Multimed Tools Appl (2013) 64:731–755
DOI 10.1007/s11042-011-0970-3

Multi-view codec with low-complexity encoding for Distributed Video Coding

Lucian Ciobanu · Luís Côrte-Real

Published online: 6 January 2012

Abstract The low-complexity encoding, as fundamental requirement of Distributed Video Coding, relies on performing the bulk of computation at decoder, including tasks as the generation of side information and particularly, inter-camera registration in the case of multi-view systems with complete-overlapped views and free motion of the cameras (e.g., video surveillance). In Ciobanu and Corte-Real (Multimedia Tools Appl 48(3):411–436, 2010) we introduced a codec-independent solution for such tasks at decoder. In this paper, we present a multi-view Wyner–Ziv codec (IWZ) designed for the architecture and scenarios from Ciobanu and Corte-Real (2010) (e.g., free motion of the cameras, no a priori knowledge of the instant camera positions, no feedback channel), based on transform domain (DCT), block-based coset coding. We aimed to achieve a compromise between the low encoder complexity and the rate-distortion performance. A detailed evaluation is presented for comparison with conventional coding (Intra 4×4 and Intra 16×16). Practical results show a better overall performance of the proposed codec at low bitrates.

Keywords Distributed Video Coding (DVC) · Distributed Source Coding (DSC) · Wyner–Ziv (WZ) · SSIM (Structural SIMilarity) · Intra 4×4 · Intra 16×16 · Discrete Cosine Transform (DCT) · CAVLC (Context-Adaptive Variable-Length Coding) · Huffman Coding · Coset Coding

1 Introduction

Distributed Video Coding (DVC), as a particular paradigm of Distributed Source Coding (DSC), has been one of the most active research areas in the signal processing

L. Ciobanu (✉) · L. Côrte-Real
Faculdade de Engenharia da Universidade do Porto/INESC Porto, Porto, Portugal
e-mail: lciobanu@inescporto.pt

L. Côrte-Real
e-mail: lreal@inescporto.pt

community in the last years, providing a revolutionary new perspective over the conventional video compression (e.g., the MPEGx, H.26x families). It emerged in the favourable context of increasing distributed architectures, due to the technical advances from the last decade, enabling the deployment of cheap, low-power sensing devices widely spread over large areas, from hand-held digital cameras to the omnipresent multimedia cellular phones.

The roots of Distributed Source Coding date back to the 1970s when the information-theoretic results of Slepian–Wolf in 1973 for lossless coding with side information at decoder side [21], and then in 1976 extended by Wyner–Ziv for lossy coding [27], have shown the conceptual importance of this distributed paradigm. As stated in theory, the Distributed Source Coding enables the same coding efficiency for architectures with independent encoders (no communication between each other) and joint decoding as in the case the encoders are jointly encoding. Consequently, when applied to Distributed Video Coding, it is an essentially reversed paradigm that enables the shift of the bulk of computation from encoder to decoder, as opposed to the conventional (e.g., H.26x, MPEGx), non-distributed coding.

Presently, most of the techniques rely on the channel coding principles due to their relationship with the Slepian–Wolf coding. As indicated in [9], there can be transmitted only the parity bits of one binary sequence X based on the statistical dependence between X and its noisy version (Y). At decoder it can be used the side information Y jointly with the correlation model to successfully decode the initial source X and therefore, it can be performed a channel coding regarded as Slepian–Wolf coding. An equivalent approach is to divide the alphabet of X into cosets, the encoder then sends the syndrome (index of the coset that X belongs to) and the decoder decides upon the most probable guess among the codewords in that respective coset by comparing them to Y. In this approach the syndromes can be seen also as parity bits, showing the relationship of this method with the channel coding principles [9]. A brief introduction to the main distributed source coding techniques is made in [9].

The work on the Distributed Source Coding problem was started by Pradhan and Ramchandran in 1999, since then more improved channel coding techniques were developed based on iterative channel decoding, most of them using turbo codes [5, 8, 13, 24, 29]. Other works rely on Low-Density Parity-Check (LDPC) codes as a powerful alternative to turbo codes for Distributed Source Coding [14, 23, 25, 26, 28], some authors suggest that in certain circumstances they might achieve better results than turbo codes. Either way, such state of the art codecs can come close to the Slepian–Wolf bound in lossless Distributed Source Coding. Some reference codecs from the literature are presented next.

The PRISM codec, as introduced in [18, 19], performs a block-based coset coding. It first classifies the correlation noise structure for each block thus contributing to the formation of the temporal predictors, a two-dimensional Discrete Cosine Transform (DCT) is then applied to the current block, the DCT coefficients are subsequently quantized with a step size proportional with the standard deviation of the correlation model. The syndrome encoding is performed, the space of quantized codewords being partitioned using trellis codes. A refinement quantization stage takes place as part of the encoding process in order to reach a target reconstruction quality by further re-quantizing the coefficients. Finally, the encoder sends a Cyclic Redundancy Checksum (CRC) check of the quantized sequence in order to help

the decoder choose the right decoded sequence when searching over the space of candidate predictors. One CRC match indicates the successful decoding.

Girod et al. presented in [1] a pixel-domain Wyner–Ziv codec based on turbo codes operating on the whole frame. The odd frames (X_{2i+1}), considered key frames, are not coded and are assumed to be perfectly known at decoder. The even frames (X_{2i}) are independently coded as follows: the frame is scanned line by line and each pixel value is quantized using 2^M levels, then the resulted symbols are fed into the turbo encoder and the generated parity bits are stored in a buffer. A subset of these parity bits are sent to decoder upon request. At decoder, the side information for each frame to be decoded is achieved by temporal interpolation between the two adjacent key-frames (X_{2i-1} and X_{2i+1}). This is used jointly with the received parity bits sub-set in order to decode the frame. If the decoder cannot reliably decode the symbols it requests more parity bits from encoder. When an acceptable probability of symbol-error is met the decoding is considered successful.

Guo et al. propose in [10] a generic architecture for multi-view Distributed Video Coding where the video is captured as a two-dimensional image matrix. The scheme was improved later in [11]. Within the predefined multi-view system each frame can be encoded as either a conventional Intra-frame (I frame) or as a Wyner–Ziv frame. The authors use the basic idea proposed in [2] in which the Wyner–Ziv frame is encoded using turbo codes and decoded with side information generated from the reference frames and furthermore, they propose a more flexible algorithm for side information generation based on both temporal and view-directional corrections in order to achieve high prediction accuracy. Wavelet transform is additionally employed on the Wyner–Ziv frame coding for exploiting the spatial correlation and at the same time to benefit from the high-order statistical correlation.

More references about DVC codecs can be found in [3, 4, 6, 17, 22].

Unlike other multi-camera codec approaches from the literature [10, 11, 15, 16], in this paper we present a multi-view Wyner–Ziv codec which doesn't require any specific camera arrangement, i.e., it enables the free motion of the cameras in the scene and requires no a priori knowledge of the instant camera positions. It is designed for real-life multi-camera environments (e.g., video surveillance) and the scenarios described in [7] (e.g., complete-overlapped views).

In our previous work [7] we introduced a simplified two-view scenario with one moving camera (the *target camera*), as illustrated in Fig. 1. The *reference camera* performs conventional encoding (e.g., H.26x, MPEGx) of its perceived view (the "scene" view) which is used by decoder to provide the side information, while the target camera conventionally encodes only the first frame and applies Wyner–Ziv encoding for the remaining frames. This paper is focused on the Wyner–Ziv coding from the target camera (starting with the second frame).

The codec was developed at INESC Porto and in this paper it is referred as IWZ (INESC's Wyner–Ziv codec). It relies on transform domain (DCT), block-based coset coding. It also requires an offline training stage with low-processing.

We aimed to achieve a compromise between the low-complexity encoding and the rate-distortion performance when compared with the conventional coding (Intra 4×4 and Intra 16×16). Practical results show a better overall performance of the proposed codec at low bitrates. Moreover, the rate-distortion performance increases significantly in scenarios with slow (target) camera movement, specific to the considered environments like video surveillance (see Section 4).

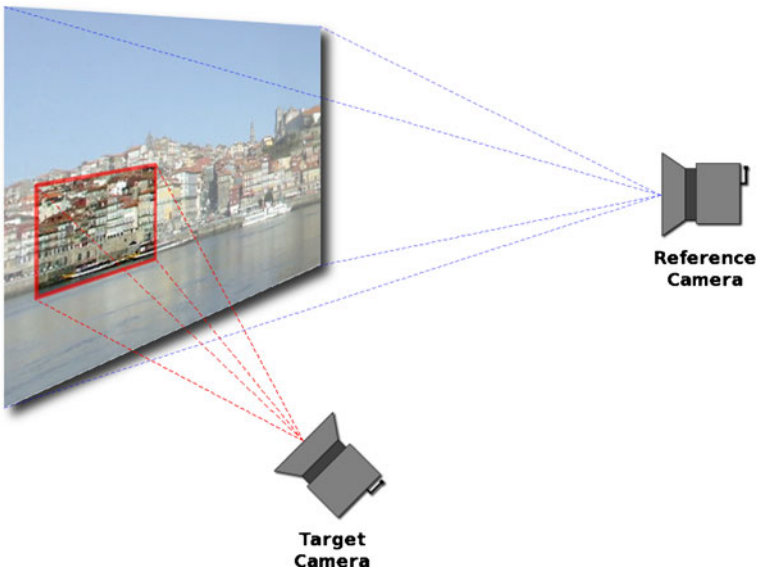


Fig. 1 Two-camera scenario with complete-overlapped views

Section 2 contains a detailed description of the IWZ codec. A methodology for evaluation of the encoder complexity is proposed in Section 3. The achieved results are presented in Section 4. Finally, the conclusions are drawn in Section 5.

