algorithm that groups similar frequencies together, inserting length-coding zeros, and then using statistic coding on what is left. The

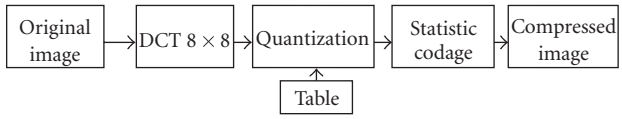


Figure 8: Synoptic JPEG.

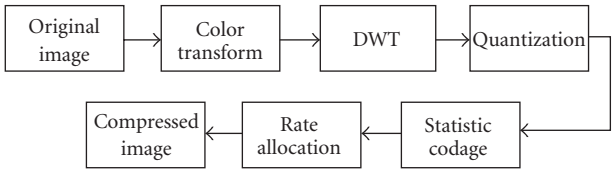


Figure 9: JPEG2000 synoptic.

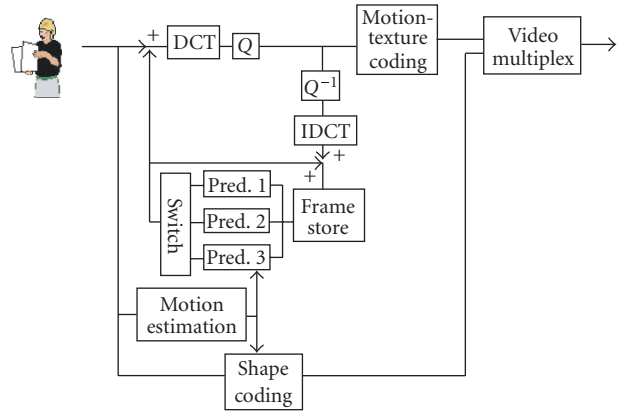


Figure 10: MPEG4 synoptic.

statistic coding is generally a Huffman coding or an arithmetic coding.

The JPEG compression reaches high performances. Using a compression rate of less than 30, a high-quality image is obtained. Nevertheless, for a higher compression rate, a block effect is appearing.

3.1.4. JPEG2000 compression

The JPEG2000 compression [20, 21] is not only more efficient than JPEG compression, it also introduces new functionalities. The JPEG2000 permits the gradual transfer of images, regions-of-interest coding, and a higher errors robustness. The JPEG2000 codec is presented in Figure 9.

The essential processing steps are the color transform, DWT (discrete wavelet transform), the quantization, the entropic coding, and the rate allocation. The color transform is optional. In JPEG2000, the 2D DWT based on Mallat's recursive algorithm is applied on each tile or on the full frame [22]. The result is a collection of subbands which represent several approximation scales. These coefficients are scalar-quantized, giving a set of integer numbers which have to be encoded bit by bit. The encoder has to encode the bits of all the quantized coefficients of a code block, starting with the most significant bits and progressing to less significant bits by a process called the EBCOT (embedded block coding with optimal truncation) scheme [23]. The result is a bit stream that is split into packets, where a packet groups selected passes of all codeblocks from a precinct into one indivisible unit. Packets are the key to quality scalability (i.e., packets containing less significant bits can be discarded to achieve lower bit rates and higher distortion).

3.1.5. MPEG4 compression

The MPEG4 standard is one of the most recent compression codings for multimedia applications. This standard has been developed to extend capacities of the earlier standard (as MPEG1, MPEG2) [24, 25]. The fundamental concept introduced by MPEG4 is the audiovisual objects concept. A video object is represented as a succession of description layers, which offers a scalable codage. This feature permits to reconstruct the video with optimal quality with respect to the constraints of the application, the network, and the terminal. An MPEG4 scene is constituted by one or several video objects characterized temporarily and spatially by their shape, texture, and movement. Figure 10 represents the MPEG4 encoder.

As in JPEG standard, the first two processing steps are DCT and quantization. The quantization level can be fixed or set by the user. The output coming from the quantization function is further processed by a zigzag coder. A temporal compression is obtained with the motion estimation. Indeed, the motion estimation's goal is to detect the differences between two frames. The motion estimation algorithm is based on mean absolute difference (MAD) processing between two image blocks (8×8 or 16×16) extracted from two consecutive images.

3.2. Comparison and analysis for embedded compression

The performance comparison of the compression algorithms [26] is not an easy task. Many features should be considered such as compression rate, image quality, time processing, memory quantity, and so forth. Moreover these features are linked together, for instance, increasing the compression rate can reduce the image quality.

In the previous sections, several types of compression have been described, as well as their performances in terms of compression rate. Indeed, to be efficient for high-speed applications, we need to select a compression with a compression rate greater than 30. The RLE coding and block coding must be associated with other compressions to reach our requirements. The three standard compressions respond to our application's requirements. The image quality must be considered taking into account the applications and the parameter settings (e.g., the selected profile in MPEG4). Anyway, all of the presented compressions can offer high image quality (e.g., PNSR $>$ 25). In term of image quality,

JPEG2000 obtains better performances than JPEG for high compression rates. The MPEG4 codage appears to be well adapted to applications with a low-motion background. These three compressions present the advantage of being standard codages, moreover, all of their functionalities are optional and do not need to be implemented (gradual image transfer).

For a defined application, the compression rate and the image quality are not the only parameters to take into account when selecting an algorithm for hardware implementation. The choice should also consider hardware resource requirements and the processing time. Considering the high-speed imaging constraint, we have chosen the compression algorithm and the proposed hardware implementation. The main difficulty is the large bandwidth and the large input data flow (660 Mpixels/s and 10 parallel pixels). We propose to focus on an implementation based on an FPGA component.

Three significant hardware implementations of famous image compression standards (JPEG, JPEG2000, MPEG4) are then presented as starting point to the implementation analysis. These methods are based on spatiotemporal algorithms and use different approaches like predictive coding, transform coding (Fourier transform, discrete cosine transform, or wavelet coding). First of all, these implementations perform a compression on the video stream at high frequency. The JPEG, JPEG2000, and MPEG4 IPs (intellectual property) can process, respectively, 50, 13, and 12 MPixels per second [27–29]. The hardware resource cost is very high, particularly for JPEG2000 and MPEG4 implementations. Indeed, the three standard implementations require, respectively, 3034, 10800, and 8300 slices with a serial pixel access. Moreover, nearly all these IPs require external memory.

