MIJ2K: Enhanced video transmission based on conditional replenishment of JPEG2000 tiles with motion compensation

Alvaro Luis Bustamante *, José M. Molina López, Miguel A. Patricio

Univ. Carlos III de Madrid, Avda. Univ. Carlos III, 22, 28270 Colmenarejo, Madrid, Spain

Abstract:

A video compressed as a sequence of JPEG2000 images can achieve the scalability, flexibility, and acces-sibility that is lacking in current predictive motion-compensated video coding standards. However, streaming JPEG2000-based sequences would consume considerably more bandwidth. With the aim of solving this problem, this paper describes a new patent pending method, called MIJ2K. MIJ2K reduces the inter-frame redundancy present in common JPEG2000 sequences (also called MJP2). We apply a real-time motion detection system to perform conditional tile replenishment. This will significantly reduce the bit rate necessary to transmit JPEG2000 video sequences, also improving their quality.

The MIJ2K technique can be used both to improve JPEG2000-based real-time video streaming services or as a new codec for video storage. MIJ2K relies on a fast motion compensation technique, especially designed for real-time video streaming purposes. In particular, we propose transmitting only the tiles that change in each JPEG2000 frame. This paper describes and evaluates the method proposed for real-time tile change detection, as well as the overall MIJ2K architecture.

We compare MIJ2K against other intra-frame codecs, like standard Motion JPEG2000, Motion JPEG, and the latest H.264-Intra, comparing performance in terms of compression ratio and video quality, measured by standard peak signal-to-noise ratio, structural similarity and visual quality metric metrics.

Keywords: JPEG2000, Streaming, rtp, Real-time, Motion, Tile, Interframe, Compensation.

1. Introduction

Video communication over IP-based networks is becoming increasingly popular, and it has emerged as one of the most important applications using Internet technology. Motion JPEG2000 (MJP2) [14] is a video coding standard based on the JPEG2000 (also called J2K) image codec compression [13,26]. JPEG2000 employs an intra-frame coding technique that avoids the use of motion compensation adopted by most of the previous standards, such as MPEG-2 [12], or MPEG-4 [21]. Thus, the compression delay achieved by the MJP2 codec could be slightly shorter than such motion compensation-based techniques.

MPEG-2 and MPEG-4 or even Motion JPEG (MJPEG) are the most used and tested codecs. However, the MJP2 codec is now being integrated into new video surveillance devices and systems [4,9]. Compared to MPEG-based systems, the MJP2 codec can take advantage of JPEG2000s unequaled number of features. This standard provides error resilience, regions of interest (ROIs) definition, as well as spatial, component, resolution and quality scalability [5,22].

The bit-stream can be easily parsed and adapted in real-time in each of these scalabilities without having to decode frames. MJP2 is also the leading digital cinema standard currently supported by Digital Cinema Initiatives [7] (a consortium of most major studios and vendors) for the storage, distribution and exhibition of high-definition motion pictures. It is an open ISO standard and an advanced update of MJPEG, which was based on the legacy JPEG format [20].

So, it is expected to provide a better solution for applications that are required to stream high-quality and high-resolution videos over IP-based networks [8,25,16]. The application areas include: digital cinema, PC-based video capturing, remote surveillance, high-resolution medical, satellite imaging and so on.

Also, the MJP2 codec should be considered in real-time transmission systems because it does not employ any motion compensation or inter-frame compression. Instead, each frame is an independent entity encoded with JPEG2000 [13]. This feature will hugely reduce compression and transmission process delay as, contrary to motion compensation techniques, it can be transmitted immediately after an individual frame is compressed.

However, this low delay is achieved at the cost of increasing bandwidth requirements, since it does not reduce any temporal redundancy in videos (which is the main goal of motion compensation-based techniques).

There is not much work related to the optimization of real-time transmissions of JPEG2000 video sequences, at least from our point of view. Some researchers have tried to perform scene analysis to

* Corresponding author.

E-mail addresses: aluis@inf.uc3m.es (A.L. Bustamante), molina@ia.uc3m.es (J.M. Molina López), mpatrici@inf.uc3m.es (M.A. Patricio).

reduce Motion JPEG2000 video surveillance delivery bandwidth and complexity [17]. This work separates foreground objects from the background, which it compresses at different qualities, selecting better qualities for the important objects in the sequence. The tests run in this case are insufficient, and the research paper does not set out a detailed process of how to detect background/foreground objects and how it is used in JPEG2000 sequences.

Other research, closer to the process proposed in this paper, is described in [18,19,6]. They perform a conditional replenishment of JPEG2000 code-blocks with motion compensation, but this technique appears to be too complex to be applied in real-time environments, since they work with a low-level code stream. Clients and servers would also have to be purposely designed for use with these techniques, and standard protocols such as described in RFC 5371 [10] designed for JPEG2000 transmission could not be used. Neither do they test the delay introduced by the techniques that they describe, and all results appear to be simulated, and not tested on a real implementation.

To solve this problem, we propose the Motion Inter-Frame JPEG2000 (MIJ2K) method. As described in this paper, it applies a real-time motion compensation technique to the MJP2 sequences before transmission. It will lead to a significant reduction in bandwidth requirements without adding extra delay to the compression and transmission process.

In previous research conducted by A.L. Bustamante et al. in [3] a system for real-time transmission of JPEG2000 live video streams over a real-time transport protocol (RTP) was applied in compliance with RFC 5371 [10]. It takes advantage of the JPEG2000 code-stream structure to perform intelligent transmission. In this paper, we also proposed a pre-transmission optimization of se-quences, using a low-complexity motion detection system. MIJ2K codec will add a real-time inter-frame compression technique in a native intra-frame system. It combines the benefits of inter-frame techniques (greater compression ratio and higher quality) with the advantages of using an intra-frame method (low complex-ity compression and fast delivery).

Added to a common JPEG2000 video streaming system, the MIJ2K technique will hugely improve transmission, saving bandwidth, and achieving better qualities for the same bit-rate, as shown throughout this paper. This paper fully describes MIJ2K, including a real-time motion detection and compensation system. For streaming purposes, we use it over the J2K Streaming RTP developed in [3]. We compare MIJ2K against other standard codecs, like standard MJP2, MJPEG, and H.264-Intra. To evaluate the quality of MIJ2K, we will test the codec in terms of delivery delay in seconds; quality of video sequences measured by peak signal-to-noise ratio (PSNR), structural similarity (SSIM) and the visual quality metric (VQM); and compression ratio, measuring the average bit-rate for transmission against other codecs.

The proposed MIJ2K is patent pending and can be easily applied to the previous work done in [3]. Also it is RFC 5371 compliant.

The remainder of the paper is organized as follows. Section 2 provides background on JPEG2000 code streams. Section 3 introduces some fundamentals about video compression evaluation. Section 4 fully describes the proposed architecture, including the adopted real-time motion compensation technique. Section 5 is used to evaluate the quality of MIJ2K compared with other standard codecs. Finally, we present some conclusions about the proposed method.

2. JPEG2000 code stream

JPEG2000 is a wavelet-based [1] image compression standard, created by the Joint Photographic Experts Group committee in the year 2000, with the intention of superseding their original discrete cosine transform-based JPEG standard (dating from 1992).

Although JPEG2000 offers a modest increase in compression performance compared with JPEG, its main benefit is significant code-stream flexibility. The code stream obtained after compression of an image with JPEG2000 is scalable, meaning that it can be decoded in a number of ways. For instance, by truncating the code stream at any point, we can get a representation of the image at a lower resolution or signal-to-noise ratio. By ordering the code stream in various ways, applications can achieve significant performance increases.

Some of the characteristics of JPEG2000 images are:

- Superior compression performance: At high bit rates, where artifacts become nearly imperceptible, JPEG2000 has a small machine-measured fidelity advantage over JPEG. At lower bit rates (e.g., less than 0.25 bits/pixel for gray-scale images), JPEG 2000 has a much more significant advantage over certain JPEG modes: artifacts are less visible and there is almost no blocking. The compression gains over JPEG are attributed to the use of DWT and a more sophisticated entropy encoding scheme [23].
- Multiple resolution representation: JPEG2000 decomposes the image into a multiple resolution representation in the course of its compression process. This representation can be put to use for other image rendering purposes beyond compression as such [13].
- Progressive transmission by pixel and resolution accuracy, commonly referred to as progressive decoding and signal-to-noise ratio (SNR) scalability: JPEG2000 provides efficient code-stream organizations which are progressive by pixel accuracy and by image resolution (or by image size). This way, after a small part of the whole file has been received, the viewer can see a lower quality version of the final picture. The quality then improves progressively as more data bits are downloaded from the source. The 1991 JPEG standard also has a progressive transmission feature, but it is rarely used.
- Lossless and lossy compression: like JPEG 1991 [28], the JPEG2000 standard provides both lossless and lossy compression in a single compression architecture. Lossless compression is provided by the use of a reversible integer wavelet transform in JPEG2000.
- Random code-stream access and processing: JPEG2000 code streams offer several mechanisms to support spatial random access or region of interest access at varying degrees of granularity. This feature is achieved in part by the concept of tiling introduced in JPEG2000, where an image is split into so-called tiles, rectangular regions of the image that are transformed and encoded separately. For each encoded tile there are also other random-access mechanisms such as the concept of precincts.
- Error resilience: like JPEG 1991, JPEG2000 is robust to bit errors introduced by noisy communication channels because data is coded in relatively small independent blocks.
- Side channel spatial information: It fully supports transparency and alpha planes.

Apart from the above features, the main benefit of using JPEG2000 for video streaming is that, unlike other video compressors including MPEG-4, compression could be done in real-time [3] because it is an intra-frame codec. Thanks to this feature, JPEG2000 can be used in events that require real-time transmission, like video surveillance.