2 Multi-view Wyner–Ziv codec (IWZ)

The architecture of the IWZ encoder is shown in Fig. 2. Each captured frame (in YUV 420 format) from the target camera is divided in equal blocks of predefined size (e.g., 8×8), i.e., the same block size is used for dividing both the luma and the two chroma components. Each block, in the raster-scan order within the current component, is individually managed by the same processing sequence, as described next.

All the DCT coefficients for the current block are determined, i.e., $B_{WH}{}^2$ coefficients, where B_{WH} is the predefined value of the block width and block height (e.g., 64 coefficients for 8×8 block size, $B_{WH} = 8$). In the implementation of the IWZ codec we reused the code of the two-dimensional integer DCT from [12]. Consequently, only two block sizes are permitted: 8×8 and 4×4. In this paper we used

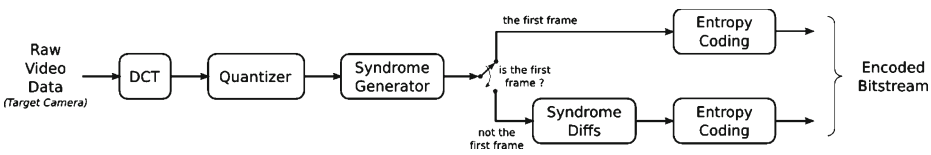


Fig. 2 Architecture of the IWZ encoder

for evaluation purposes the 8×8 block size (see Section 4). The DCT coefficients are subsequently rearranged into a one-dimensional array by the zig-zag scan specified in [20], sections 8.5.6 and 8.5.7, for 4×4 and 8×8 integer DCT, respectively. The first (low-frequency) K coefficients are further on processed in the order given by the array, i.e., from the DC (zero-frequency) coefficient to the highest frequency AC coefficient. The trailing coefficients up to ${B_{WH}}^2$ are discarded. The *number of DCT coefficients* to be processed (K) is predefined (e.g., 32 for 8×8 block size). These steps are performed by the *DCT* module (see Fig. 2).

All K coefficients are uniformly quantized by a predefined value (e.g., 1000), called *quantization levels* (QL), in the *Quantizer* module (see Fig. 2). For each quantized coefficient (CC) is generated the corresponding syndrome (Syn) in the *Syndrome Generator* module, as follows: $Syn = CC \ \% \ SL$, where SL is a predefined parameter called *syndrome levels* (e.g., 64) and % is the modulo operation (the remainder, on division of CC by SL).

Therefore, an integer array of syndromes is initially generated for each frame. The number of syndromes (N_S) from array is equal for each frame, as follows:

$$N_S = 1.5 \frac{FW}{B_{WH}} \frac{FH}{B_{WH}} K \tag{1}$$

where FW and FH represent the predefined *frame width* and *frame height*, e.g., $N_S = 57{,}600$ syndromes for 320×240 frame size, 8×8 block size and 32 DCT coefficients per block. In Section 4 we used 320×240 frame size for evaluation of the IWZ codec and the YUV 420 format for the raw video sequences. Consequently, the total number of blocks for a frame is 1.5 times bigger than the number of blocks for the luma component (see (1)).

The arrangement of syndromes within the array is given (in this order) by: (1) the Y-U-V order of the frame components, (2) the raster-scan order of blocks within each component and finally, (3) the order of the zig-zag scan of DCT coefficients for a block.

The array of syndromes is finally entropy coded and the resulted bitstream is sent to decoder (see the two instances of *Entropy Coding* module from Fig. 2). For higher efficiency of the codec, starting with the second frame, only the syndrome differences are entropy coded (see the *Syndrome Diffs* module), i.e., $Syn(i) = Syn(i) - Syn'(i)$, $i = \overline{1, N_S}$, where Syn is the array of syndromes for the current frame and Syn' is the array for the previous frame. At decoder, these syndromes are restored in the *Syndrome Reconstruction* module (see Fig. 3).

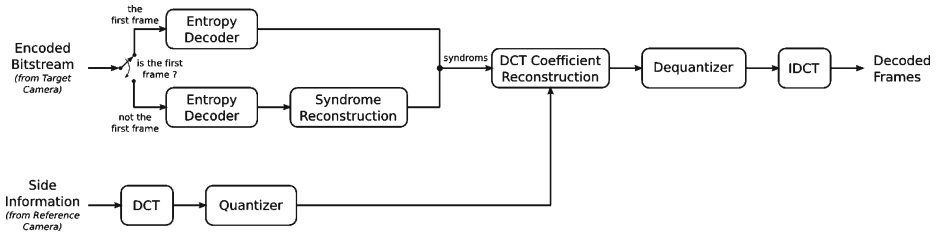


Fig. 3 Architecture of the IWZ decoder

The entropy coding algorithm for one frame is depicted next.

```
Begin algorithm
    Read the array of syndromes, the Huffman table for
        syndromes (the syndrome table) and the Huffman table
        for zero-runs (the zero-runs table)

    Prepare an empty bit array representing the bit coding
        of the current frame
    Select the first (leftmost) element (from the array
        of syndromes) as the pivot element
    While pivot has not reached the last (rightmost)
        position yet
        Get the Huffman code corresponding to the value of
            the current element (from the syndrome table);
            add this bit sequence (code) to the bit array
        Count the number of trailing zeros (NTZ) succeeding
            this element (the zero-run for this syndrome);
            get the Huffman code corresponding to NTZ
            (from the zero-runs table); add this bit
            sequence (code) to the bit array
        Move the pivot (rightwards) to the next non-zero
            element (after skipping NTZ elements)
    Endwhile

    Send the bit array to decoder; it represents the
        Wyner-Ziv encoding of the current frame
End algorithm
```

The bitstream sent to decoder comprises an alternation between the Huffman code of a non-zero syndrome and the Huffman code of the number of trailing zeros succeeding it (the length of the zero-run). Therefore, the entropy coding algorithm uses two Huffman tables which are previously generated in an offline training stage by two tools called *Syndrome Statistics* and *Zero-Runs Statistics*. Nevertheless, they perform a low-complexity processing comparable with the complexity of the encoder (see Section 4). The architectures of the two offline tools are shown in Figs. 4 and 5.

The two tools have the same architecture as the encoder (see Figs. 4 and 5), except for the entropy coding which is replaced by the generation of the Huffman tree (performed by the *Huffman* module). The additional module *Zero-Runs Generator* from Fig. 5 counts the number of trailing zeros after each non-zero element. Each

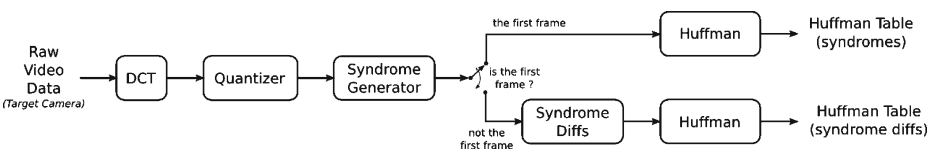


Fig. 4 Architecture of the *Syndrome Statistics* offline tool

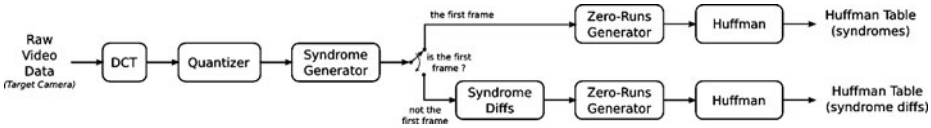


Fig. 5 Architecture of the *Zero-Runs Statistics* offline tool

offline tool generates two Huffman tables, one for syndromes and one for syndrome differences. Therefore, four Huffman tables are generated in the offline training stage. All four tables will be used by both the encoder and decoder. Examples of Huffman tables are shown in Tables 1, 2, 3 and 4.

On encoding are used simultaneously two Huffman tables at a time, depending on the frame number, i.e., for the first frame are used the Huffman tables for syndromes (Tables 1 and 3) and for the remaining frames are used the tables for syndrome differences (Tables 2 and 4). See the two *Entropy Coding* instances from Fig. 2.