These processing performances do not match with high-speed constraints (660 Mpixels per second = 66 MHz × 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 spatiotemporal dependency does not permit the splitting of data flow between several IPs, the IP must be modified to deal with the input data flow and it improves the output throughput. Obviously, the hardware resource cost will then increase. We propose to restrict the implementation by integrating only parts of the standard compression.

The DCT or the DWT, the quantization associated with coding, and the motion estimation represent crucial parts of the three standards. Unfortunately, their implementations are also expensive in terms of hardware resources. For instance, the DCT and the motion estimation are the most time-consuming steps in MPEG4 standard implementation, therefore many hardware accelerators are still currently proposed [30–32]. Other partial implementations focus on hardware resources reduction, such as a partial JPEG2000 [33]. In this design, the entropy encoder has not been implemented, therefore the complexity is reduced to 2200 slices. Nevertheless, the processing frequency is still not sufficient (33 Mpixels/s), hence the input flow constraint does not match.

Table 1: Comparison of compression implementation. P = parallel data flow, S = serial data flow, RLE = run-length encoding, BC = block coding, Huff = Huffman encoding.

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	10 800/41	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

We have focussed on reducing flexibility to reach a solution with a low cost in terms of hardware resources, that of course matches the input flow requirements. This solution is based on 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 (1D 10P-DWT). We propose two implementations where the wavelet coefficients are, respectively, compressed with RLE, and with an association of block coding and Huffman coding. The second implementation reaches the 660 Mpixel/s. Their performances and the hardware resource cost for the two implementations are reported in Table 1. The full description and quality image are discussed in the next section.

3.3. Embedded compression for high-speed imaging

3.3.1. Wavelet coding using lifting scheme

This compression approach uses the wavelet theory, which was first introduced by Grossmann and Morlet [34], in order to study seismic reflexion signals in geophysics applications, and it was then applied to sound and image analyses. Many authors proposed different wavelet functions, and some of them have very interesting applications for multiresolution image analysis [22].

The advantage of wavelet transform coding 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 11 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 (ranges from 0 to 255), while the wavelet coefficients histogram is thinner and centered on the zero value. Using this property, wavelet coefficients can be coded with better efficiency than the pixels in the original image. The 1D 10P-DWT implementation and two associated compressions are described in the next section. Nevertheless, as a comparaison point with standard compression implementations, their performances and hardware requirements are reported in Table 1.

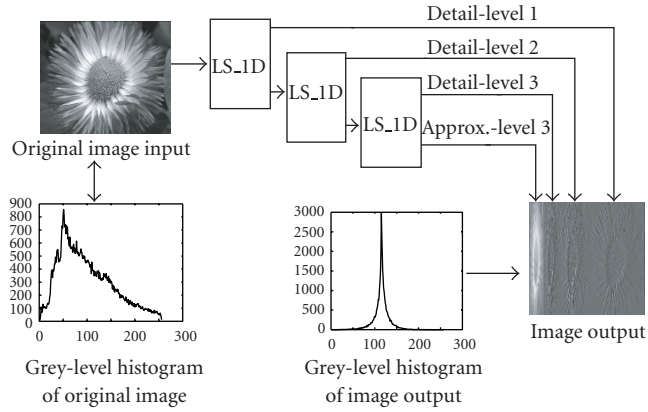


Figure 11: Wavelet pyramidal algorithm.

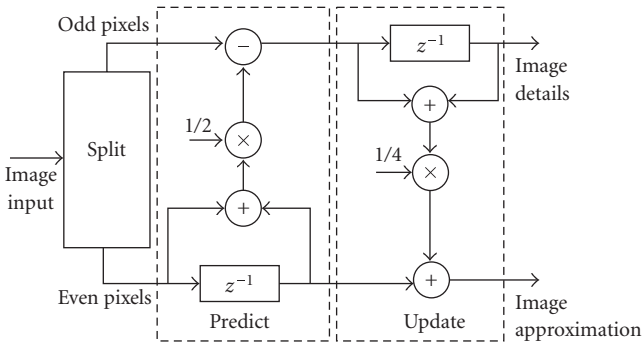


Figure 12: LS-1D algorithm.

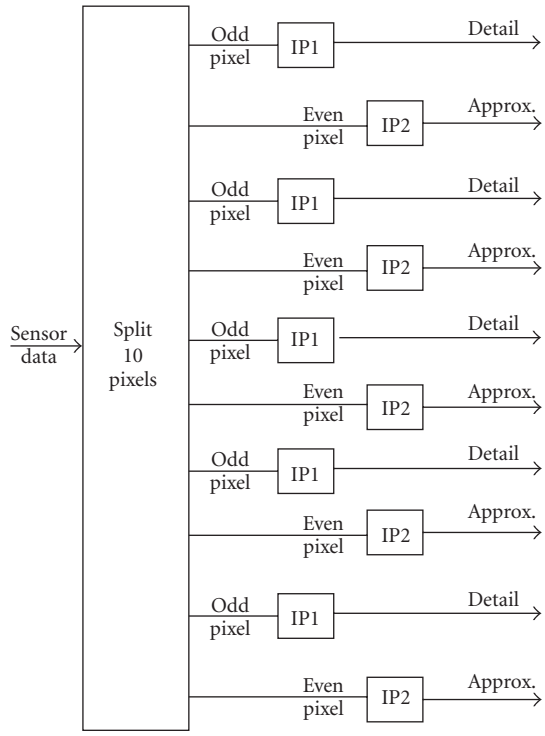


Figure 13: Split 10-pixel IP.

3.3.2. *Wavelets' preprocessing and compression*

In order to implement a wavelet transform compatible with hardware constraints, we use the lifting-scheme approach proposed by Sweldens [35]. This wavelet transform implementation method is described in Figure 12 where we consider the original-image pixels in a data-flow mode (in a 1D representation).