As far as our proposal is concerned, however, the main advantage is that the image can, optionally, be partitioned into smaller independent non-overlapped rectangular blocks called *tiles* [13]. We will exploit this exceptional feature, provided by this compressor alone, to perform real-time inter-frame compression using the proposed conditional tile replenishment method. We will employ a

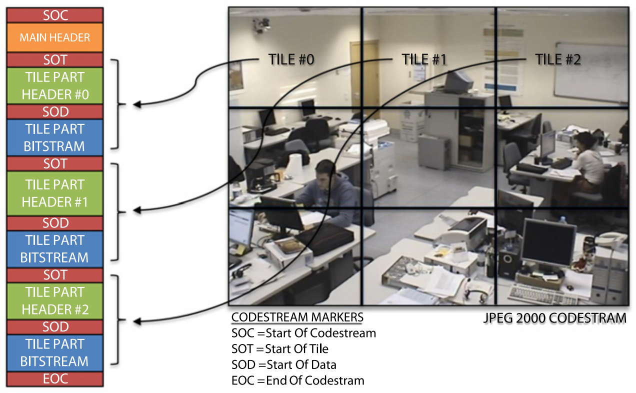

Fig. 1. JPEG2000 code stream structure.

new block-based difference coding technique [24] for this task, having tiles assume the role of blocks.

Tiles can be any size, and the whole image can even be considered as one single tile. Once the size has been chosen, though, all the tiles will be of the same size (except, optionally, tiles on the right and bottom borders). Dividing the image into tiles is advantageous in that the encoder/decoder will need less memory to encode/decode the image, and it can opt to encode/decode only selected tiles to achieve a partial coding/decoding of the image. It will provide full control of whatever area of the image is being compressed, decompressed, transmitted, etc. However, a trade-off exists when selecting the tile size. As discussed in [33], a small tile-size reduces the JPEG2000 compression efficiency, limiting the interest of using a wavelet transform. Also it may create blocking artifacts at moderate to high compression ratios. We use in this work small tile sizes, since the worse compression efficiency is highly compensated with the motion compensation technique, as we found out in [2].

Fig. 1 presents an example of how the J2K code stream is structured, and how an image is divided using tiles. The first marker present is the Start of Code Stream (SOC). This is followed by a main header (MH), which includes the common parameters required for image decoding. The tile-part header (TH) contains the necessary information for decoding each tile. It is followed by the corresponding tile-part bit-stream. Finally, the End of Code-Stream (EOC) marker denotes the termination of a J2K code stream.

Notice that each region of the image occupies a definite region in the J2K code stream. Thanks to the method for defining headers [13], each region can also be accessed randomly.

3. Video compression evaluation

Since video compression somehow alters the original videos, methods are required to evaluate the quality of the compressions performed. They are directly concerned with the data reduction quantity, and the compressed video quality (similarity) compared with the original video.

As regards quality, or similarity, there are some standard quality metrics. The most common objective quality metrics are the mean square error (MSE) and the peak signal-to-noise ratio (PSNR). The MSE_i is the cumulative squared error between the original (f_i) and compressed ($\hat{f}_i$) images. So, MSE is calculated pixel by pixel over a frame of $X \times Y$ (1). On the other hand, PSNR_i represents quality on a logarithmic decibel (dB) scale. This is calculated from the MSE_i value (2). When comparing compression codecs, they are used as an approximation to human perception of reconstruction quality. Higher PSNR (and lower MSE) values would normally indicate that the reconstruction is of higher quality.

$$\text{MSE}_i = \frac{1}{XY}\sum_{x=1}^{X}\sum_{y=1}^{Y}\{f(x,y) - \hat{f}(x,y)\}^2, \tag{1}$$

$$\text{PSNR}_i = 10 \cdot \log_{10}\frac{255^2}{\text{MSE}_i}. \tag{2}$$

MSE and PSNR are both objective quality metrics, but there exist other quality metrics that take subjective quality measurements, since PSNR and MSE have proved to be inconsistent with human eye perception [11].

These techniques are based on the human vision system and theoretically are more like our perception. Some current techniques designed to fulfill this purpose are structural similarity (SSIM) [29] and the visual quality metric (VQM) [32].

The SSIM metric is calculated on various windows of an image. The measure between two windows of size $N \times N$, x and y, is defined in (3), where μ_x is the average of x; μ_y is the average of y; σ_x^2 is the variance of x; σ_y^2 is the variance of y; cov_{xy} is the covariance of y; $c_1 = (k_1L)^2$, $c_2 = (k_2L)^2$ are two variables to stabilize the division with weak denominator; L is the dynamic range of the pixel values (typically, this is $2^{\#bits\ per\ pixel} - 1$); and $K_1 = 0.01$ and $K_2 = 0.03$ are default values.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2cov_{xy} + c_2)}{\left(\mu_x^2 + \mu_y^2 + c1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)}. \tag{3}$$

The resultant SSIM index is a decimal value between −1 and 1, and value 1 is only reachable if there are two identical sets of data. Typically, it is calculated on 8×8 window sizes.

On the other hand, VQM is based on Watson's Digital Video Quality (DVQ) model [30,31]. DVQ uses the Discrete Cosine Transform (DCT). The process for this measurement is more complex than SSIM, and Fig. 2 outlines its flowchart. The VQM algorithm is as follows. Both the compressed and original video sequences are converted to the YUV color space, and undergo DCT transformation. The DCT coefficients are converted to units of local contrast, which is defined as the ratio of the AC amplitude to the temporally low-pass filter DC amplitude. The local contrasts are subjected to spatial contrast sensitivity functions for the static and dynamic frames, and the DCT coefficients are converted to just noticeable differences. The video sequences are subtracted to produce a difference sequence, and this is subjected to a contrast masking in a maximum operation and a weighted pooling mean distortion. In this case, higher values in this metric denote less similarity with the original frame, and zero values are assigned when the likeness is perfect.

On the other hand, the data reduction quantity is measured by the compression ratio factor (CR). CR indicates by how much the image has been compressed, and is defined in (4). Higher values of CR will be equivalent to better compression ratios.

$$CompressionRatio = \frac{UncompressedSize}{CompressedSize} \tag{4}$$

These standard metrics are useful for evaluating the quality of compressions, but some standard videos are also needed to evaluate and compare compression with other codecs. There are several

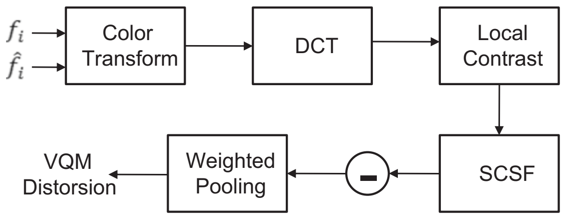

Fig. 2. Overview of VQM.

Fig. 3. Standard videos used for compression evaluation.

Fig. 4. 'Surveillance' sequence for compression evaluation.

sequences available for this purpose. We have selected some from the Xiph.org Test Media repository [15]. The selected sequences are 'Akiyo' and 'Hall monitor', shown in Fig. 3.

The real-time motion compensation technique used in this work, which is described further, performs better in videos with a still background. In this paper, we will demonstrate the particular skills of our proposal in these type of scenarios, which are very common in video surveillance systems. To evaluate the encoder in its real environment, we also propose a new video sequence, called 'Surveillance' [1], that has different features to these standard sequences: video surveillance from a fixed camera. Fig. 4 shows a frame from this sequence. It is recorded from a fixed camera monitoring the back of a building.

4. MIJ2K architecture

This section details the MIJ2K method designed to introduce an inter-frame technique to any JPEG2000-based streaming system. This procedure is applied directly to the streams that are being transmitted, and consequently, only blocks (denoted tiles in JPEG2000) with movement will be transmitted.

The inter-frame technique adopted in MIJ2K is a real-time specific block-based difference coding, adapted to JPEG2000 streams. This technique is useful in the real-time transmission scheme because it provides low computational complexity. It also preserves the real-time latency provided by the native JPEG2000 intra-frame architecture.

The MIJ2K architecture designed for this task is outlined in Fig. 5 and explained in more detail in the following sections. The general operating procedure is as follows.

A common JPEG2000-based streaming system is basically divided into three steps. The first step is related to frame acquisition and compression. It is followed by the transmission of the resulting compressed frames. Finally, it ends with the reception and display of each frame. The acquisition and compression step in these systems is not complex. Each frame is acquired and compressed separately, and can be transmitted as soon as it is compressed.

In the proposed MIJ2K streaming architecture, an extra process is inserted in the compression step. Instead of compressing each whole frame, it compresses and transmits only the areas that are different (changing tiles) from the previously transmitted frame. Compressing only changing tiles improves compression, transmission, and decoding performance, since it hugely reduces the total amount of data for management.

For example, Fig. 6 shows the tiles detected as changing in frame 127 of the 'Akiyo' sequence. They are marked with a green rectangle. Notice how only 17% of the tiles in this frame are detected as changing. When the MIJ2K method is applied to the whole 'Akiyo' sequence, it saves around 87% of bandwidth compared with a native JPEG2000 streaming system.

Changing tiles are detected using a reference frame F_i^R that stores a representation of the last frame transmitted. Each new frame F_i^S to be transmitted is compared tile by tile with the reference frame F_i^R, in order to detect the tiles that differ with their counterparts. Frame by frame, F_i^R is updated with the tiles that are detected as 'changing' to create a real representation of what the client is viewing.

The client receiving this modified stream (a JPEG2000 code stream containing just some of the tiles) will have to decode each tile received and use it to update the displaying frame F_i^D. The client can easily locate each tile position since they are identified by a tile number. Knowing that all tiles (except image boundaries) are of the same size, it is easy to calculate the position of the tile inside the full image.