In Fig. 3 is illustrated the architecture of the decoder. The contained modules perform the inverse operations corresponding to those described for encoder (e.g., the *Syndrome Reconstruction* module simply adds the received difference to the previous syndrome, in order to restore the current syndrome). The same order of the syndromes is used (as for encoder). For each block, the remaining DCT coefficients up to ${B_{WH}}^2$ are set to 0 (zero). Additionally, the decoder also performs a parallel processing for generating the quantized DCT coefficients for the side information frames (the side information based on the reference frames, as introduced in [7]), in synchronization with the target frames. This parallel processing is identical with the one performed by encoder up to (not including) the generation of the syndromes (see Fig. 2).

For each received syndrome from the target camera, the decoder tries to estimate the most probable value of the respective (quantized) DCT coefficient. To this end,

Table 1 Example of Huffman table for syndromes

Syndrome	Huffman code
−5	11010000010
−4	1101000000
−3	11010001
−2	11011
−1	00
1	01
2	11001
3	110001
4	110101
5	1110
6	1011
7	100
8	1010
9	1111
10	110000
11	1101001
12	110100001
13	11010000011

Table 2 Example of Huffman table for syndrome differences

Syndrome difference	Huffman code
−9	0101010100000111
−8	0101010100000110
−7	0101010100000100
−6	010101010001
−5	010101011
−4	0101011
−3	01011
−2	0100
−1	00
0	01010101001
1	1
2	011
3	010100
4	01010100
5	0101010101
6	0101010100000
7	0101010100000101

Table 3 Example of Huffman table for zero-runs based on syndromes

Zero-run length	Huffman code
0	01
1	001
2	1100
3	1011
4	1111
5	10001
6	11011
7	100110
8	101000
9	11101
10	100100
11	11010
12	1000011
13	10000101
14	10000100
15	1001110
16	101001
17	10000001
18	10000010
19	100101111
20	1001111
21	1001010
22	101010
23	100101110
24	100000000
25	10010110
26	100000001
27	1010110
28	1010111
29	11100
30	10000011
31	000

Table 4 Example of Huffman table for zero-runs based on syndrome differences

Zero-run length	Huffman code
0	00
1	011
2	0100
3	0101
4	1010
5	1110
6	10010
7	10110
8	11011
9	11000
10	11110
11	11010
12	100010
13	110011
14	1111101
15	1100100
16	1000000
17	1000001
18	1011100
19	10001101
20	1011101
21	1011110
22	1001110
23	10000111
24	10011010
25	10000101
26	10001110
27	10000100
28	10011111
29	1100101
30	1111110
...	...

the decoder uses as reference the corresponding DCT coefficient from the reference camera, i.e., from the same frame number, YUV component, block position and DCT coefficient index. This decision process is performed by the *DCT Coefficient Reconstruction* module (see Fig. 3), as follows.

For a given DCT coefficient from the reference camera (C_R), the decoder determines the nearest candidate to C_R, among all the candidates whose remainders on division by SL are equal to the received syndrome from the target camera (Syn). The corresponding implementation of the reconstruction method is detailed as follows. A variable Q is initially computed as the integer result of the division of C_R by SL.

$$Q = \left[\frac{C_R}{SL}\right] \tag{2}$$

Then are determined the two nearest candidates (C_1 and C_2) to C_R, the closest candidate greater or equal to C_R and the closest candidate smaller or equal to C_R:

$$C_1 = Q * SL + Syn \tag{3}$$

$$C_2 = (Q + sign(C_R - C_1)) * SL + Syn, \tag{4}$$

where

$$sign(x) = \begin{cases} -1, \ if \ x < 0 \\ 0, \ if \ x = 0 \\ 1, \ if \ x > 0 \end{cases}, \tag{5}$$

e.g., for $C_R = 17$, $Syn = 3$ and $SL = 8$, the two candidates are $C_1 = 19$ and $C_2 = 11$.

Finally, the closer of the two candidates is chosen (e.g., in the example above, 19 is the closer value to 17) as the most probable value of the original DCT coefficient encoded by the target camera: if $ABS(C_R - C_1) \leq ABS(C_R - C_2)$ then the candidate C_1 is chosen, or C_2 otherwise. $ABS(x)$ represents the absolute value of x.

In the implementation of the IWZ codec (the encoder, decoder and the two offline tools) it is considered a common set of configuring parameters: *frame width*—FW, *frame height*—FH, *block size*—B_{WH}, *number of DCT coefficients per block*—K, *quantization levels*—QL, *syndrome levels*—SL. In the evaluation presented in Section 4, the QL parameter was varied in order to obtain various points of rate-distortion.

3 Evaluation methodology for encoder complexity

The low-complexity encoding is a fundamental feature of Distributed Video Coding. Therefore, we present in this section a methodology for complexity evaluation of the IWZ encoder, introduced in Section 2. To this end, we consider as references two conventional encodings with low-complexity: Intra 4×4 and Intra 16×16, specified in the H.264/AVC standard [20]. For more accuracy of the evaluation, regardless of the eventual implementation optimizations that any encoder may have, in this section we propose a detailed assessment based on counting all the arithmetic operations (e.g., additions, subtractions, multiplications, divisions, etc.) and conditions (e.g., flag testing, checking the input parameters, etc.) employed for processing one frame. The relatively low complexity of the three encodings (IWZ, Intra 4×4 and Intra 16×16) makes it an achievable objective and therefore, it motivated the authors to propose this solution.

Due to the considered 8×8 block size for evaluation of the IWZ coding, it will be referred as "IWZ 8×8".

The total number of operations (arithmetic operations and conditions) for processing one frame can be counted in two steps, as follows: first, the bulk of computation is evaluated in Section 3.1 by counting all the operations from the high-level encoding (e.g., DCT transform, intra prediction, syndrome generation, etc.) up to (not including) the entropy coding. The number of operations per frame is always the same for each of the three encodings (IWZ 8×8, Intra 4×4 and Intra 16×16), regardless of the input video sequence. Consequently, the operations are manually counted. Secondly, the operations from entropy coding are counted in Section 3.2. Due to the usual variation of the complexity for this component, i.e., the total number of operations depends on the input data (for each of the three encodings), this partial

evaluation is automatically conducted by dedicated implementations and the average number of operations for six video sequences (see Section 4) is considered the actual result.

Finally, the sum of the two results (the total number of operations per frame) for each encoding is considered the evaluated complexity.

3.1 Evaluation of the high-level encoding

The evaluation of each of the three encodings is based on the H.264/AVC specification from [20], either partially (for IWZ encoder) or totally (Intra 4×4 and Intra 16×16), as discussed below. Although [20] specifies the decoding process, the same number of operations, either arithmetic operations or conditions, is expected for the corresponding high-level encoding.

We first consider an example in order to discuss the proposed evaluation method. In section 8.5.13.2 from the H.264/AVC standard [20] is specified the 8×8 integer DCT. Table 5 shows the partial counting (by groups of equations) of the operations required by this process. It also enumerates the types of arithmetic operations (see the "Counted operations" column) and specifies how many times each group of equations is repeated (see the "Multiplication factor" column). The total number of operations is also provided.

The total number of arithmetic operations is 896, as follows: 352 additions (+), 256 subtractions (−), 224 bitwise right shifts (>>) and 64 raisings of 2 to a power (2^X). Note that in the process defined in section 8.5.13.2 from [20] there is no condition defined and also, there are no sub-processes to be recursively called. However, these elements are usually present in other sections from [20] and they are fully considered in this evaluation. Moreover, for each encoding (IWZ 8×8, Intra 4×4 and Intra 16×16) are only considered those execution paths (subset of operations from a given section) from [20] that apply for the specified input parameters, described below.

The complexity of the high-level encoding for IWZ, as total number of operations per frame, can be evaluated as follows. First, are identified those input parameters which influence the number of operations and, specific values are chosen for this particular evaluation: *frame size*—320×240, *YUV format*—YUV 420, *block size*—8×8 and, *number of used DCT coefficients per block* (K)—32. Given these parameters, the total number of 8×8 blocks to be processed for the luma component is $40 \times 30 = 1{,}200$ blocks, and for each of the chroma components is $20 \times 15 = 300$ blocks. Therefore, the total number of 8×8 blocks to be processed per frame is 1800. Each block is evaluated by the following procedure.