The one-dimensional lifting-scheme (LS_1D) approach is decomposed into three main blocks: split, predict, and update. The split block separates pixels into 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. The image approximation is used in the next LS_1D stage. For this camera, the width of data flow is 10 pixels width and the IPs that we designed are based on it.

The CMOS image sensor send 10 pixels simultaneously, and therefore a real-time parallel processing is necessary. For this, the 10 pixels are split into five odd pixels and five even pixels (Figure 13). For the odd pixel, we designed the IP1 and for the even the IP2. These two IPs are based on the same principle of LS_1D [36, 37]. For the IP1, the central pixel is the odd pixel and we use the two neighbor even pixels with the appropriate coefficients (Figure 14). For the IP2, the central pixel is the even pixel and we use the two neighbors odd pixel and the two neighbors even pixels with the appropriate coefficients (Figure 15). For each process, we have five detail pixels and five approximation pixels in the same time. In our case, a pyramidal algorithm is described where three LS_1D blocks 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 a 10-pixel word is formed to be used at next level.

In implementation, we have four outputs, three for the detail level and one for the approximation level. The four outputs are not synchronous as a result of the cascade of

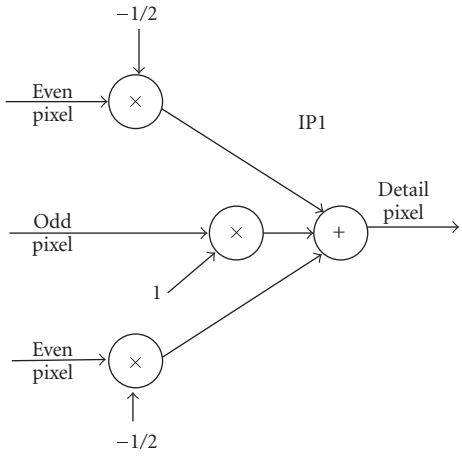


Figure 14: Detail IP.

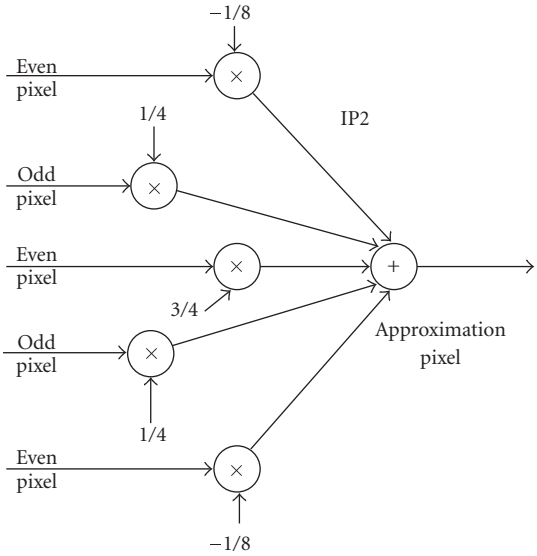


Figure 15: Approximation IP.

LS_1D blocks. Therefore, four FIFO (first-in first-out) memories are used to store the flow. The transform image is generated row by row, hence the FIFO memory's readout is sequential. Two memory banks are implemented. One bank is filled with the current row simultaneously, the second is readout. Therefore, eight FIFO memories are required. The hardware resources for the 3 levels are 1465 slices (10% of the selected FPGA's slices), and 8 BRAMs (8% of the selected FPGA's BRAMs).

A compression is then applied on the wavelet coefficients. We have implemented two types of compressions which are adapted to the important data flow and the nature of the wavelet codage.

The first method consists in using a threshold and then applying an RLE coding for detail pixels. The approximation pixels are not modified. Online modification of the threshold is possible. As we have seen in Section 3.3.1, the wavelet coefficient histogram is thinner and centered on the zero value. With the thresholding, the values close to zero are replaced by zero. The number of consecutive zeros is therefore high. The RLE coding is then very efficient as the implementation is. Indeed, the thresholding can be applied on five parallel samples. The five resulting samples are transferred to the RLE block. If the block is not homogeneous and equal to zero, the previous chain is then closed and transferred. The nonequal resulting coefficients are transferred one by one, then as soon as possible, a new chain is started. In this configuration, we obtain a maximum of 7 : 1 compression rate with an acceptable PSNR (30 dB). This wavelet and compression have been implemented (1D 10P-DWT + RLE) in the FPGA and require 2465 slices (17% of the selected FPGA's slices) and 9 BRAMS (9% of the selected FPGA's BRAMs). This solution does not permit a compression rate superior than 26 to be obtained, therefore, we propose a more efficient solution but it requires more hardware resources.

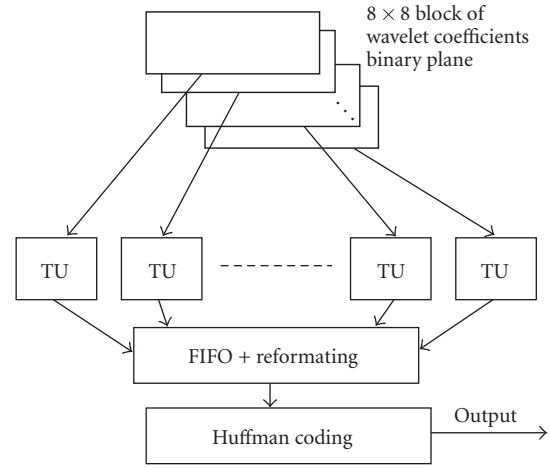


Figure 16: (1D10P-DWT + BC + Huffman) synoptic.

The second proposed compression is based on block coding method. The thresholding is still applied on wavelet coefficients in order to eliminate low values, and these resulting coefficients are coded. This compression method consists of processing the image with an $n \times n$ pixel window. A window size of 8×8 pixels is selected, taking into account the hardware resources and the algorithm's performances. The uniformity of each window is tested. If the window is not uniform, then it is split into subwindows. The 8×8 block can be split into 4×4 subwindows, that also can be split into 2×2 subwindows in case of nonuniformity. The uniformity test is described in Section 3.1.1. The main difference in the uniformity test is due to the algorithm utilization. Each sample is split into binary planes, with a pixel resolution equal to 10 bits, hence 10 binary planes are obtained. The binary planes are coded in parallel (Figure 16). The uniformity test is done with the logical operators due to the binary nature of the plan. The type of operators are suitable for FPGA implementation. In this implementation, a code is generated for each binary plane. The code size is variable. Therefore the reformatting stage requires a FIFO memory. The reformatting adapts the code for Huffman coding block's data input.