All the subsystems illustrated in Fig. 5, and the details of the designed MIJ2K system are explained in more detail in the following sections.

4.1. JPEG2000 encoder

This module compresses the JPEG2000 still images. It basically takes and compresses the source frame F_i^S using tile partitioning. The encoder should be modified to assure that only the tiles specified by the T_i^{INDEX} parameter, and not all the tiles of the frame are compressed. This is feasible since the tiles in JPEG2000 images can be compressed and decompressed separately. Ideally, then, only the changing tiles are compressed. This will save compression

[1] http://www.giaa.inf.uc3m.es/miembros/alvaro/jjbase.avi

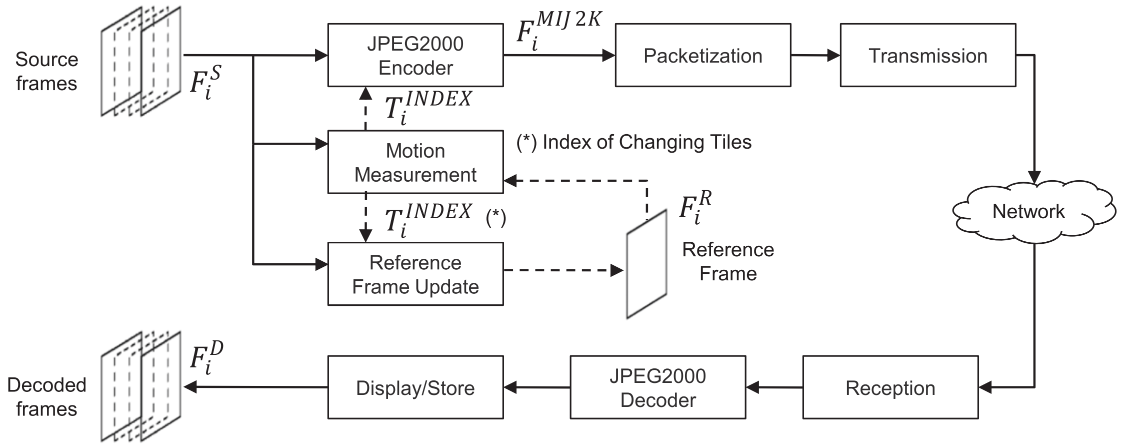

Fig. 5. Overall functioning of the MIJ2K architecture, performing real-time selective tile compression and transmission.

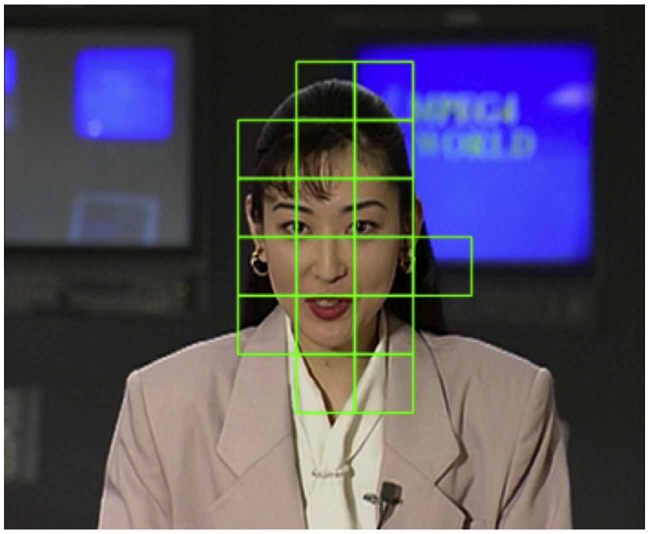

Fig. 6. 'Akiyo' sequence. Tiles detected as changing in frame 127. In this frame, a bandwidth saving of around 83% is achieved.

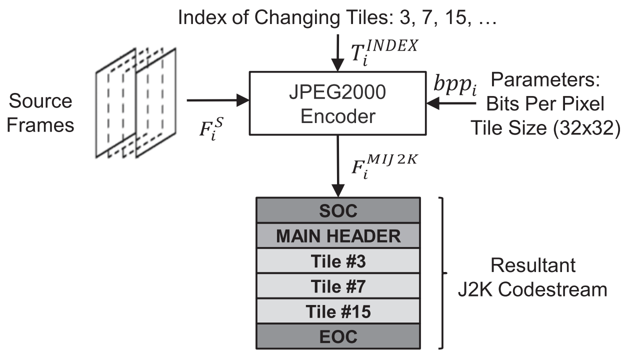

Fig. 7. MIJ2K encoder operation.

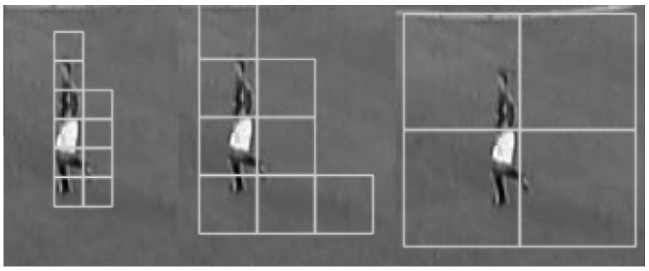

Fig. 8. Overlay areas with different tile sizes. From left to right 16 × 16, 32 × 32 and 64 × 64.

time, improving the overall system operation. All frames will be compressed with the same quality, given by bpp_i.

Fig. 7 is a diagram of this module. It shows all the required inputs and outputs. These are described in more detail in the following:

- Source frame F_i^S: This is the last image acquired by the digitizer board or digital camera, that is, the frame that is going to be transmitted. It should be formatted in some J2K encoder-understandable format, for instance, a RAW 8 *bpp* or 24 *bpp* RGB image (depending on whether it is a gray-scale or color picture).
- Quality bpp_i: This parameter is related to compression quantity, or target quality after each still image has been compressed. This parameter is usually expressed as the quantity of bits used to represent each pixel in the generated JPEG2000 code stream, that is, bits per pixel (*bpp*).
- Tile size T_{SIZE}: This parameter defines the size of the tiling performed in the J2K compressed image. Once the tile size has been set, all images will be compressed in separate squared regions of the same size. Tile size is variable, but, theoretically, small tile sizes can achieve a better fitting to moving objects. This is illustrated in Fig. 8, where the soccer player is a moving object in the sequence, and little tiles are a better fit for the player. But we have found that the best tile size for a motion compensation technique using JPEG2000 code streams is 32 × 32. This was optimized in [2] before the release of the MIJ2K codec. So, the default value will be set to 32 × 32.
- Changing tiles index T_i^{INDEX}: This input is delivered by the motion measurement subsystem (this process is detailed later), as shown in Fig. 5. It indicates the index of the tiles that contain some movement, that is, the tiles that should be compressed. In this case, the JPEG2000 encoder should compress these tiles (regions) of the image only. This will improve the compression delay since the encoder does not have to compress the whole image.
- J2K code stream F_i^{MIJ2K}: This is the J2K code stream output after compression. It should contain only the changing tiles specified in the T_i^{INDEX} input. The tiles bit-stream should be between the main header and End of Code-Stream markers, as shown in

Fig. 7. In this case, only tiles 3, 7 and 15 have been detected as changing. The F_i^{MIJ2K} output will be passed directly to the Packetization Subsystem, which will transmit the frame over the network.

This process defined as in Eq. (5), where the J2K function represents the encoder, and the inputs of the function are: source frame F_i^S, index of tiles for compression T_i^{INDEX}, quality bpp_i and size of tiles T_{SIZE} with the default value 32 × 32.

$$F_i^{MIJ2K} = J2K\left(F_i^S, bpp_i, T_i^{INDEX}, T_{SIZE} = 32 \times 32\right). \tag{5}$$

4.2. Motion measurement

This subsystem manages the motion measurement between two consecutive frames and detects the tiles index of images that contain some movement. The algorithm proposed for this task achieves a low complexity in order to meet real-time transmissions, instead of look for high-efficient motion detection performed by common inter-frame encoders. Such sophisticated techniques, like used in H.264/MPEG-4 AVC, usually are not feasible for critical real-time environments such as video surveillance.

This algorithm, as shown in Fig. 9, takes two frames, F_i^S and F_i^R (source and reference), for comparison. Also it has to know the selected tile size T_{SIZE} used in the compression step.

All the inputs and outputs of this system are described below:

- Source frames F_i^S and F_i^R: F_i^S should be the same image as used in the J2K encoder in some affordable format where the pixel values of the image can be directly manipulated. The other required image is the reference frame F_i^R. For comparison purposes, it should have the same format as the source frame F_i^S.
- Tile size T_{SIZE}: As compression is performed using the concept of tiling, motion should be measured in tile units. So, this subsystem must know the working tile size T_{SIZE}. It should take the same value as used in the J2K encoder, that is, a default value of 32 × 32.
- Changing tiles index T_i^{INDEX}: As mentioned in the JPEG2000 encoder section, this subsystem should provide the index of tiles containing some movement. The indexes could range from 0 to the total amount of tiles in any order and without any restriction. If no tiles with movement are detected (the current frame is almost equal to the last frame), this subsystem should somehow notify this circumstance and then send no tile for the current F_i^S frame (but the client side must be notified).

$$T_i^{INDEX} = Motion\left(F_i^S, F_i^R, T_{SIZE} = 32 \times 32\right). \tag{6}$$

So, this subsystem can be defined as in (6). In this case, we will specify what the 'Motion' function does in more detail, since this is the most important part of the adopted inter-frame technique. Fig. 10 shows this function in detail. We also describe all the tasks involved as follows:

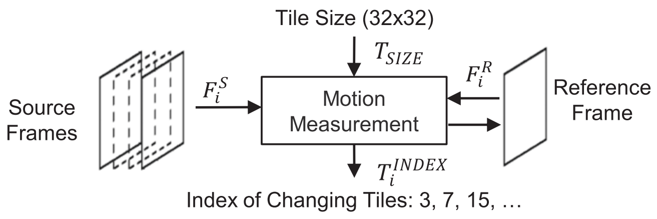

Fig. 9. Motion measurement inputs/output.