Table 5 Example of operation counting for section 8.5.13.2 (the specification of 8×8 integer DCT) from [20]

Group of equations	Counted operations	Multiplication factor
Eq. (8-362) ... (8-369)	10 +, 8 −, 6 >>	8x (for $i = 0..7$)
Eq. (8-370) ... (8-377)	4 +, 4 −, 4 >>	8x (for $i = 0..7$)
Eq. (8-378) ... (8-385)	4 +, 4 −	8x (for $i = 0..7$)
Eq. (8-386) ... (8-393)	10 +, 8 −, 6 >>	8x (for $j = 0..7$)
Eq. (8-394) ... (8-401)	4 +, 4 −, 4 >>	8x (for $j = 0..7$)
Eq. (8-402) ... (8-409)	4 +, 4 −	8x (for $j = 0..7$)
Eq. (8-410)	1 +, 1 "2^X", 1 >>	64x (for i, $j = 0..7$)
Total	896 operations (352 +, 256 −, 224 >>, 64 "2^X")	

```
for each of the 1800 blocks of size 8x8 do
  manually count (recursively) the number of operations from
      section 8.5.13.2 (the 8x8 integer DCT) => 896
      arithmetic operations (352 +, 256 -, 224 >>, 64 "2^X"),
      no condition found
  quantize the 32 DCT coefficients => 64 arithmetic
      operations (32 +, 32 >>)
  additional tests (implementation specific) => 96 conditions
  syndrome generation => 32 arithmetic operations (32 %)
endfor

syndrome difference => 57600 arithmetic operations (57600 -)
is the first frame ? => 1 condition
additional tests (implementation specific) => 2 conditions
```

Consequently, for the IWZ encoder we counted a total of 2,016,003 operations per frame, as follows: 1,843,200 arithmetic operations (691,200 +, 518,400 −, 57,600 %, 460,800 >>, 115,200 "2^X") and 172,803 conditions. Note that the other input parameters of IWZ encoder (*syndrome levels*—SL, *quantization levels*—QL) do not have any influence on complexity evaluation. Therefore, they were not mentioned in this evaluation.

We used the same methodology for the evaluation of Intra 4×4 and Intra 16×16 encodings. The considered parameters are: *frame size*—320×240, *YUV format*—YUV 420, *coding mode* of each macroblock pair (the MBAFF flag)—frame mode, *intra prediction mode* of each macroblock (for both luma and chroma)—DC mode, *baseline profile* (level 5.1) [20]. We aimed to simplify the evaluation by adopting the same DC prediction mode for all the 4×4 blocks from all the macroblocks, for both luma and chroma. Moreover, as specified in [20], the DC prediction mode can always be applied for any 4×4 block. In this paper we also considered the disabled deblocking filter (for Intra 4×4 and Intra 16×16) for an objective comparison with IWZ 8×8 (see Section 4). However, the deblocking filter can be applied to the IWZ decoder without affecting the low-complexity of the IWZ encoder.

For both encodings the total number of macroblocks (of 16×16 size) to be processed per frame is $20 \times 15 = 300$.

The procedure for evaluation of Intra 4×4 is described next.

```
for each of the 300 macroblocks do
  manually count (recursively) the number of operations from
      section 8.3 (intra prediction, transform decoding for
      luma, picture construction) => 3165 arithmetic
      operations (2135 +, 84 -, 245 *, 245 /, 179 %,
      32 <<, 229 >>, 16 "2^X"), 842 conditions

  manually count (recursively) the number of operations from
      section 8.5.4 (transform decoding for chroma, not
      explicitly called in section 8.3) => 2947 arithmetic
      operations (780 +, 644 -, 289 *, 162 /, 416 %,
      264 <<, 264 >>, 128 "2^X"), 1409 conditions
endfor
```

The total number of operations per frame for Intra 4×4 encoding is 2,508,900, as follows: 1,833,600 arithmetic operations (874,500 +, 218,400 −, 160,200 ∗, 122,100 /, 178,500 %, 88,800 <<, 147,900 >>, 43,200 "2^X") and 675,300 conditions.

Finally, the Intra 16×16 is evaluated by the following procedure.

```
for each of the 300 macroblocks do
  manually count (recursively) the number of operations
      from section 8.3 (intra prediction) => 12114
      arithmetic operations (10340 +, 175 -, 358 *,
      470 /, 319 %, 452 >>), 1429 conditions

  manually count (recursively) the number of operations
      from section 8.5 (transform decoding and picture
      construction for luma, not explicitly called in
      section 8.3) => 6156 arithmetic operations
      (1713 +, 1362 -, 578 *, 386 /, 835 %, 513 <<,
      513 >>, 256 "2^X"), 2713 conditions

  manually count (recursively) the number of operations
      from section 8.5.4 (transform decoding for chroma,
      not explicitly called in section 8.5) => 2947
      arithmetic operations (780 +, 644 -, 289 *,
      162 /, 416 %, 264 <<, 264 >>, 128 "2^X"),
      1409 conditions
endfor
```

The total number of operations per frame for Intra 16×16 encoding is 8,030,400, as follows: 6,365,100 arithmetic operations (3,849,900 +, 654,300 −, 367,500 ∗, 305,400 /, 471,000 %, 233,100 <<, 368,700 >>, 115,200 "2^X") and 1,665,300 conditions.

3.2 Evaluation of the entropy coding

The same configuration for each encoding is assumed for evaluation of entropy coding (see Section 3.1).

We used our implementation of the IWZ encoder, described in Section 2, in order to automatically count the number of operations per frame. The average for six video sequences (see Section 4) was considered the actual evaluation: 115,200 arithmetic operations (112,223.67 +, 2,976.33 −) and 250,758.53 conditions.

For Intra 4×4 and Intra 16×16 encodings, we implemented a dedicated encoder by rigorously following the H.264/AVC specification from [20], both the high-level decoding from section 8.3 (e.g., intra prediction, residual frame, DCT transform, etc.) and the CAVLC (Context-Adaptive Variable-Length Coding) decoding from section 9.2. Our encoder (automatically) counts only the operations from the CAVLC encoding.

For CAVLC from Intra 4×4 we counted 172,217.83 arithmetic operations (37,861.67 +, 83,901.17 −, 5,764.33 ∗, 10,923.67 <<, 17,378.67 >>, 10,455.67 &, 3,956.67 |, 1,976.17 !) and 449,588.5 conditions, where the operations &, | and ! represent the binary AND, binary OR and binary NOT, respectively. For

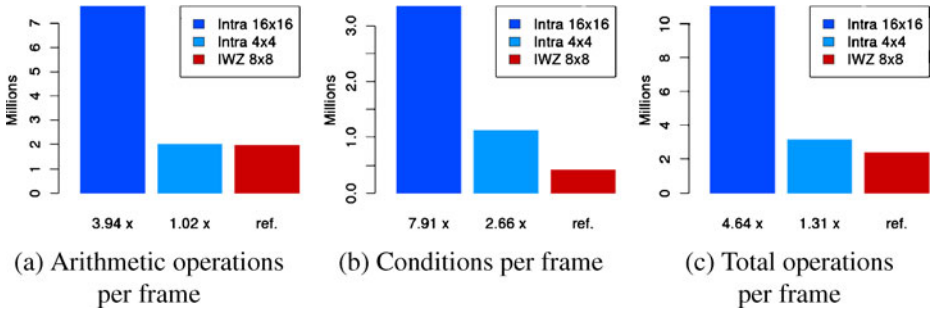


Fig. 6 The evaluated encoding complexity per frame: arithmetic operations in **a**, conditions in **b** and total operations in **c**

CAVLC from Intra 16×16 the counting results are: 1,341,483 arithmetic operations (156,620.67 +, 485,149.67 −, 53,532 ∗, 151,024.83 <<, 206,640.67 >>, 199,717.67 &, 59,664.5 |, 29,132.33 !) and 1,683,590.67 conditions.

In Section 4 are illustrated the graphical results for the entire evaluation. More details and discussions are also provided.

4 Results

We present the overall results for the evaluation methodology described in Section 3. Figure 6 illustrates the general complexity of the three encodings (Intra 16×16, Intra 4×4 and IWZ 8×8), as total number of arithmetic operations and/or conditions per frame. We used the IWZ 8×8 encoding as reference for comparison with Intra 16×16 and Intra 4×4 and therefore, the overall complexity of the IWZ 8×8 encoder is 1.31 times lower than Intra 4×4 and 4.64 times lower than Intra 16×16 (see Fig. 6c).