The utilization of this compression after a wavelet coding enables a compression ratio of 15 : 1 to be obtained with a high image quality (PSNR greater than 30 dB). To obtain a compression ratio of 30 : 1, a Huffman algorithm is applied after this processing. The association of wavelet and block coding (1D 10P-DWT + BC) requires around 3500 slices (24% of the selected FPGA's slices) and 17 BRAMs (17% of the selected FPGA's BRAMs).

4. EMBEDDED IMAGE PROCESSING AND FEATURE DETECTION

A lot of fast vision applications do not need to store the full image because only pertinent information 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 a preprocessing which allows object segmentations and in the second part the marker extraction.

4.1. Preprocessing

In many applications, objects in the image are extracted from the background. The image is then binarized: objects are coded at high logic level and background in low logic level. Many segmentation methods exist [38] to extract the object from the background, we have retained a very simple one in term of implementation: binarize with a threshold. In our implementation, the threshold is defined by the user and is transferred via a USB 2.0 link to the processing block. The threshold can be processed offline on the PC in view of the image nature. The user can apply his own segmentation method, this solution is then very flexible. The threshold determination can be also implemented into the FPGA, some local adaptative binarization based on robust Niblack algorithm [39] (using 8×8 or 16×16 neighborhood) can be implemented with a moderate hardware cost: 1286 slices (9% of the selected FPGA's slices) and 5 BRAMs (5% of the selected FPGA's BRAMs). In targeted application, the number of the segmented regions is low and the resulting high-level regions are very homogeneous. The information can then be compressed. The transfer is obviously done row by row, therefore, on each row, only the beginning and the end of each high level regions are transferred. For many applications, the obtained compression rate is over 30, enabling a full-frame resolution (1280×1024) of 500 images per second to be obtained. The image frequency increases proportionally with a reduction of the image size. This solution is very economical in terms of hardware implementation, only one embedded memory block and then 330 slices (2% of the selected FPGA's slices) are required. Most of the logic is mainly due to the large input data flow (10 pixels).

4.2. Markers extraction

In this section, we present a specific application concerning the tracking of a mouse running on a moving walkway with a speed of 12, 20, or 37 cm/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 was 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 interests the biologists [40]. It is in this perspective that we proposed an 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 centers of mass calculation (Figure 19).

In particular, we had previously studied the implementation of real-time centers of mass calculation in the context of subpixel metrology [41]. We used this result for our application.

Figures 17 and 18 illustrate this application and the simple image processing which can extract the X and Y positions of specific markers placed on the mouse. Thus, Figure 17(a) shows the mouse running on the moving walkway. On this mouse, 7 markers have been placed. The first step of our algorithm consists of extracting 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. With this ROI (shown in Figure 17(b)), two processing are executed in parallel: image thresholding (Figure 17(c)) and edge detection using a Sobel filter (Figure 17(d)). A logic combination between the 3 images (Figures 17(b), 17(c), 17(d)) followed by an erosion produces the final image shown in Figure 17(e). Erosion allows the suppression on isolated white pixels. The final image corresponds to the markers extraction. The final step consists of determining, with a subpixel resolution the coordinates of the centers of mass (Figure 18) of each detected marker [41]. The resulting regions or only the centers of mass can then be transferred to reduce the output flow. The marker extraction implementation then requires 2103 slices (7% of the selected FPGA's slices) and 18 BRAMs (19% of the selected FPGA's BRAMs) and enables 500 images per second with a full resolution (1280×1024) to be reached. Moreover, other processings can be implemented using the available hardware resources. In order to illustrate our approach, we present in this section the description of real-time image processing implementation for edge detection, erosion, dilation, and centers of mass.

4.2.1. Sobel filter

The choice of the Sobel filter was made considering two main reasons. First, as markers that we wish to detect present a good contrast, the Sobel filter can deliver in this case an edge detection with a good signal-to-noise ratio. Second, the Sobel filter is suitable for FPGA implementation due to the possible hardware implementations. This sum of absolute values is easy to implement (adders, inverters, and test on bits).

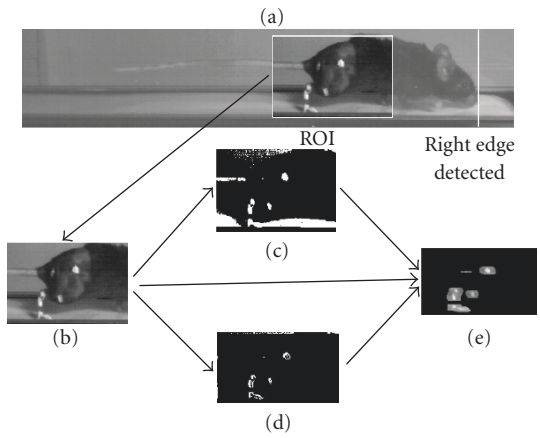


Figure 17: Marker extraction on a mouse.

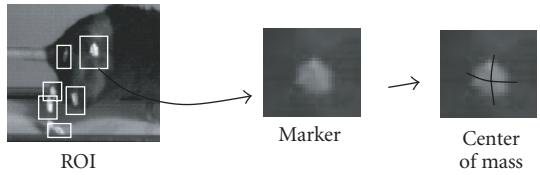


Figure 18: Marker's center of mass.