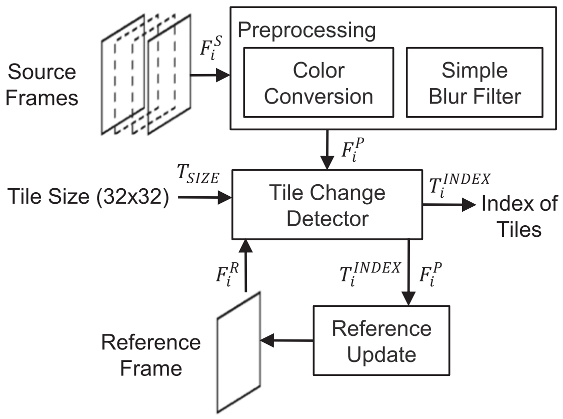

Fig. 10. Detailed MIJ2K motion measurement.

- Preprocessing: This task prepares the source frame F_i^S for motion measurement. The conversion performed is related to the extraction of image intensity information, that is, the Y-luminance component of the YUV color space. The conversion specified in (7) is applied in case of RGB images. This way we can work with one simple representation of the image, leading to a faster analysis of source frames.

$$Y = 0.2999R + 0.587G + 0.114B. \tag{7}$$

Once the source frame has been correctly transformed to a gray-scale image, a simple blur filter is used to reduce excessive detail and noise present in many surveillance cameras.

Fig. 11 shows the effects of the simple blur applied in the preprocessing subsystem for noise reduction. Fig. 11 a and b are two consecutive frames from the 'Surveillance' sequence (with the extracted Y component). Fig. 11c, and e represent the absolute difference (between frames n and $n + 1$) and binarization, where blur is not used in source frames. On the other hand, d and f represent the same frames but applying the blur filter to source frames. Notice that using simple blur in preprocessing can reduce noise, generally present in contours, and provide a clear binarization of the image. It hugely improves movement detection, discarding almost all noisy information.

Other techniques like erode and dilate are also useful for this task, but are computationally more complex. It is also harder to adjust the proper structuring element [27] to prevent too much information from being discarded.

Then, the preprocessed frame output by this subsystem, F_i^P, should be a gray-scale image conversion of F_i^S passed trough a simple blur filter, as described in (8).

$$F_i^P = Blur\left(GrayScale\left(F_i^S\right)\right). \tag{8}$$

- Tile change detector: This module, illustrated in Fig. 10, should detect the tiles that contain movement for the purpose of selective tile compression and transmission. The method used in this subsystem should be relatively simple in order to meet real-time requirements.

This subsystem compares the whole preprocessed frame F_i^P against the reference frame F_i^R tile by tile. It detects how much movement there is in each tile to decide whether or not it should be transmitted. Fig. 12 shows how this subsystem works. It is also described in detail in the following:

- Absolute difference $ABS_{i[x]}$: Each tile from both preprocessed and reference frames is passed through an absolute difference

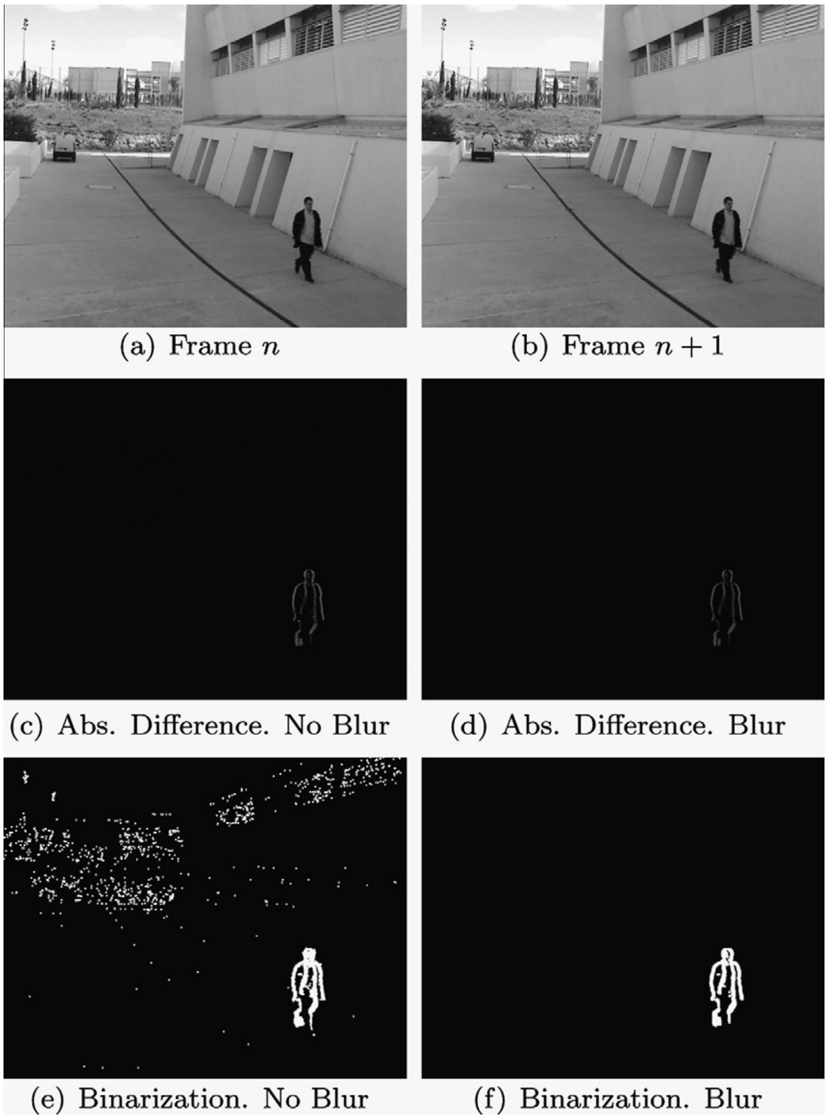
(a) Frame n
(b) Frame $n+1$
(c) Abs. Difference. No Blur
(d) Abs. Difference. Blur
(e) Binarization. No Blur
(f) Binarization. Blur

Fig. 11. Effects of simple blur on noise reduction.

filter to detect absolute changes between two tiles. The computational complexity of this operation a low, and it is useful for detecting objective differences between tiles. It generates a black-and-white image in which white pixels represent differences, whereas black pixels signify no changes. This is described in (9), where $F^P_{i[x]}$ and $F^R_{i[x]}$ are tile x of frame i in the preprocessed P and reference R frames. On the other hand, $ABS_{i[x]}$ represents the absolute difference between the tiles. In this case, both $F^P_{i[x]}$ and $F^R_{i[x]}$ are two matrices of $u \times v$ containing all the tile's pixel values.

$$ABS_{i[x]} = \left|F^P_{i[x]} - F^R_{i[x]}\right|. \tag{9}$$

The tile image $ABS_{i[x]}$ output by the absolute difference process should be analyzed in order to detect how much movement there is, and thus decide whether or not it should be transmitted. Changes should be measured somehow, and here we propose two efficient methods, which should work together.

- Mean value $MEAN_{i[x]}$: This is the first measurement. It is the mean pixel value of the $ABS_{i[x]}$ tile. This process takes all the values of the $ABS_{i[x]}$ tile, calculates the total and divides this by the number of tile pixels, as described in (10).

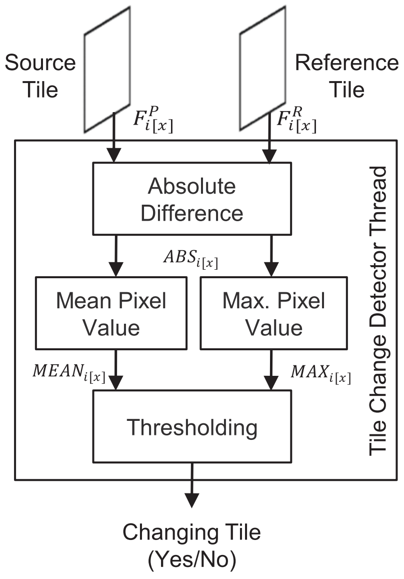

Fig. 12. MIJ2K tile change detector.

$$MEAN_{i[x]} = \frac{\sum_{s=0}^{u-1}\sum_{t=0}^{v-1} ABS_{i[x]\,(s,t)}}{u \times v}. \tag{10}$$

- Max value $MAX_{i[x]}$: The second measurement is the maximum pixel value of the $ABS_{i[x]}$ tile, as described in (11). This is useful for detecting movement peaks in tiles, since they could be overlooked by the mean value when nearby pixels values are near to zero.

$$\begin{aligned} &\forall s, t/s \geqslant 0 \wedge s < u, \quad t \geqslant 0 \wedge t < v \\ &MAX_{i[x]} = ABS_{i[x]\,(s,t)} \iff \\ &\neg\exists\ ABS_{i[x]\,(g,h)} > ABS_{i[x]\,(s,t)} \end{aligned} \tag{11}$$

Operating in conjunction, $MEAN_{i[x]}$ and $MAX_{i[x]}$ can detect all kinds of movements, ranging from small uniform variations (using the mean) to occasional big changes (using max threshold). Both metrics are easy to implement, providing a very low complexity method for detecting movement. This complexity will be analyzed in the test sections where we will compute the delay introduced.