The overall complexity is discriminated in Fig. 7 between high-level encoding and entropy coding, as discussed in Section 3. The entropy coding represents 27.36%, 19.86% and 15.36% of the overall complexity for Intra 16×16, Intra 4×4 and IWZ 8×8, respectively, as illustrated in Fig. 7c. The complexity of Intra 16×16 is

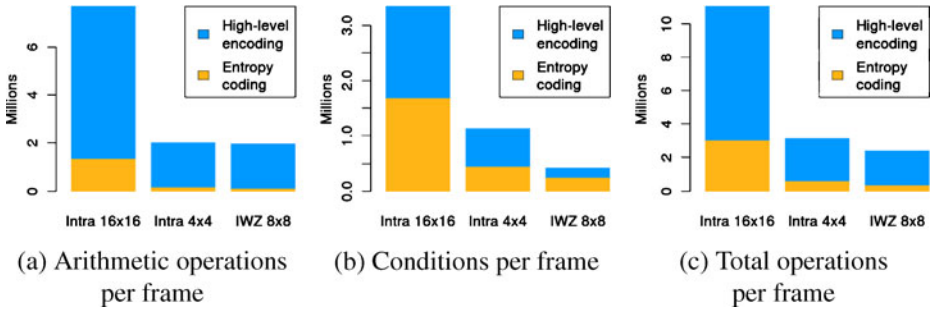
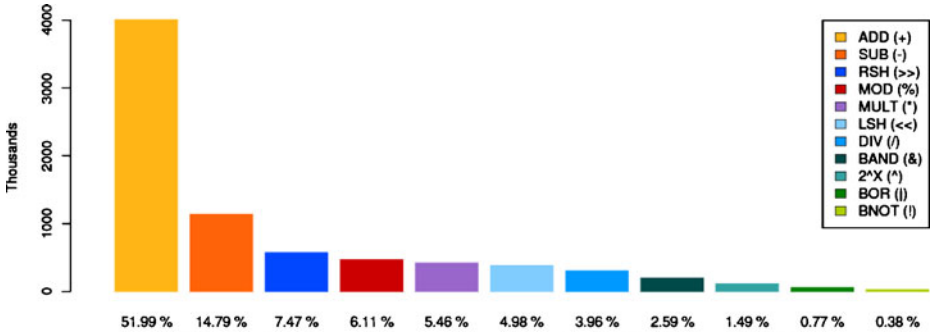
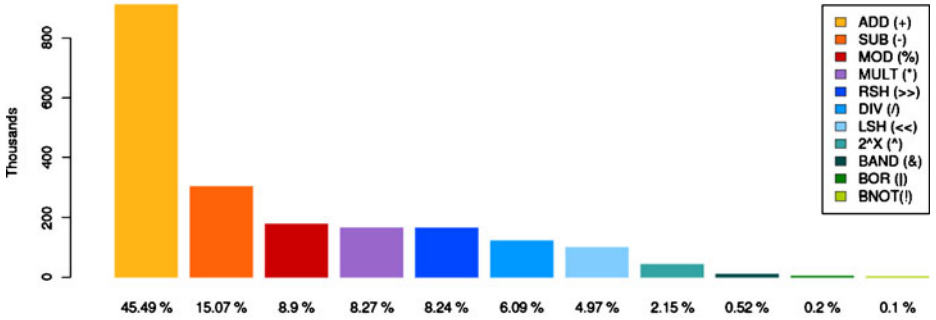


Fig. 7 The discriminated encoding complexity per frame, between high-level encoding and entropy coding: arithmetic operations in **a**, conditions in **b** and total operations in **c**

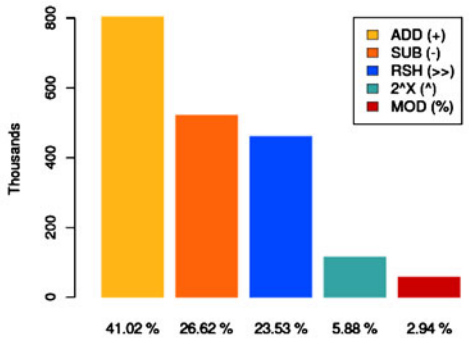
significantly higher than Intra 4×4 and IWZ 8×8, for both components (high-level encoding and entropy coding). Although Intra 16×16 uses the same entropy coding method (the CAVLC encoding) as Intra 4×4, the input data (i.e., the generated DCT coefficients, as specified in [20]) is considerably different and the complexity of CAVLC encoding increases significantly (4.86 times). This shows the variation of the complexity for the entropy coding component, as discussed in Section 3.



(a) Detailed arithmetic operations per frame for Intra 16x16



(b) Detailed arithmetic operations per frame for Intra 4x4



(c) Detailed arithmetic operations per frame for IWZ 8x8

Fig. 8 Detailed arithmetic operations per frame: for Intra 16×16 in **a**, for Intra 4×4 in **b** and for IWZ 8×8 in **c**

Fig. 9 Examples of original frames, one from each sequence: "Ribeira" in **a**, "Serralves" in **b**, "Castelo de Queijo" in **c**, "Sé do Porto" in **d**, "Rotunda da Boavista" in **e** and "Tourists" in **f**

The overall arithmetic operations per frame presented in Fig. 6a are classified by operation type in Fig. 8. The following types were found in our analysis: additions (ADD), subtractions (SUB), multiplications (MULT), divisions (DIV), modulo (MOD)—reminder of a division, raisings of 2 to a power (2^X), bitwise left shift (LSH), bitwise right shift (RSH), binary AND (BAND), binary OR (BOR), binary NOT (BNOT). As illustrated in Fig. 8c, the first three operation types found in the IWZ 8×8 encoder (ADD, SUB, RSH) represent a major proportion (91.17%) of all the arithmetic operations. These three are among the fastest operation types on most

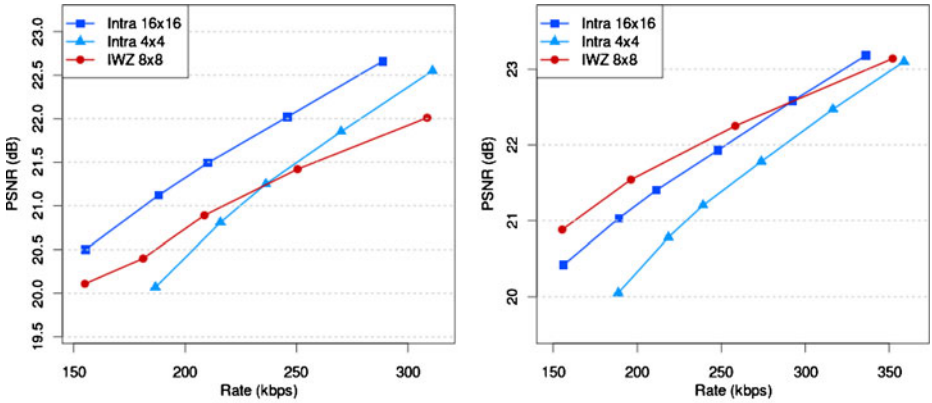
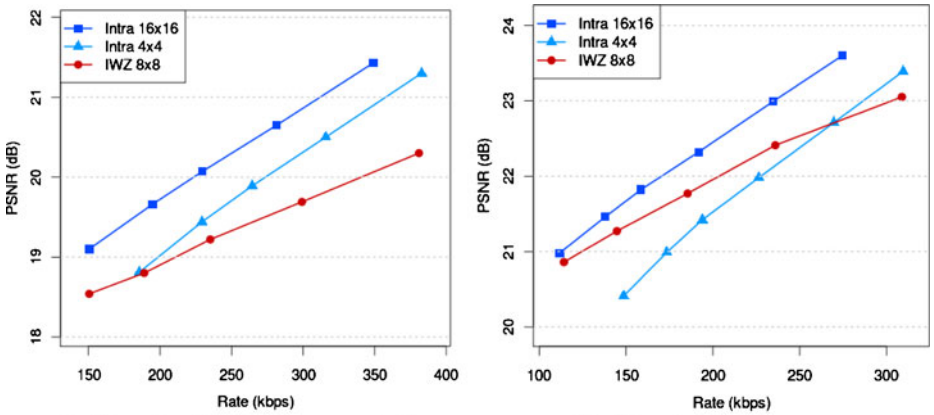


Fig. 10 The rate-distortion performance for the "Ribeira" sequence: for 10 fps sampling in **a** and for simulated slow camera movement in **b**



(a) "Serralves" sequence sampled at 10 fps

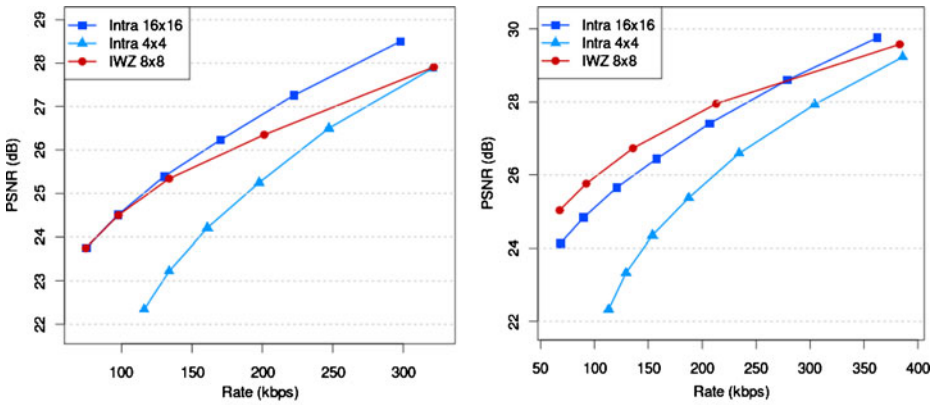
(b) "Serralves" sequence (simulated slow camera movement)

Fig. 11 The rate-distortion performance for the "Serralves" sequence: for 10 fps sampling in **a** and for simulated slow camera movement in **b**

implementations, i.e., the number of clock cycles required is much lower, as opposed to operations like multiplications or divisions.