Moreover, this solution is low-cost in terms of hardware resources. The Sobel implementation has also been selected for its cost in terms of hardware resources. It has been preferred to the more complex solution such as Canny-Deriche implementations that have been explored in the past [42]. 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:

$$g(x, y) = |(g_1(x, y))| + |(g_2(x, y))|, \tag{4}$$

where

$$\begin{aligned} g_1(x, y) &= (f(x, y) + 2f(x-1, y) + f(x-2, y)) \\ &\quad - (f(x, y-2) + 2f(x-1, y-2) \\ &\quad + f(x-2, y-2)), \\ g_2(x, y) &= (f(x, y) + 2f(x, y-1) + f(x, y-2)) \\ &\quad - (f(x-2, y) + 2f(x-2, y-1) \\ &\quad + f(x-2, y-2)). \end{aligned} \tag{5}$$

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,

$$\begin{aligned} G_1(z) &= (F(z) + 2Z^{-1}F(Z) + Z^{-2}F(Z)) \\ &\quad - Z^{-2N}(F(z) + 2Z^{-1}F(Z) + Z^{-2}F(Z)), \end{aligned} \tag{6}$$

where $F(z)$ is the Z transform of $f(x, y)$ and N is the number of pixels per line. Similarly, we obtain for the $g_2(x, y)$ component that

$$\begin{aligned} G_1(z) &= (F(z) + 2Z^{-N}F(Z) + Z^{-2N}F(Z)) \\ &\quad - Z^{-2}(F(z) + 2Z^{-N}F(Z) + Z^{-2N}F(Z)). \end{aligned} \tag{7}$$

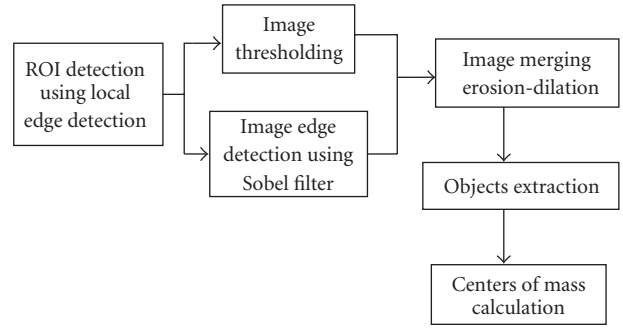


Figure 19: Marker's extraction synoptic.

Figure 20 shows the direct implementation of this filter. In this implementation, 10 adders and 2 FIFOs are needed to compute $g_1(x, y)$ and $g_2(x, y)$. As the data from the CMOS image sensor are delivered in a parallel mode (10 pixels for each clock cycle), all the calculations have to be executed in parallel, and thus the number of adders increases (10×10 adders) while the number of FIFOs stays the same. In order to reduce this arithmetic complexity, some authors [43] proposed to factorize original equations. This allows a cascade of elementary cells to be obtained and the complexity is reduced.

We can illustrate this approach as follows. $G_1(z)$ can be factorized as

$$\begin{aligned} G_1(z) &= (F(z) \cdot (1 + Z^{-1}) \cdot (1 + Z^{-1})) \\ &\quad - Z^{-2N}(F(z) \cdot (1 + Z^{-1}) \cdot (1 + Z^{-1})) \\ &= F(z) \cdot (1 - Z^{-2N}) \cdot (1 + Z^{-1}) \cdot (1 + Z^{-1}). \end{aligned} \tag{8}$$

This last equation shows that the $g_1(x, y)$ component can be calculated using simple first-order digital FIR filters. Similarly, we obtain for the $g_2(x, y)$ component that

$$\begin{aligned} G_2(z) &= (F(z) \cdot (1 + Z^{-N}) \cdot (1 + Z^{-N})) \\ &\quad - Z^{-2}(F(z) \cdot (1 + Z^{-N}) \cdot (1 + Z^{-N})) \\ &= F(z) \cdot (1 - Z^{-2}) \cdot (1 + Z^{-N}) \cdot (1 + Z^{-N}). \end{aligned} \tag{9}$$

Figure 21 shows the optimized cascade implementation of this filter. In this implementation, only 6 adders are needed to compute $g_1(x, y)$ and $g_2(x, y)$ and the architecture is more regular than the original implementation. This approach is a compromise: a FIFO memory is added but the number of adders is considerably reduced. This compromise combined with our parallel approach permits a low-cost solution to be proposed for the large and high-speed data flow. Using this approach, 10 pixels can be processed in parallel using only 60 adders (10×6) and 30 FIFOs in comparison with 100 adders and 20 FIFOs used by the direct Sobel implementation. Moreover, by changing the width of the FIFO from one pixel to 10 pixels, the memories can be organized in 3 large FIFO memories. This organization is more economical in terms of hardware resources considering the memory organization into the selected Xilinx's FPGA. This type of FPGA proposes a configurable internal memory organized

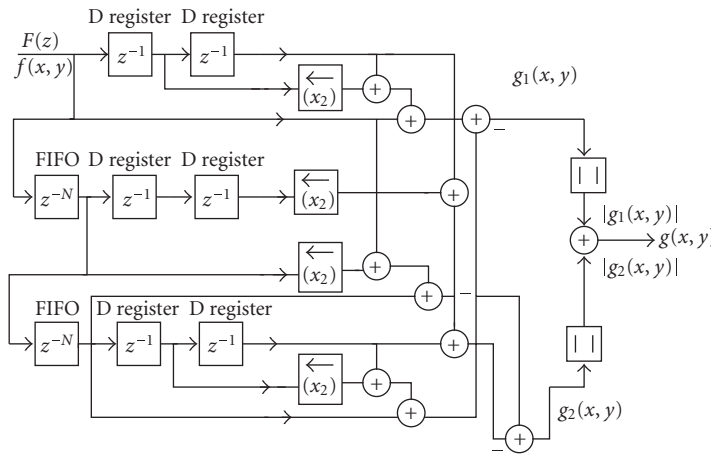


Figure 20: Direct Sobel filter implementation.