- Thresholding: Both $MEAN_{i[x]}$ and $MAX_{i[x]}$ indicators together tell us when the tile should be transmitted. In this way, there is said to be a big enough change in a tile to warrant transmission when both values are above some threshold. The threshold values were optimized in [2]. This process should only compare the results of the processed values with the static thresholds established in the MIJ2K encoder. For example, tested values with good results are 2.0 for $MEAN_{i[x]}$ and 15 for $MAX_{i[x]}$.
- Reference update: This subsystem is used to update the reference frame F_i^R. The first reference frame, F_0^R, is an entirely black image, and it is updated frame by frame with the tiles that are different from preprocessed frame F_i^P. Logically, the first frame F_0^P will update all the tiles of the reference frame. The reference frame is also updated directly from the blurred image output by the preprocessing subsystem. This will stop the reference frame from having to be preprocessed at each time and reduce system complexity.

4.3. Packetization

This subsystem comes into operation once the motion measurement process has detected the tiles that should be transmitted, and the MIJ2K encoder has compressed the detected tiles only. The input of this subsystem is the output of the MI2JK encoder, that is, a F_i^{MIJ2K} code stream, containing tiles with detected movement only.

The packetization subsystem should parse the J2K code stream in order to perform an intelligent transmission. The transmission achieved is compliant with RFC 5371 [10]. The implementation is described in more detail in [3].

It basically takes the input J2K code stream and sends RTP packets in units of main header and tiles. In this case, RTP packets use a special payload defined for transmitting JPEG2000 sequences, as described in RFC 5371.

The transmission is performed as shown in Fig. 13. Each independent part of the J2K code stream should be transmitted in an independent RTP packet, that is, the first RTP packet should be the main header, followed by each independent tile of the frame.

Some benefits of this procedure are reported in [3], and briefly explained as follows:

- Parallel processing: The whole frame does not have to be compressed before it is sent. As soon as a tile is compressed, it can be sent. Thus, it is possible to compress and transmit at the same time, improving overall system performance. The same applies on the client side: it is possible to receive and decode tiles from the same frame at the same time.

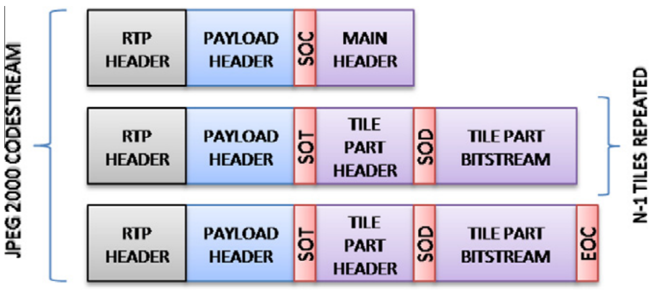

Fig. 13. RTP transmission.

- Error resilience: If a packet is lost during a tile transmission, the loss would affect this specific tile only rather than the whole frame.
- Main header recovery: Main header is the most important part of the code stream. Without the main header, decompression is impossible. To solve this problem, the system can use a main header identification (like a content-dependent hash encode) for recovery if the frame transmission is missing the main header. To do this, each RTP packet includes an integer value identifying the main header associated with the transmitted content. itemThanks to this ID, the client side can use a previous main header backup to recover a main header lost during transmission. This is feasible since all frames in a video sequence tend to share the same values for width, height, components, bits per pixel, number and size of tiles, etc., that is, the same main header. Exploiting this feature the main header does not always have to be sent, saving some bandwidth in transmissions.

4.4. Transmission/reception

These subsystems are involved in the transmission and reception of RTP packets. RTP packets should be transmitted over a transport protocol to define the source and destination address. We use the user datagram protocol (UDP). This protocol caters for real-time requirements, and avoids the overhead introduced by other transport protocols, such as the transmission control protocol (TCP). Moreover, UDP is not connection oriented, allowing the video sequences to be transmitted to many clients via broadcast and multicast.

Another important transmission and reception component is the transmission channel. It should provide a fast and reliable transmission and enough bandwidth for transmitting video signals. Nowadays, available Internet connections tend to provide enough bandwidth for this purpose, and the system could be used almost all over the world.

Although all of our tests have been performed in a 100 Mbps local area network (LAN), better LANs like Gigabit Ethernet or Fiber Optic could hugely improve the system, reducing latency and increasing the number of video channels.

4.5. JPEG2000 decoder

The MIJ2K architecture JPEG2000 Decoder subsystem should decode JPEG2000 tiles as soon as they are received. As we can see from the transmission subsystem, each part of the J2K code stream (main header and tiles) is transmitted in independent RTP packets. In this way, the decoder does not have to wait to receive the whole J2K code stream. The J2K decoder should decode the independent tiles upon reception, while the other tiles are still being received.

Fig. 14 shows this process. The first RTP packet received is the main header. The header is necessary to decode the rest of the code stream. The following RTP packets are tiles. Tiles can be decoded independently. Therefore, this subsystem should provide the decoded tile immediately, without waiting to receive the other tiles.

Notice that the potential for parallelism in decoding independent tiles. Many threads can be waiting for and decode incoming tiles simultaneously. This will speed up the decoding time, as the number of core processors increases.

4.6. Display/store decoded video

This subsystem displays and/or stores the decoded video sequences in the client side. This subsystem is quite straightforward, since its only task is to update the decoded $F^D_{i[x]}$ tiles in the displaying frame F^D_i. This frame can also be stored to keep a copy of the received sequence.

5. MIJ2K performance tests

This section details all the tests performed on the proposed MIJ2K architecture in order to evaluate its suitability. We evaluate MIJ2K in the following terms:

- Video quality: MIJ2K adds a pure basic inter-frame technique to the native JPEG2000 intra-frame compression system. So, we want to evaluate the improvements on the native JPEG2000 in terms of video quality. We will also evaluate other intra-frame compressors, like MJPEG or H.264-Intra.
- Added computational complexity: Since MIJ2K introduces some processing of live video frames, we want to find out how much time is spent on this task, and how this can affect the live video streaming system.
- Latency: This test should show the total latency of our implementation of MIJ2K in a real environment. We use our laboratory (VISLAB) to stream a live video sequence, and compute the total time from when the video is captured, compressed and transmitted by the server to when it is displayed on the client side. This will tell us if it is really suitable for real-time purposes.

For video quality evaluation we will use a standard set of videos, as described in Section 3. The selected sequences are ‘Akiyo’ and ‘Hall Monitor’. We also use the ‘Surveillance’ sequence.

All sequences used for this task are stored as uncompressed 24 bit RGB, without any audio channel. They are described in Table 1.

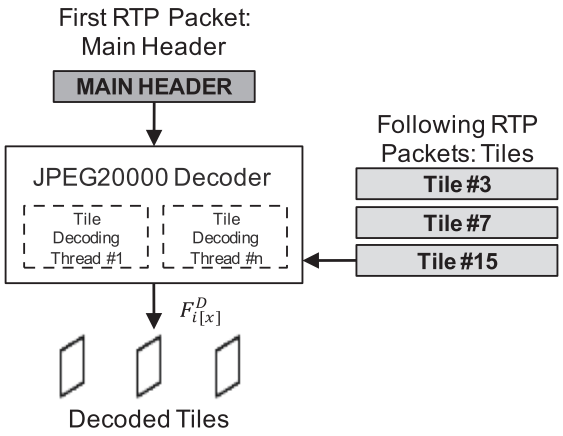

Fig. 14. MIJ2K decoder operation.

Table 1
Video sequences details.

Sequence	Resolution	Length	Original size
Akiyo	352 × 288	251 f, 10 s	72.8 MB
Hall Monitor	352 × 288	251 f, 10 s	72.8 MB
Surveillance	640 × 480	704 f, 28 s	622.3 MB

5.1. Comparison against native JPEG2000 streaming system

This test should evaluate the suitability of MIJ2K against traditional JPEG2000 sequences. We evaluate the quality of all output video sequences, compressed both either MIJ2K or standard JPEG2000. All video sequences have been compressed at approximately the same compression ratio (CR) in order to evaluate the quality output by both encoders. In order to achieve this tests, JPEG2000 encoder has been setup with a constant bit-rate compression without tiling. MIJ2K also uses constant bit-rate compression but in this case achieves a tiling compression. The bit-rate compression in both cases is not the same, since MIJ2K may use higher bit-rates (this is how it obtains better image quality) to achieve the same compression ratio provided by JPEG2000, as it takes advantage of the motion compensation technique.

The parameters used for MIJ2K algorithm are 32 × 32 for T_{SIZE}, 2.0 for $MEAN_{i[x]}$, and 15 for $MAX_{i[x]}$ as result of the optimization performed in [2]. The JPEG2000 encoder/decoder used for this tests is Kakadu JPEG2000 SDK. Results are summarized in Table 2.

Figs. 15–17 indicate that MIJ2K outperforms native JPEG2000 intra-frame compression in all sequences and especially the ‘Surveillance’ sequence. Figs. 15–17 show the frame-by-frame PSNR achieved by both compressors, and MIJ2K clearly outdoes the standard JPEG2000 video coder. Remember too that JPEG2000 sequences do not use tiling for frame compression; instead each frame is compressed in a single tile. This is an advantage for JPEG2000 sequences, since the compressor is more efficient, but it not is enough for it to outperform MIJ2K. Using the same tiling as MIJ2K, the results for JPEG2000 sequences would be even worse.

The extra PSNR achieved by MIJ2K compared with standard JPEG2000 appears to depend on the compressed sequence and especially the resolution. ‘Akiyo’ and ‘Hall Monitor’ share the same resolution, and both compression qualities are better with MIJ2K by around 5 dB (see Table 2 for average PSRN and CR). On the other hand, the ‘Surveillance’ sequence has a better resolution, which appears to increase the probability of non-moving areas (since the camera is fixed) and better tile reuse. Thanks to this, compression was better with MIJ2K than with traditional JPEG2000 by around 9 dB. So, MIJ2K can be expected to achieve higher performances at better resolutions.