We also present the evaluation results for a dataset composed by six video sequences, as follows: "Ribeira", "Serralves", "Castelo de Queijo", "Sé do Porto", "Rotunda da Boavista" and "Tourists". Each sequence has 320×240 frame size, 100 frames, 10 fps, and was sub-sampled and extracted from original sequences having 320×240 frame size, 30 fps. Figure 9 illustrates examples of original frames, one from each sequence.

This dataset was used for evaluation of the entropy coding complexity for each of the three encodings (see Section 3.2).



(a) "Castelo de Queijo" sequence sampled at 10 fps

(b) "Castelo de Queijo" sequence (simulated slow camera movement)

Fig. 12 The rate-distortion performance for the "Castelo de Queijo" sequence: for 10 fps sampling in **a** and for simulated slow camera movement in **b**

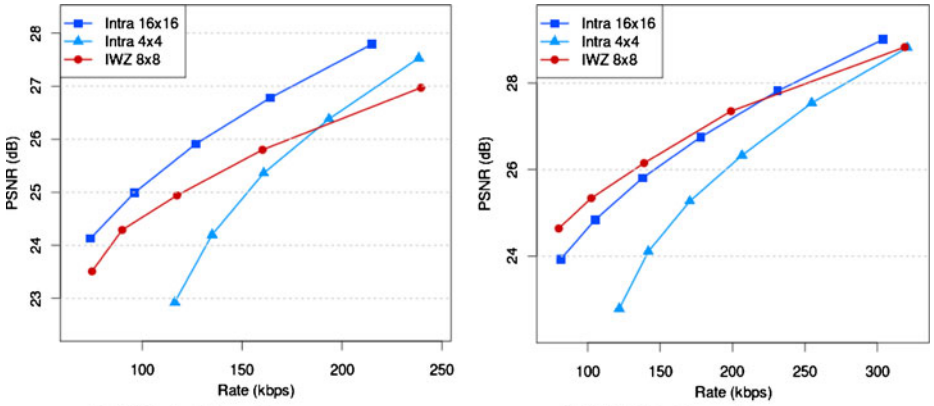


Fig. 13 The rate-distortion performance for the "Sé do Porto" sequence: for 10 fps sampling in **a** and for simulated slow camera movement in **b**

The rate-distortion performance evaluation is shown in Figs. 10a, 11a, 12a, 13a, 14a and 15a. The results for Intra 16×16 and Intra 4×4 were generated by the H.264/AVC reference software (JM 16.2) [12] with the configurations described in Section 3. The IWZ 8×8 coding generally shows a better performance than Intra 4×4 for low bitrates. It also provides inferior (although comparable) results when compared with Intra 16×16. Nevertheless, the Intra 16×16 coding is 4.64 times more complex, as illustrated in Fig. 6c.

For each of the six video sequences, the two offline tools presented in Section 2 (*Syndrome Statistics* and *Zero-Runs Statistics*) were initially run. The four generated

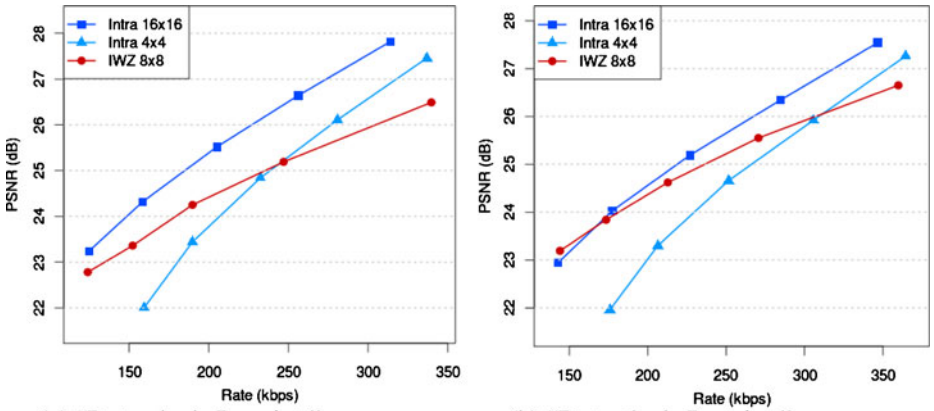


Fig. 14 The rate-distortion performance for the "Rotunda da Boavista" sequence: for 10 fps sampling in **a** and for simulated slow camera movement in **b**

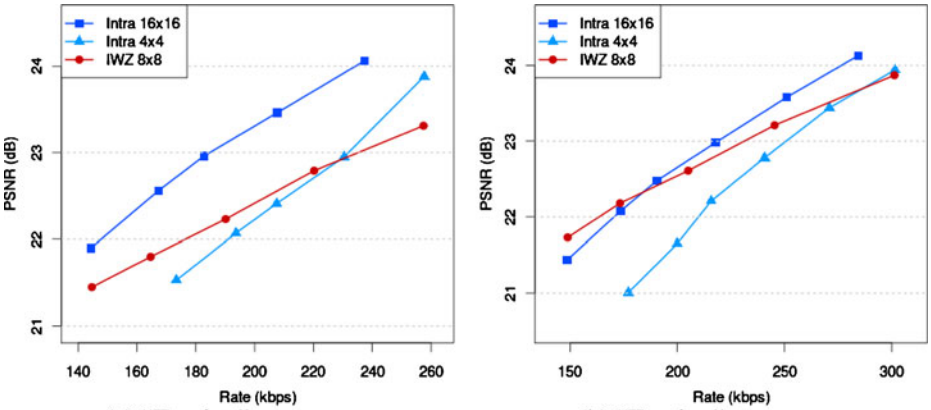


Fig. 15 The rate-distortion performance for the "Tourists" sequence: for 10 fps sampling in **a** and for simulated slow camera movement in **b**

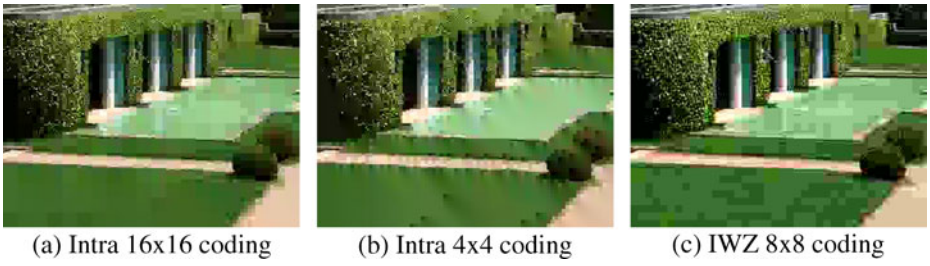


Fig. 16 Examples of decoded frames for the "Serralves" sequence: Intra 16×16 coding in **a**, Intra 4×4 coding in **b**, and IWZ 8×8 coding in **c**

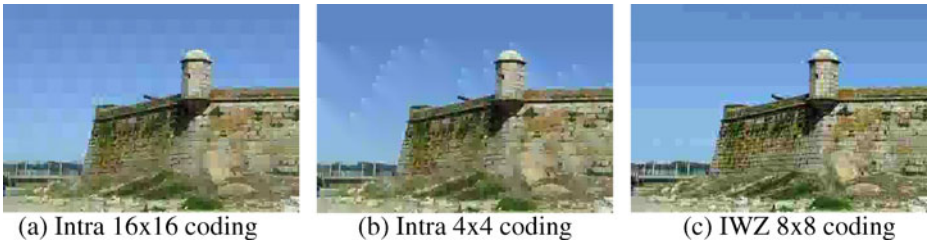


Fig. 17 Examples of decoded frames for the "Castelo de Queijo" sequence: Intra 16×16 coding in **a**, Intra 4×4 coding in **b**, and IWZ 8×8 coding in **c**

Huffman tables were subsequently used on encoding (of the same sequence). Nevertheless, due to the cyclic variation of the syndrome values and their distribution in a short range, i.e., the syndromes are simply remainders of divisions, the four generated Huffman tables use to be similar for different sequences.

Given the syndrome difference from one encoded frame to another, as discussed in Section 2, the IWZ coding is expected to have better rate-distortion performance on slower movement of the (target) camera, i.e., smaller differences between two consecutive frames. This efficiency technique is suitable for the scenarios described in Section 1 (e.g., video surveillance). The "Castelo de Queijo" sequence is particularly distinct from the others, i.e., for the first ten frames and for the last 13 frames the camera almost stalls (23 frames out of 100 with no noticeable movement of the camera). Figure 12a shows the better performance of the IWZ 8×8 coding in this case.