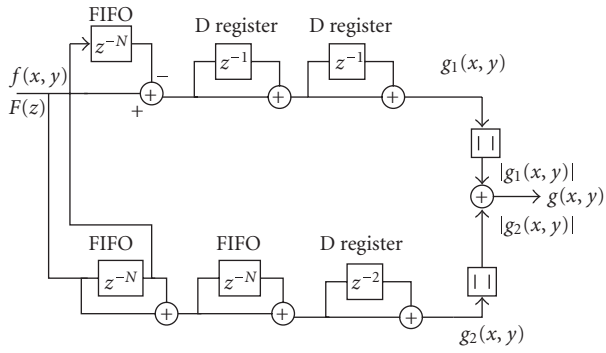


Figure 21: Optimized cascade Sobel filter implementation.

into blocks of RAM (BRAMs). For instance, a FIFO memory which can store 1023 words of 10 bits can be implemented using a single BRAM. Due to the architecture of BRAM, only 3 blocks are required to generate a FIFO memory with a width of 100 bits. Each FIFO can store 511 words of 100 bits. Therefore, organizing the FIFO memories to store words of 100 bits enables the reduction of the number of BRAMs (the depth of the FIFO memory should be taken into account).

Finally, using words of 100 bits enables high-speed performances to be reached by processing 10 pixels in parallel. The described basic block has been adapted to the parallel pixel access as it was for the wavelet implementation. The structure of this basic block is essentially the same. Delays are not necessary between two consecutive pixels in the same 10-pixel word. The majority of the original structure is repeated, but many delays and some adders can be economized due to the presence of the large 100 bits FIFO based on BRAM embedded memory. Therefore, the final architecture can be implemented with 52 adders and 3 FIFO memories with a width of 100 bits. This implementation requires 1560 slices (9% of the selected FPGA's slices) and 9 BRAMs (9% of the selected FPGA's BRAMs).

4.2.2. Erosion and dilatation

The erosion and dilation operations are obtained, respectively, by processing the minimum and the maximum values inside a selected window. Thus, for a 3×3 window, erosion and Dilation are obtained as follows:

$$\begin{aligned} \text{erosion } (x, y) &= \text{minimum } \big(f(x, y), f(x-1, y)\big), \\ &\quad \ldots, \big(f(x-2, y-2)\big) \\ \text{dilation } (x, y) &= \text{maximum } \big(f(x, y), f(x-1, y)\big), \\ &\quad \ldots, \big(f(x-2, y-2)\big). \end{aligned} \tag{10}$$

An implementation of these two operations is illustrated in Figure 22. Again, the time calculation with the FPGA device is less than 15 nanoseconds per pixel. This time depends on the design and the selected FPGA.

4.2.3. Centers of mass

The general form of center of mass is

$$C_k = \frac{\sum_{i=0}^{31} i \cdot p_k(i)}{\sum_{i=0}^{31} p_k(i)}, \tag{11}$$

where C_k is the kth center of mass calculated and where the grey level of pixel i is $p_k(i)$.

The division is not performed inside of the camera, only the numerator and the denominator calculations are processed.

This function is applied for a 30-pixel center of mass. The numerator and the denominator of each center of mass are processed on 30-pixel windows. The computation step is 10 pixels, therefore the hardware resources are reduced. The numerator and the denominator calculations are processed on 10 pixels at each cycle. The numerator's computation requires 10 multipliers and 9 adders (Figure 23). The denominator's computation is more simple and does not require any multipliers. The numerator value for the 30-pixel

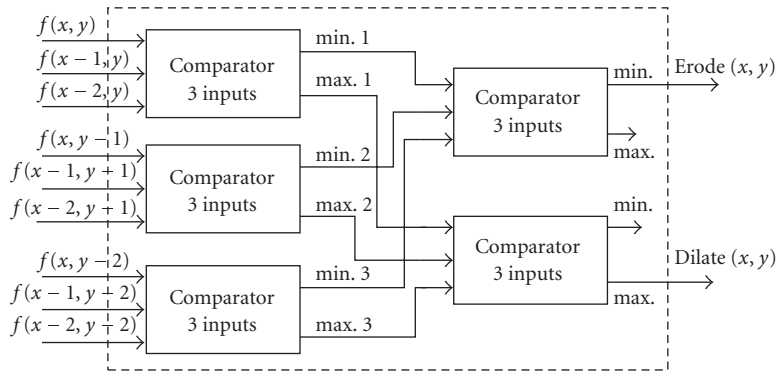


Figure 22: Schematics of erosion and dilation calculation.

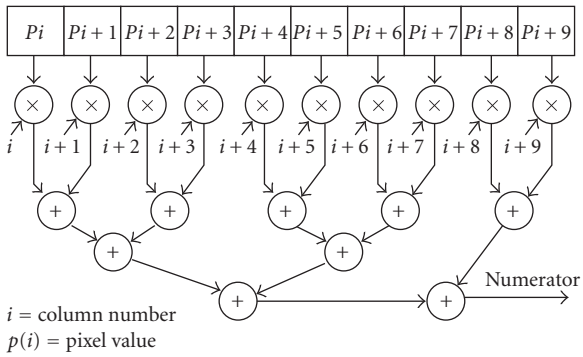


Figure 23: Numerator for a 10-pixel window.

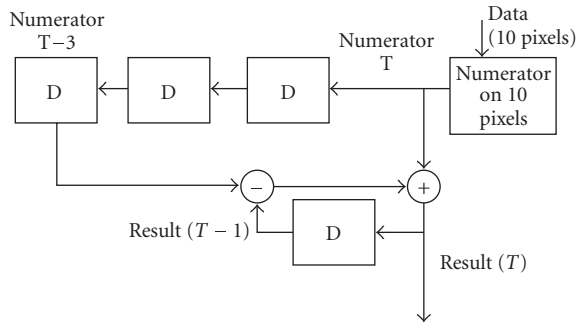


Figure 24: Numerator of 30 pixels.

window is obtained by substracting a previous numerator with 3 clock-cycle delays, then by adding the current numerator value (Figure 24). The same method is applied on the denominator. The calculation is made row by row, and the starting value is 0. The numerator and the denominator are sent to the PC where the division is processed.

It is a simple approach, in terms of hardware resources. The centers of mass implementation requires 636 slices (4% of the selected FPGA's slices) and 0 BRAM (0% of the selected FPGA's BRAMs).