Notice that the human eye can detect a variation of just 0.5 dB, and we are increasing this value by around 10 to 18 times. An example of the difference of quality is shown in Fig. 18, where the top image is the 550th frame of the ‘Surveillance’ sequence, compressed with JPEG2000 and the bottom image is the same frame compressed with MIJ2K. The contours are better defined in the MIJ2K frame, whereas the J2K frame appears to be blurred.

Let us also look at how the size of each frame varies depending on the movement detected in the scene. Fig. 19 compares frame size variation in the ‘Akiyo’ sequence. In the case of MIJ2K, the whole first frame has to be transmitted, leading to the initial peak. The following frames reuse the information of the first frame, saving around 80% of data with respect to the first frame. This will obviously depend on the movement in the scene, but we have found that on average it is around 80 to 95% in fixed cameras.

In the case of standard JPEG2000, all frames are encoded similarly, then a continuous bit-rate is output for the entire sequence. Note that the size of the first frame of MIJ2K (whole frame) and

Table 2
Results comparing the compression quality of the proposed MIJ2K architecture against a native J2K intra-frame codec.

Sequence	MIJ2K Size	J2K Size	MIJ2K CR	J2K CR	MIJ2K Avg. PSNR	J2K Avg. PSNR
Akiyo	1265 KB	1201 KB	59:1	62:1	41.2525 dB	36.6815 dB
Hall Monitor	2070 KB	2138 KB	36.01:1	34.86:1	39.9168 dB	34.9348 dB
Surveillance	2534 KB	2556 KB	251.46:1	249.302:1	39.5379 dB	30.9377 dB

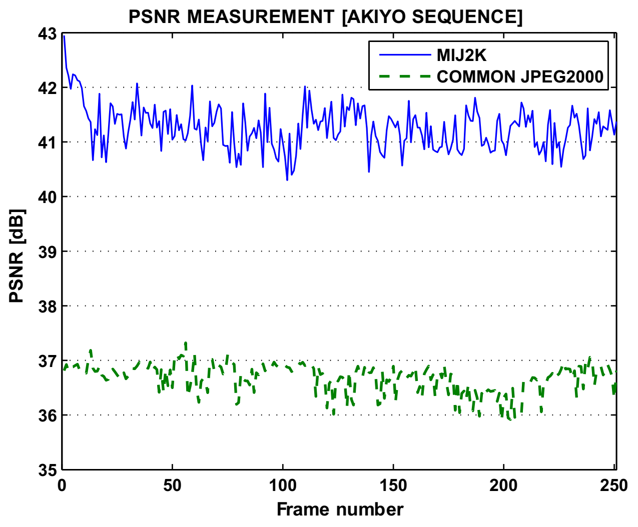

Fig. 15. Video quality of 'Akiyo' sequence.

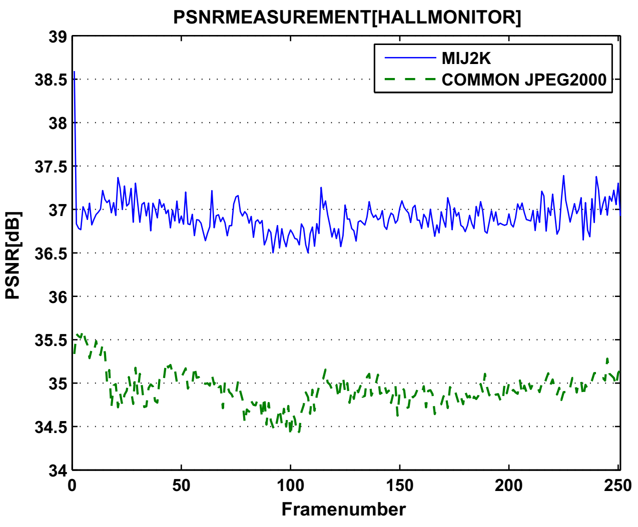

Fig. 16. Video quality of 'Hall Monitor' sequence.

first frame of JPEG2000 is very unalike. This is due to the different *bpp* qualities used in the two types of compression. MIJ2K can use better qualities, because it will use only some tiles of the image, and can compress these tiles with better qualities (this is why MIJ2K outperforms JPEG2000). On the other hand, JPEG2000 would have to use worse compression to maintain the set compression ratio, since it has to compress whole frames every time.

5.2. Motion measurement delay

This test should somehow measure the complexity of the motion measurement technique used in MIJ2K. Since there is no specific metric to evaluate this point, we are going to measure the

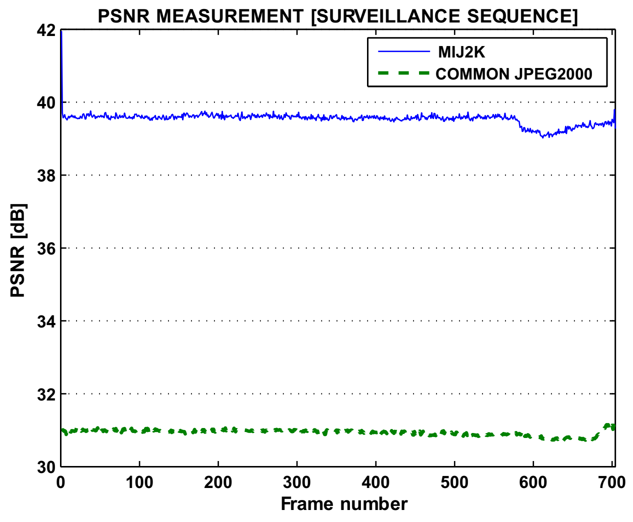

Fig. 17. Video quality of 'Surveillance' sequence.

total delay introduced by this motion measurement in the real-time transmission system. In other words, we will measure the time it takes to detect changing tiles. This includes all the operations performed by this subsystem, ranging from the preprocessing of source F_i^S images to the updating of reference frame F_i^R.

Almost all the operations that this subsystem performs on images, that is, gray-scale conversion, simple blur, absolute difference between source and reference frame and both max and mean pixel values have been implemented in C++ using the OpenCV vision library. Notice that at the time of writing, this subsystem had not yet been parallelized. We think that the use of multi-threading architectures could hugely improve these results.

The motion analysis system is an Intel Centrino2 Duo at 1.66 GHz, with 2.5 GB of DDR2 at 533 MHz. The operating system is 64 bit Windows Vista SP1 Business. As mentioned in the definition of the MIJ2K architecture, the size of tiles for detection is 32×32 pixels over 352×288 and 640×480 frames. To measure the code execution time, we have used the highly accurate Windows QueryPerformanceFrequency API.

Fig. 20 includes times measured in milliseconds for both resolutions, 352×288 and 640×480. It takes only about 6 ms to process 352×288 frames, whereas 640×480 frames are processed in around 19 ms. These are good values, since a live video source usually provides 25 frames per second (FPS), that is, there is a new frame every 40 ms. As a result, there is enough time to process the current frame before a new frame arrives, meaning that this system is suitable for real-time purposes. Using more capable hardware and implementing parallelization, it could also support higher resolutions.

Remember also that this system can reduce compression time, since it is not necessary to compress the whole frame. Therefore, combined with the compression system, this system could reduce the overall compression time (changing tile detection plus compression) to less than taken by a simple compression system compressing whole frames.

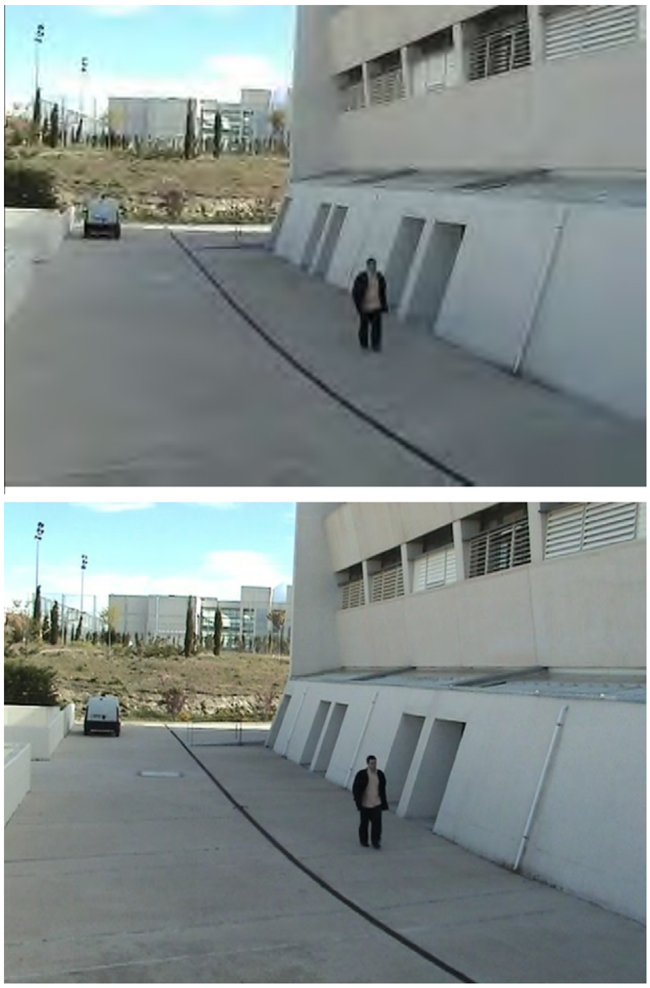

Fig. 18. Difference of quality between traditional J2K (top) to MIJ2K (bottom) at the same compression ratio for the 'Surveillance' sequence.