We prepared six additional video sequences for further evaluation of the scenarios with slow camera movement: we extracted six video sequences (each having 320×240 frame size, 100 frames, 30 fps) from the same original sequences mentioned above. We considered the sampling rate of 10 fps instead, in order to simulate a three times slower camera movement. We adopted this approach for a direct comparison with

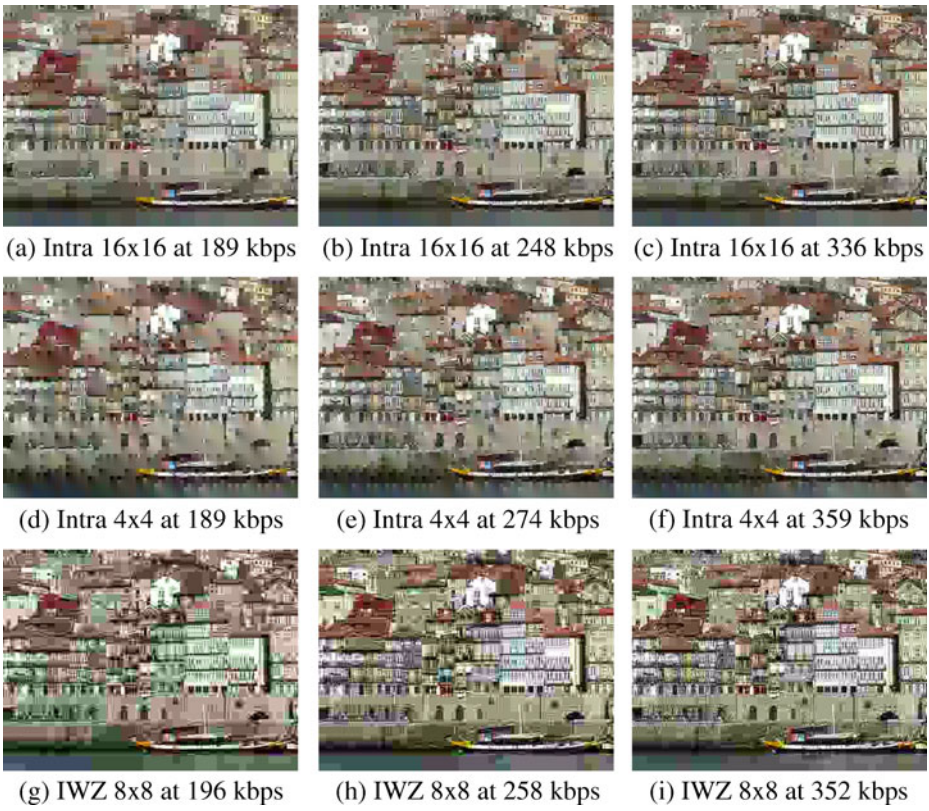


Fig. 18 Examples of decoded frames for various encoding rates, for the "Ribeira" sequence (simulated slow camera movement)

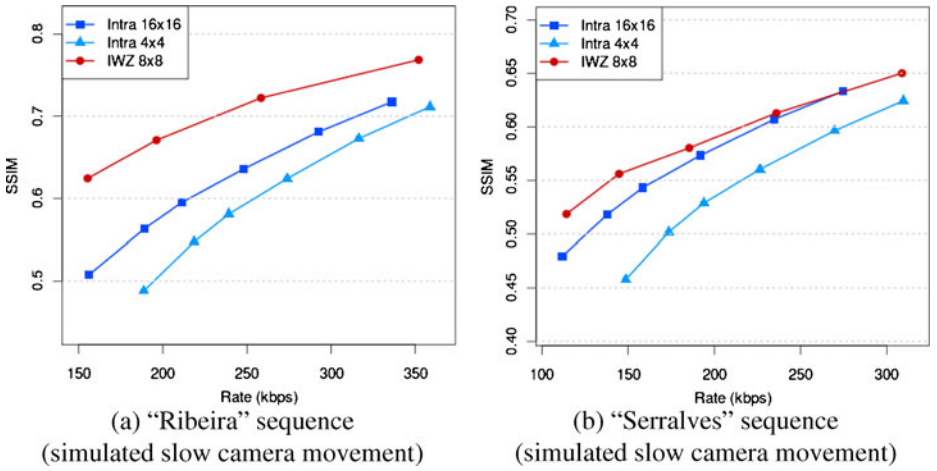


Fig. 19 The rate-SSIM performance for the "Ribeira" sequence in **a** and "Serralves" in **b**

the results presented in Figs. 10a, 11a, 12a, 13a, 14a and 15a. Therefore, maintaining the same complexities shown in Fig. 6c, the rate-distortion performance of the IWZ 8×8 coding improves significantly in this case (see Figs. 10b, 11b, 12b, 13b, 14b and 15b), providing even better results than Intra 16×16 for some sequences.

Figures 16 and 17 illustrate a few examples of decoded frames. For Intra 16×16 and Intra 4×4 codings there can be noticed the blocking effect (see Figs. 16a, b, 17a and b). As discussed in Section 3, the deblocking filter was disabled in this evaluation, for an objective comparison with the IWZ 8×8 coding.

Figure 18 illustrates more examples of decoded frames showing the distortion effects in the three codings for various encoding rates.

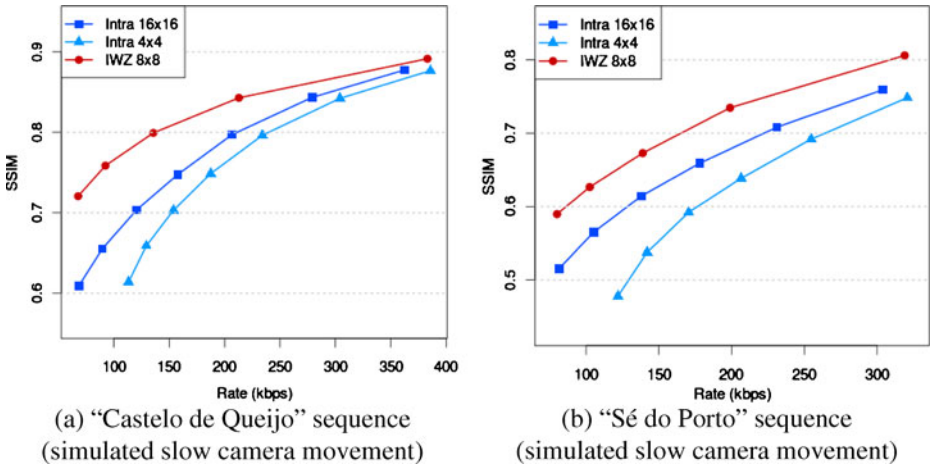


Fig. 20 The rate-SSIM performance for the "Castelo de Queijo" sequence in **a** and "Sé do Porto" in **b**

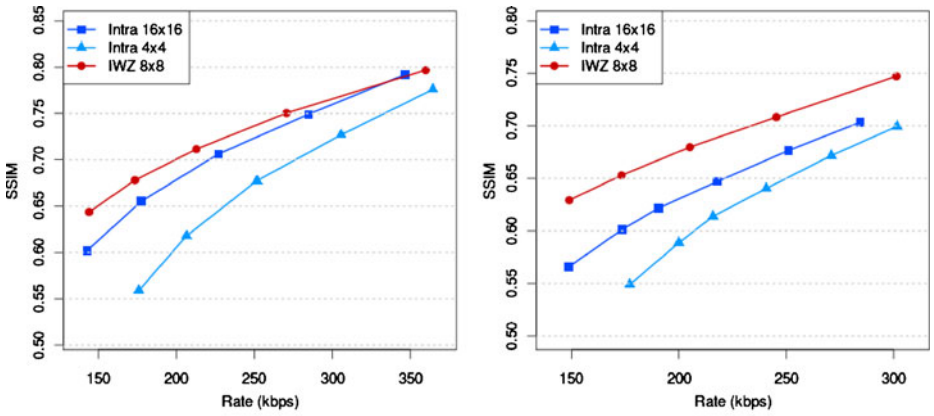


Fig. 21 The rate-SSIM performance for the "Rotunda da Boavista" sequence in **a** and "Tourists" in **b**

Due to the blocking artefacts present in the decoded frames (see Figs. 16 and 17), the perceptual measure SSIM (Structural SIMilarity) was also used for additional evaluations. Figures 19, 20 and 21 show the rate-SSIM performance for the same scenario with slow camera movement. The IWZ 8×8 coding provides better overall results over both the Intra 16×16 and Intra 4×4.

The dataset was evaluated on a computer with an Intel® Core™ 2 6420 2.13 GHz processor and 2.0 GB of physical memory. The implementation of the IWZ 8×8 encoder took an overall average time of 4.78 ms per encoded frame. It was considered the average time of five consecutive runs for each rate-distortion point from Figs. 10, 11, 12, 13, 14 and 15. For *Syndrome Statistics* and *Zero-Runs Statistics* it took an overall average time of 4.03 and 14.55 ms, respectively, per processed frame.

5 Conclusions

In this paper we presented a multi-view codec for Distributed Video Coding, called IWZ. The evaluation outlines two main achievements: first, the lower complexity of the IWZ encoder when compared with Intra 4×4 and Intra 16×16 conventional codings, and secondly, for low bitrates (as required by Distributed Video Coding), the higher rate-distortion performance of the overall IWZ codec over the Intra 4×4, and even over the Intra 16×16 for a slower (target) camera movement, specific to the considered scenarios (e.g., video surveillance).