5. SYSTEM PERFORMANCES AND DISCUSSION

In this section, we propose to summarize our system's performances and to compare them with other smart cameras. All the referenced systems propose different processings. Therefore the comparison is difficult. Nevertheless, we propose to insist on the processing's limitations due to the sensor. A main feature of a CMOS sensor is the possibility to select a region of interest (ROI), a window. Hence, we propose to compare the maximum of windows that can be processed in one second. This number, for a defined size, is directly connected to the sensor's specifications, nevertheless it can represent the limitation of the processing. Table 2 has been elaborated with the references [39, 44–47]. Our sensor proposes the highest resolution with the highest speed 4.2.3. There are some other systems which can be performant on reduced-size windows (32×32) but none are as efficient as our system on larger windows as 1024×32. Obviously, the number of windows represents a maximum and it depends on the algorithm being processed and on the processing element. In order to indicate the shown smart camera's possibilities, we precise the nature of the processing element. Our system is probably less flexible than a smart camera with an embedded processor such as [39, 47]. Nevertheless, these systems would not be able to deal with the large input data flow because of the interface.

The high speed and high resolution are not the only camera features to consider. The system is reconfigurable in terms of processing. We have summarized in Table 3 the processing implemented in our camera. The hardware resources are specified for each processing.

6. CONCLUSION AND PERSPECTIVES

In this paper, we showed that embedded processing can be implemented on a high-speed camera with high-resolution. A fast marker extraction running at 500 images per second on the full resolution is presented. We have also showed that it is possible to implement a real-time image compression based on wavelet transform coding into an FPGA device. Hence, we propose a high-speed camera based on a CMOS

Table 2: Comparison table. TUDelft = Technische Universiteit Delft, PE = parallel data flow, Wps = windows per second.

Camera	Resolution × bits	Images per second	PE type	1024 × 32 (Wps)	32 × 32 (Wps)
Our camera	1280 × 1024 × 10	500	FPGA	16 000	16 000
Berry	640 × 480 × 8	25	FPGA	Not applicable	7500
Muelhmann	1024 × 1024 × 10	30	FPGA	960	30 720
Lepisto	1280 × 1024 × 10	18	FPGA	824	26 368
TUDelft and Philips	640 × 480 × 10	50	Trimedia	Not applicable	15 625
Dubois	1280 × 1024 × 10	25	Trimedia + FPGA	1000	32 000

Table 3: Result table. P = parallel data flow, S = serial data flow, RLE = run-length encoding, BC = block coding, Huff = Huffman encoding.

Compression IP	Input flow	Slices/BRAM	Freq. Mpix/s	External memory
1D10P-DWT + RLE	P	2465/9	130	No
1D10P-DWT + BC + Huff	P	3500/17	660	No
Marker extraction	P	2103/18	660	No

image sensor and an FPGA device. The performance of this camera allows, firstly, the acquisition of fast images with a 1280 × 1024 pixels resolution running at 500 images per second, and secondly, the transmission in real-time coded images with a 30 : 1 compression ratio with a PSNR greater than 30 dB using a USB 2.0 link. With this performance, it is possible to store long image records directly inside a PC without any specific memory requirement, which is an advantage because we can benefit from continuously increasing PC performance, particularly concerning memory technologies.

In the future, we will design a new version of our camera using a gigabit ethernet link. This will allow us to decrease compression ratio (6.5 : 1), and thus will increase the image quality (PSNR increasing). With this approach, it will also be possible to design new high-speed cameras with a faster pixel data rate sensor (like 1 Mpixel at 1000 images per second or more).

Using the performance of FPGA technology, integration of new embedded real-time image processing inside the camera is also possible, for example object tracking, image analysis, or pattern recognition [48].

REFERENCES

[1] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, no. 6582, pp. 520–522, 1996.

[2] Y. Kondo, H. Maruno, H. Tominaga, H. Soya, and T. G. Etoh, "An ultrahigh-speed video camera and its applications," in *25th International Congress on High-Speed Photography and Photonics*, vol. 4948 of *Proceedings of SPIE*, pp. 53–58, Beaune, France, September-October 2002.

[3] K. Tajima, K. Tamura, and K. Awano, "Design of 1-M pixels high-speed video camera," in *25th International Congress on High-Speed Photography and Photonics*, vol. 4948 of *Proceedings of SPIE*, pp. 83–88, Beaune, France, September-October 2002.

[4] W. Wolf, B. Ozer, and T. Lv, "Smart cameras as embedded systems," *Computer*, vol. 35, no. 9, pp. 48–53, 2002.

[5] E. Fauvet, M. Paindavoine, and F. Cannard, "Human movement analysis with image processing in real time," in *19th International Congress on High-Speed Photography and Photonics*, vol. 1358 of *Proceedings of SPIE*, pp. 620–630, Cambridge, UK, September 1991.

[6] F. Bouffault, J. Febvre, C. Milan, M. Paindavoine, and J. C. Grapin, "A high-speed video microsystem," *Measurement Science and Technology*, vol. 8, no. 4, pp. 398–402, 1997.

[7] F. Bouffault, C. Milan, M. Paindavoine, and J. Febvre, "High-speed cameras using a CCD image sensor and a new high-speed image sensor for biological applications," in *21st International Congress on: High-Speed Photography and Photonics*, vol. 2513 of *Proceedings of SPIE*, pp. 252–258, Taejon, Korea, August-September 1994.

[8] M. Paindavoine, D. Dolard, and J.-C. Grapin, "Real-time imaging system applied to human movement analysis," in *Advanced Signal Processing Algorithms, Architectures, and Implementations IX*, vol. 3807 of *Proceedings of SPIE*, pp. 150–156, Denver, Colo, USA, July 1999.

[9] Xilinx, http://www.xilinx.com.

[10] Nac, "Memrecam fx K4 High-Speed Color Video System," http://www.nacinc.com.

[11] Photron, "Ultima APX-RS Fastcam," http://www.photron.com.

[12] Weinberger, htpp://www.weinbergervision.com.

[13] Micron, http://www.micron.com.