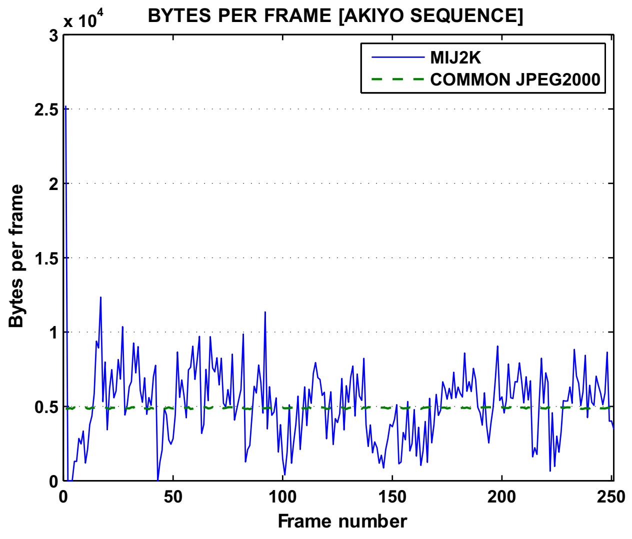

Fig. 19. Bytes per frame used in 'Akiyo' Sequence.

5.3. Total latency

The test performed in this section is related to the total amount of time that the streaming system takes from when the image is

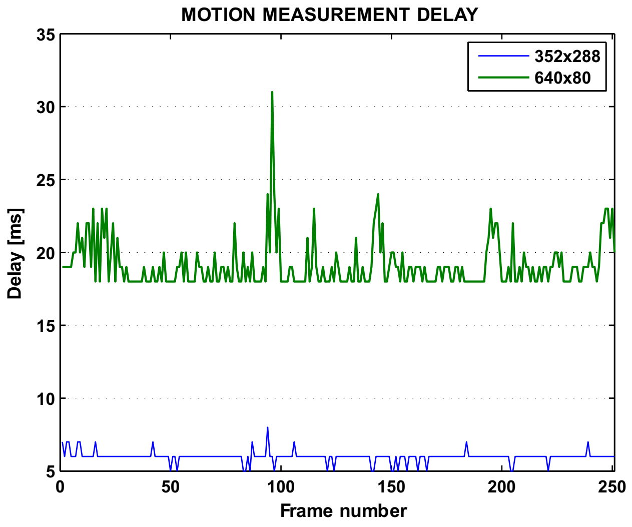

Fig. 20. Delay introduced by the motion measurement technique used in the MIJ2K architecture.

captured in the server to when the image is displayed in the client. In this case, we are evaluating more than the designed MIJ2K architecture. Consequently, this test is not fully representative of the proposed method, but it is useful for getting an idea of potential performance.

The systems used for this test are as follows:

- Server: Computer running Microsoft Windows XP Service Pack 2. Intel Core 2 Duo at 2.0 GHz and 2 GB of DDR2 at 533 MHz. It uses Matrox Morphis boards to acquire video and JPEG2000 compression. The streaming software is our own implementation of the RFC 5371 and the MIJ2K architecture, all developed in C++.
- Network: Network used in this test is a 100 Mbps Ethernet multicast transmission by the server. Client and server are connected by a single CISCO router running at the same speed.
- Client: The client hardware in this case is the same as for the motion measurement delay test. The client software used to receive and decode RTP streams is our own implementation developed in C++ with graphical interface developed using WxDevCpp. It complies with RFC 5371. JPEG200 decoding is performed in this case using Kakadu JPEG2000 SDK, developed by Dr. Taubman.

One way to evaluate how long the acquisition, compression, transmission, decoding and displaying processes take is to synchronize two computers (client and server) at exactly the same time. Then, the server can use timestamps to indicate when the process began. Once the frame with this timestamp has been received and decoded on the client side, the client can subtract the current time from the timestamp received and find out the total process time.

But it is quite a challenge to synchronize the clock of two computers with millisecond precision. So, we chose an easier option that provides the same results. This method displays a clock on a computer screen, which is focused on the camera used for streaming. The streaming video received is displayed on the same screen, and two clocks will appear, the 'local' clock, and the 'remote' clock. The difference between these two clocks will give the total time that MIJ2K takes to transmit the sequence.

The screenshot in Fig. 21 shows the result of this test. The time taken in this case to perform the live video streaming is about

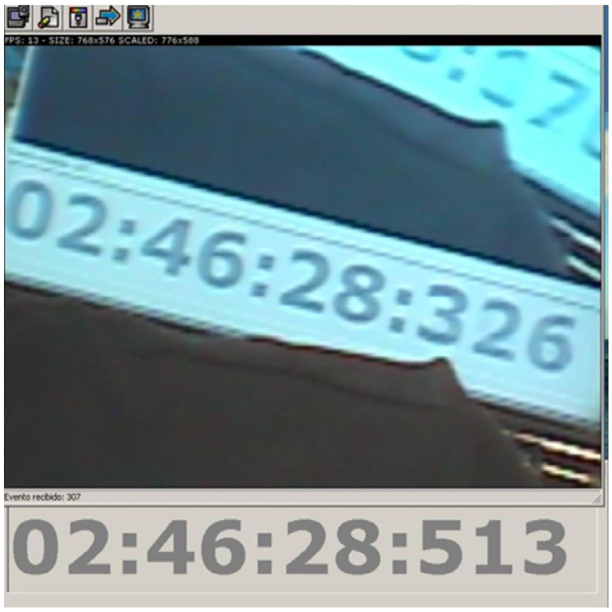

Fig. 21. Total delay in the image obtained with the MIJ2K video streaming architecture. It takes around 187 ms to perform the acquisition, compression, transmission, decoding and displaying processes.

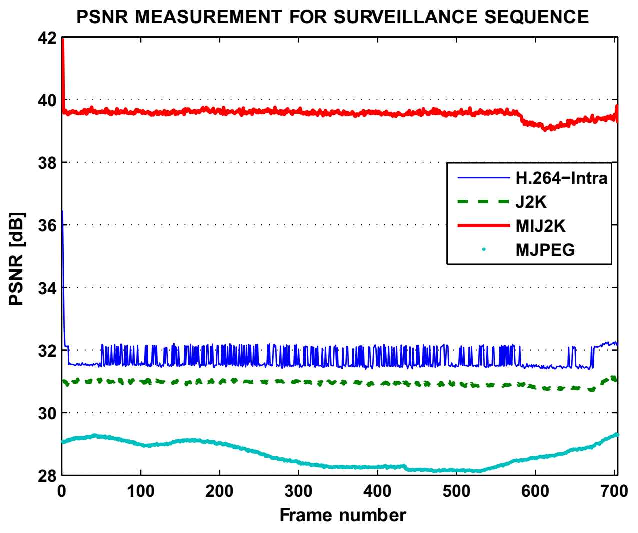

Fig. 22. PSNR measurement for H.264-Intra, J2K, MIJ2K and MJPEG for the 'Surveillance' sequence.

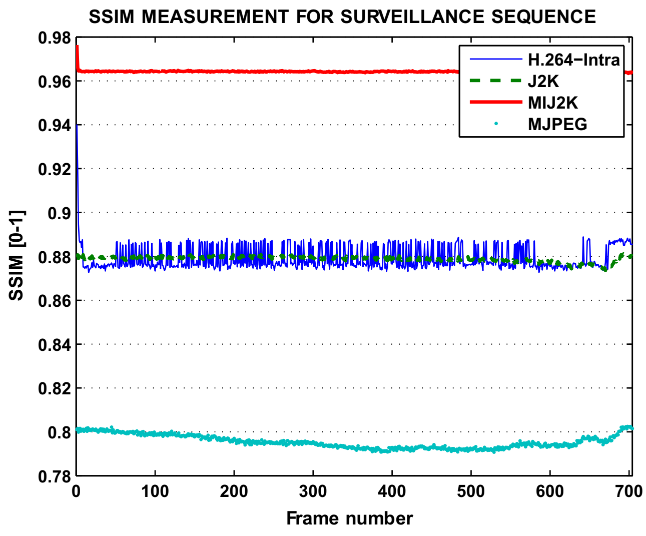

Fig. 23. SSIM measurement for H.264-Intra, J2K, MIJ2K and MJPEG for the 'Surveillance' sequence.

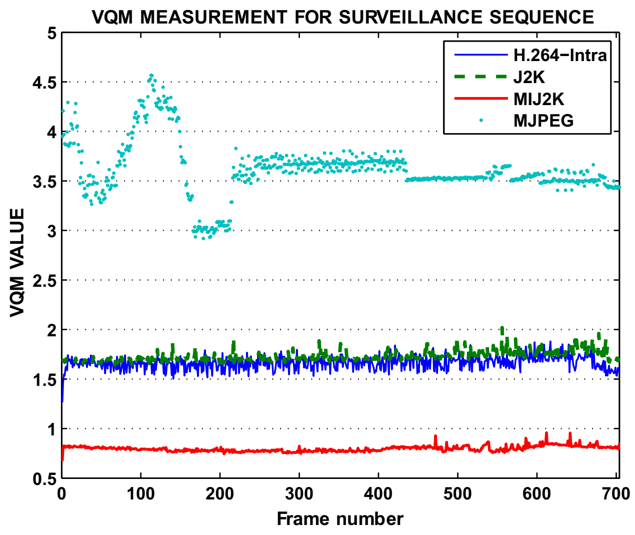

Fig. 24. VQM Measurement for H.264-Intra, J2K, MIJ2K and MJPEG for the 'Surveillance' sequence.

187 ms, which is adequate for real-time purposes, as discussed in [34].