We adopted a compromise between the low-complexity of the IWZ encoder and the high rate-distortion performance of the overall IWZ codec, both being considered fundamental requirements of Distributed Video Coding. We were challenged by the mutual dependence of the two objectives (e.g., for higher rate-distortion performance is needed a higher complexity of the IWZ encoder, and vice versa). To this end, we tried to benefit from the mature stage of the conventional coding [20] as much as possible, and to adapt it to the requirements of Distributed Video Coding. We finally adopted the integer DCT specified in [20].

Due to the lack of other multi-view codecs for the considered scenarios (e.g., no specific camera arrangement), we compared our codec with low-complexity conventional codings (Intra 4×4 and Intra 16×16). The evaluation is focused on the codec's performance. The robustness is beyond the scope of this paper and adequate solutions can be provided in future work to enforce the codec's resilience (e.g., sending periodically an Intra-frame when transmission errors occur), employed at decoder side in order to maintain the low-complexity of the encoder.

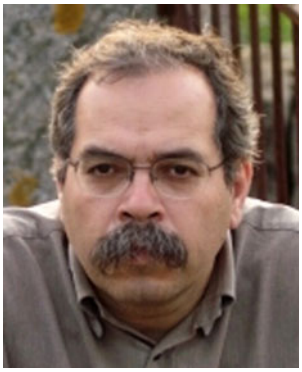
Acknowledgement The first author acknowledges the *Fundação para a Ciência e a Tecnologia, Portugal*, for the financial support.

References

1. Aaron A, Zhang R, Girod B (2002) Wyner–Ziv coding of motion video. In: 36th Asilomar conference on signals, systems and computer, Pacific Grove, USA
2. Aaron A, Rane S, Setton E, Girod B (2004) Transform-domain Wyner–Ziv codec for video. In: Proc. SPIE conference on visual communication and image processing
3. Artigas X, Tagliasacchi M, Torres L, Tubaro S (2006) A proposal to suppress the training stage in a coset-based distributed video codec. In: IEEE international conference on acoustics, speech, and signal processing, Toulouse, France, 14–19 May 2006
4. Artigas X, Ascenso J, Dalai M, Klomp S, Kubasov D, Ouaret M (2007) The DISCOVER codec: architecture, techniques and evaluation. In: Picture coding symposium (PCS)—2007, Lisbon, Portugal, 7–9 November 2007
5. Belkoura Z, Sikora T (2006) Towards rate-decoder complexity optimisation in turbo-coder based distributed video coding. In: Proc. international picture coding symposium, Beijing, P. R. China
6. Brites C, Ascenso J, Pedro JQ, Pereira F (2008) Evaluating a feedback channel based transform domain wyner-ziv video codec. Signal Process Image Commun 23(4):269–297. [Online]. Available: http://www.sciencedirect.com/science/article/B6V08-4S32NK9-1/2/e0b368561cd87009a992861007fc0eaf
7. Ciobanu L, Corte-Real L (2010) Successive refinement of side information for multi-view distributed video coding. Multimedia Tools Appl 48(3):411–436
8. Dalai M, Leonardi R, Pereira F (2006) Improving turbo codec integration in pixel-domain distributed video coding. In: IEEE international conference on acoustics, speech, and signal processing, Toulouse, France, 14–19 May 2006
9. Girod B, Aaron A, Rane S, Rebollo-Monedero D (2005) Distributed video coding. In: Proceedings of the IEEE, vol 93
10. Guo X, Lu Y, Wu F, Gao W, Li S (2006) Distributed multi-view video coding. In: Proceedings of SPIE-IS&T electronic imaging, SPIE, vol 6077, San Jose, California, USA, 15–19 January 2006
11. Guo X, Lu Y, Wu F, Zhao D, Gao W (2008) Wyner–Ziv-based multiview video coding. IEEE Trans Circuits Syst Video Technol 18(6):713–724
12. H.264/AVC JM reference software (JM 16.2) (2009) [Online]. Available: http://iphome.hhi.de/suehring/tml/
13. Lajnef K, Guillemot C, SiohanP (2006) Distributed coding of three binary and Gaussian correlated sources using punctured Turbo Codes. EURASIP Signal Process J (Elsevier) 86(11):3131–3149
14. Lan C, Liveris AD, Narayanan K, Xiong Z, Georghiades C (2004) Slepian–Wolf coding of multiple M-ary sources using LDPC codes. In: Proc. IEEE data compression conference, DCC'04, Snowbird, Utah, 23–25 March 2004
15. Li Y, Liu H, Liu X, Ma S, Zhao D, Gao W (2009) Multi-hypothesis based multi-view distributed video coding. In: Proceedings of the 27th conference on picture coding symposium, ser. PCS'09. IEEE Press, Piscataway, pp 45–48. [Online]. Available: http://portal.acm.org/citation.cfm?id=1690059.1690071
16. Lv H, Xiong H, Zhang Y, He Z (2008) Side information generation with constrained relaxation for distributed multi-view video coding. In: IEEE international symposium on circuits and systems, 2008. ISCAS 2008, pp 3450–3453

17. Pedro J, Soares L, Brites C, Ascenso J, Pereira F, Bandeirinha C, Dufaux F, Ebrahimi T (2007) Studying error resilience performance for a feedback channel based transform domain Wyner-Ziv video codec. In: Picture coding symposium (PCS)—2007, Lisbon, Portugal, 7–9 November 2007
18. Puri R, Ramchandran K (2002) PRISM: a new robust video coding architecture based on distributed compression principles. In: Proc. Allerton conference on communication, control and computing
19. Puri R, Ramchandran K (2003) PRISM: a "reversed" multimedia coding paradigm. In: Proc. Intl. conference on image processing, (ICIP), Barcelona, Spain
20. Recommendation ITU-T H.264: advanced video coding for generic audiovisual services. March 2009
21. Slepian D, Wolf JK (1973) Noiseless coding of correlated information sources. IEEE Trans Inf Theory 19:471–480
22. Tonoli C, Dalai M, Migliorati P, Leonardi R (2007) Error resilience performance evaluation of a distributed video codec. In: Picture cODING sYmposium (PCS)—2007, Lisbon, Portugal, 7–9 November 2007
23. Varodayan D, Aaron A, Girod B (2005) Rate-adaptive distributed source coding using low-density parity-check codes. In: Proc. Asilomar conf. signals, syst., comput., Pacific Grove, CA
24. Vatis Y, Klomp S, Ostermann J (2007) Inverse bit plane decoding order for turbo code based distributed video coding. In: International conference on image processing (ICIP)—2007, San Antonio, Texas, USA, 16–19 September 2007
25. Westerlaken R, Borchert S, Klein Gunnewiek R, Lagendijk I (2006) Dependency channel modeling for a LDPC-based Wyner-Ziv video compression scheme. In: International conference on image processing (ICIP)—2006, Atlanta, USA, 8–11 October 2006
26. Westerlaken RP, Borchert S, Gunnewiek RK, Lagendijk I (2007) Analyzing symbol and bit plane-based LDPC in distributed video coding. In: International conference on image processing (ICIP)—2007, San Antonio, Texas, USA, 16–19 September 2007
27. Wyner AD, Ziv J (1976) The rate-distortion function for source coding with side information at the decoder. IEEE Trans Inf Theory 22:1–10
28. Yang Y, Cheng S, Xiong Z, Zhao W (2003) Wyner–Ziv coding based on TCQ and LDPC codes. In: Proc. Asilomar conference on signals, systems and computers
29. Zhao Y, Garcia-Frias J (2006) Turbo compression/joint source-channel coding of correlated binary sources with hidden Markov correlation. EURASIP Signal Process J (Elsevier) 86(11):3115–3122

Lucian Ciobanu was born in Iasi, Romania, in 1978. He graduated in software engineering at the Faculty of Automatic Control and Computer Engineering, the "Gheorghe Asachi" Technical University of Iasi, Romania. He is researcher at the Institute for Systems and Computer Engineering (INESC) Porto, Portugal, since 2002. He received the PhD degree in electrical and computer engineering from the Faculdade de Engenharia da Universidade do Porto, Portugal, in 2011. His research interests include image/video coding and processing.



Luís Corte-Real was born in Vila do Conde, Portugal, in 1958. He graduated in electrical engineering from the Faculdade de Engenharia, Universidade do Porto, Portugal, in 1981. He received the M.Sc. degree in electrical and computers engineering in 1986 from Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisbon, Portugal and the Ph.D degree from the Faculdade de Engenharia, Universidade do Porto, in 1994. In 1984 he joined Universidade do Porto as a lecturer of telecommunications. He is currently associate professor at the Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto. He is researcher at the Institute for Systems and Computer Engineering (INESC) Porto, Portugal since 1985. His research interests include image/video coding and processing.