[14] E. R. Fossum, "Active pixel sensors: are CCDs dinosaurs?" in *Charge-Coupled Devices and Solid State Optical Sensors III*, vol. 1900 of *Proceedings of SPIE*, pp. 2–14, San Jose, Calif, USA, February 1993.

[15] D. Litwiller, "CCD vs. CMOS: facts and fiction," *Photonics Spectra*, vol. 35, no. 1, pp. 154–158, 2001.

[16] USB2.0, http://www.usb.org.

[17] D. A. Huffman, "A method for construction of minimum-redundancy codes," *Proceedings of IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[18] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[19] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Kluwer Academic, Norwell, Mass, USA, 1992.

[20] JPEG Commitee, http://www.jpeg.org.

[21] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice*, Kluwer Academic, Norwell, Mass, USA, 2001.

[22] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[23] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, 2000.

[24] F. C. Pereira and T. Ebrahimi, *The MPEG-4 Book*, Prentice-Hall PTR, Upper Saddle River, NJ, USA, 2002.

[25] MPEG Commitee, http://www.mpeg.org.

[26] D. Santa-Cruz, T. Ebrahimi, J. Askelof, M. Larsson, and C. A. Christopoulos, "JPEG 2000 still image coding versus other standards," in *Applications of Digital Image Processing XXIII*, vol. 4115 of *Proceedings of SPIE*, pp. 446–454, San Diego, Calif, USA, July 2000.

[27] http://www.cast-inc.com/cores/jpeg-e/.

[28] P. Schumacher, M. Paluszkiewicz, R. Ballantyne, and R. Turney, "An efficient JPEG2000 encoder implemented on a platform FPGA," in *Applications of Digital Image Processing XXVI*, vol. 5203 of *Proceedings of SPIE*, pp. 306–313, San Diego, Calif, USA, August 2003.

[29] P. Schumacher and W. Chung, "FPGA-based MPEG-4 codec," *DSP Magazine*, pp. 8–9, 2005.

[30] J. Dubois, M. Mattavelli, L. Pierrefeu, and J. Miteran, "Configurable motion-estimation hardware accelerator module for the MPEG-4 reference hardware description platform," in *Proceedings of IEEE International Conference on Image Processing (ICIP '05)*, vol. 3, pp. 1040–1043, Genova, Italy, September 2005.

[31] M. Alam, W. Badawy, and G. Jullien, "A new time distributed DCT architecture for MPEG-4 hardware reference model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 726–730, 2005.

[32] T. Chiang, M. Mattavelli, and R. D. Turney, "Introduction to the special issue on integrated multimedia platforms," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 589–592, 2005.

[33] A. Staller, P. Dillinger, and R. Männer, "Implementation of the JPEG 2000 standard on a virtex 1000 FPGA," in *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL '02)*, pp. 503–512, Montpellier, France, September 2002.

[34] A. Grossmann and J. Morlet, "Decomposition of hardy functions into square integrable wavelets of constant shape," *SIAM Journal on Mathematical Analysis*, vol. 15, no. 4, pp. 723–736, 1984.

[35] W. Sweldens, "Lifting scheme: a new philosophy in biorthogonal wavelet constructions," in *Wavelet Applications in Signal and Image Processing III*, vol. 2569 of *Proceedings of SPIE*, pp. 68–79, San Diego, Calif, USA, July 1995.

[36] A. Cohen, I. Daubechies, and J.-C. Feauveau, "Biorthogonal bases of compactly supported wavelets," *Communications on Pure and Applied Mathematics*, vol. 45, no. 5, pp. 485–560, 1992.

[37] C. Diou, L. Torres, and M. Robert, "Implementation of a wavelet transform architecture for image processing," in *Proceedings of the 10th International Conference on Very Large Scale Integration (VLSI '99)*, pp. 101–112, Lisbon, Portugal, December 1999.

[38] 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.

[39] J. Dubois and M. Mattavelli, "Embedded co-processor architecture for CMOS based image acquisition," in *Proceedings of IEEE International Conference on Image Processing (ICIP '03)*, vol. 2, pp. 591–594, Barcelona, Spain, September 2003.

[40] M. G. Ribotta, J. Provencher, D. Feraboli-Lohnherr, S. Rossignol, A. Privat, and D. Orsal, "Activation of locomotion in adult chronic spinal rats is achieved by transplantation of embryonic raphe cells reinnervating a precise lumbar level," *Journal of Neuroscience*, vol. 20, no. 13, pp. 5144–5152, 2000.

[41] D. Rivero, M. Paindavoine, and S. Petit, "Real-time sub-pixel cross bar position metrology," *Real-Time Imaging*, vol. 8, no. 2, pp. 105–113, 2002.

[42] E. Bourennane, P. Gouton, M. Paindavoine, and F. Truchetet, "Generalization of Canny-Deriche filter for detection of noisy exponential edge," *Signal Processing*, vol. 82, no. 10, pp. 1317–1328, 2002.

[43] A. Pirson, J.-L. Jacquot, T. Court, and D. David, "A highly efficient method for synthesizing some digital filters," in *Proceedings of European Signal Processing Conference (EUSIPCO '88)*, Grenoble, France, September 1988.

[44] F. Berry and P. Chalimbaud, "Smart camera and active vision: the active detector formalism," *Electronic Imaging*, vol. 14, no. 1, pp. 2–9, 2004.

[45] U. Muehlmann, M. Ribo, P. Lang, and A. Pinz, "A new high speed CMOS camera for real-time tracking applications," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA '04)*, vol. 5, pp. 5195–5200, New Orleans, La, USA, April-May 2004.

[46] N. Lepistö, B. Thörnberg, and M. O'Nils, "High-performance FPGA based camera architecture for range imaging," in *Proceedings of the 23rd NORCHIP Conference*, pp. 165–168, Oulu, Finland, November 2005.

[47] W. Caarls, P. Jonker, and H. Corporaal, "Smartcam: devices for embedded intelligent cameras," in *Proceedings of the 3rd PROGRESS Workshop on Embedded Systems*, pp. 1–4, Utrecht, The Netherlands, October 2002.

[48] F. Yang and M. Paindavoine, "Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1162–1175, 2003.