5.4. Comparison with other intra-frame codecs

This last test should evaluate MIJ2K performance with respect to other intra-frame codecs. In Section 5.1 MIJ2K was compared with a native JPEG2000 system. In this case, we will also use the MJPEG codec and H.264-Intra specification. The use of H.264-Intra instead of standard H.264 is due to we should compare the codecs in the same conditions of low latency and complexity, as MIJ2K provides. Standard H.264 achieves more complexity in motion detection and the compression latency is not suitable for real-time transmissions.

The quality measurements will be taken using the PSNR, VQM, and SSIM metrics on the 'Surveillance' video sequence. We use the Surveillance video because it is the most representative sequence, as it is captured from a real surveillance camera. The resolution is also more representative of today's cameras. This evaluation will cover both objective and subjective metric measurements.

Figs. 22–24 illustrate the results of the test performed in this section. We can see how MIJ2K is better than the other compression systems in all cases. The output PSNR is around 8 dBs better than the second-best codec, H.264-intra. The worst compression achieved in this case was for the MJPEG codec, which has a mean PSNR of 28 dB, around 11 dBs less than MIJ2K. Notice also in Table 3 that the MJPEG compression ratio is worse, since it cannot achieve a better image compression.

The SSIM test shows how MIJ2K achieves better results for this subjective quality metric. The quality results for JPEG2000 and H.264-intra are more or less the same, but H.264-intra looks to be slightly better. Again, MJPEG achieves poor results for SSIM metric.

The last test performed is related to the VQM subjective quality metric. Note how MJPEG scores the highest values (worst results),

Table 3
Results comparing the compression quality of the 'Surveillance' sequence. Codecs used are H.264-Intra, JPEG2000, MIJ2K, and MJPEG.

Codec	Video size	Compression ratio	PSNR	SSIM	VQM
H.264-Intra	2567 KB	248.23:1	31.7048 dB	0.87921	1.66347
JPEG2000	2556 KB	249.30:1	30.9377 dB	0.87876	1.72719
MIJ2K	2534 KB	251.46:1	39.5379 dB	0.96434	0.79581
MJPEG	8094 KB	78.72:1	28.6561 dB	0.79552	3.61294

whereas the qualities for JPEG2000 and H.264-intra performance are almost the same. MIJ2K again achieves the best compression results with values nearer to 0.8 (better results).

So, we can definitely say that MIJ2K outperforms all tested intra-frame compressors. This is good news, taking into count that MIJ2K compresses in near real-time as if it were a pure intra-frame codec. Potential compression time is even lower, since it does not compress the whole frame.

6. Conclusions

In this paper, we have fully described the patent pending codec MIJ2K. It is based on the introduction of a new real-time motion detection system for JPEG2000 sequences, using the JPEG2000 code-stream tiling concept. It reduces the bandwidth necessary for transmission, improving the overall quality of sequences. Basically it involves detecting and transmitting the tiles that change in each frame. We have demonstrated that MIJ2K is suitable for use in real-time environments, like video surveillance, where latency is a critical point. We have also compared the MIJ2K codec with other intra-frame codecs, like standard JPEG2000 sequences, Motion JPEG, and H.264-Intra. All tests show that MIJ2K outputs better qualities (around 9 dBs at best) at the same compression ratio.

MIJ2K sequences can also be transmitted with the standard RTP transmission payload for JPEG2000 images described in RFC 5371 [10]. So, any standard RFC-capable client could benefit from MIJ2K built into the server without any modification whatsoever.

Acknowledgments

This work was supported in part by Projects CICYT TIN2008–06742-C02–02/TSI, CICYT TEC2008–06732-C02–02/TEC, SINPROB, CAM MADRINET S-0505/TIC/0255 and DPS2008–07029-C02–02.

References

[1] M. Adams, R. Ward, Wavelet transforms in the JPEG-2000 standard, in: 2001 IEEE Pacific Rim Conference on Communications, Comput. signal Proces., 2001. PACRIM, vol. 1, 2001.
[2] A.L. Bustamante, J.M. Molina, M.A. Patricio, Video encoder optimization via multiobjective evolutionary algorithms, Proceedings of the 11th Annual conference on Genetic and evolutionary computation. GECCO '2009, 2009, pp. 1835–1836.
[3] A.L. Bustamante, M.A. Patricio, Scalable Streaming of JPEG 2000 Live Video Using RTP over UDP, International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008), 50/2009 (2008) 574–581.
[4] M. Charrier, D. Cruz, M. Larsson, JPEG2000, the next millennium compression standard for still images, in: IEEE International Conference on Multimedia Computing and Systems, 1999, vol. 1, 1999.
[5] C. Christopoulos, A. Skodras, T. Ebrahimi, The JPEG2000 still image coding system: an overview, IEEE Trans. Consumer Electron. 46 (4) (2000) 1103–1127.
[6] F. Devaux, J. Meessen, C. Parisot, J. Delaigle, B. Macq, C. De Vleeschouwer, A flexible video transmission system based on JPEG 2000 conditional replenishment with multiple references, in: IEEE International Conference on Acoustics, Speech Signal Proces. (ICASSP 07).
[7] L. Digital Cinema Initiatives, Digital cinema system specification version 1.2. March 07, 2008.
[8] S. Fossel, G. Fottinger, J. Mohr, Motion JPEG2000 for high quality video systems, IEEE Trans. Consum. Electron. 49 (4) (2003) 787–791.
[9] T. Fukuhara, K. Katoh, S. Kimura, K. Hosaka, A. Leung, Motion-JPEG2000 standardization and target market, in: Image Processing, 2000. Proceedings. 2000 International Conference on, vol. 2, 2000.
[10] S. Futemma, E. Itakura, A. Leung, RTP Payload Format for JPEG 2000 Video Streams, Technical Report 5371, Oct. 2008.
[11] B. Girod, What's wrong with mean-squared error? (1993)
[12] B. Haskell, A. Puri, A. Netravali, Digital video: An introduction to MPEG-2, Kluwer Academic Publishers., 1996.
[13] ISO/IEC. 15444-1:2000 information technology JPEG2000 image coding system-part 1: core coding system, Technical report, ISO/IEC, 2000.
[14] ISO/IEC, 15444-3:2007 information technology JPEG2000 image coding system-part 3: Motion jpeg 2000, Technical report, ISO/IEC, 2007.
[15] M. Lora, Xiph.org:: Test media, World Wide Web electronic publication, 1994–2008.
[16] D. Marpe, V. George, H. Cycon, K. Barthel, Performance evaluation of Motion-JPEG2000 in comparison with H. 264/AVC operated in pure intra coding mode, in: Proceedings of SPIE, vol. 5266, 2004, pp. 129–137.
[17] J. Meessen, C. Parisot, X. Desurmont, J. Delaigle, Scene analysis for reducing motion JPEG 2000 video surveillance delivery bandwidth and complexity, in: IEEE International Conference on Image Processing, 2005. ICIP 2005, vol. 1, 2005.
[18] A. Naman, D. Taubman, A novel paradigm for optimized scalable video transmission based on JPEG2000 with motion, in: IEEE International Conference on Image Processing, 2007. ICIP 2007, vol. 5, 2007.
[19] A. Naman, D. Taubman, Optimized scalable video transmission based on conditional replenishment of JPEG2000 code-blocks with motion compensation, in: Proceedings of the International Workshop on Mobile Video, ACM, New York, NY, USA, 2007, pp. 43–48.
[20] W. Pennebaker, J. Mitchell, JPEG Still Image Data Compression Standard, Kluwer Academic Publishers., 1993.
[21] F. Pereira, T. Ebrahimi, The MPEG-4 Book, Prentice Hall, PTR Upper Saddle River, NJ, USA, 2002.
[22] D. Santa-Cruz, T. Ebrahimi, An analytical study of JPEG 2000 functionalities, in: Image Processing, 2000. Proceedings. 2000 International Conference on, vol. 2, 2000.
[23] D. Santa-Cruz, T. Ebrahimi, J. Askelof, M. Larsson, C. Christopoulos, JPEG 2000 still image coding versus other standards, in: PROC SPIE INT SOC OPT ENG, vol. 4115, 2000, pp. 446–454.
[24] Y. Shi, H. Sun, Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards, CRC press, 2000.
[25] D. Shirai, T. Yamaguchi, T. Shimizu, T. Murooka, T. Fujii, 4K SHD real-time video streaming system with JPEG 2000 parallel codec, in: IEEE Asia Pacific Conference on Circuits and Systems, 2006. APCCAS 2006, 2006, pp. 1855–1858.
[26] A. Skodras, C. Christopoulos, T. Ebrahimi, JPEG2000: The upcoming still image compression standard, Pattern Recognit. Lett. 22 (12) (2001) 1337–1345.
[27] M. Van Droogenbroeck, H. Talbot, Fast computation of morphological operations with arbitrary structuring elements, Pattern Recognit. Lett. 17 (14) (1996) 1451–1460.
[28] G. Wallace et al., The JPEG still picture compression standard, Commun. ACM 34 (4) (1991) 30–44.
[29] Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli, Image quality assessment: From error visibility to structural similarity, IEEE Trans. Image Proces. 13 (4) (2004) 600–612.
[30] A. Watson, Toward a perceptual video quality metric, in: Proc. SPIE, vol. 3299, 1998, pp. 139–147.
[31] A. Watson, J. Hu, J. McGowan III, Digital video quality metric based on human vision, J. Electron. Imag. 10 (2001) 20.
[32] F. Xiao. DCT-based video quality evaluation. Final Project for EE392J, 2000.
[33] K. Varma, A. Bell, JPEG2000-choices and tradeoffs for encoders, IEEE Sign. Proces. Mag. 21 (6) (2004) 70–75.
[34] G. Karlsson, Asynchronous transfer of video, IEEE Commun. Mag. 34 (8) (1996) 118–126.