The UGent Institutional Repository is the electronic archiving and dissemination platform for all UGent research publications. Ghent University has implemented a mandate stipulating that all academic publications of UGent researchers should be deposited and archived in this repository. Except for items where current copyright restrictions apply, these papers are available in Open Access.

This item is the archived peer-reviewed author-version of:

**Title:** Distributed video coding for video communication on mobile devices and sensors

**Authors:** Peter Lambert, Stefaan Mys, Jozef Škorupa, Jürgen Slowack, Rik Van de Walle, and Christos Grecos


**Optional: link to the article** DOI: 10.4018/978-1-61520-761-9.ch020

To refer to or to cite this work, please use the citation to the published version:

**Authors (year). Title. journal Volume(Issue) page-page. doi**

Distributed Video Coding
for Video Communication
on Mobile Devices and Sensors

P. Lambert, S. Mys, J. Škorupa, J. Slowack, R. Van de Walle
Multimedia Lab,
Ghent University – IBBT,
Belgium

C. Grecos
School of Computing
University of the West of Scotland
Scotland, UK

ABSTRACT

In the context of digital video coding, recent insights have led to a new video coding paradigm called Distributed Video Coding, or DVC, characterized by low-complexity encoding and high-complexity decoding, which is in contrast to traditional video coding schemes. This chapter provides a detailed overview of DVC by explaining the underlying principles and results from information theory and introduces a number of application scenarios. It also discusses the most important practical architectures that are currently available. One of these architectures is analyzed step-by-step to provide further details of the functional building blocks, including an analysis of the coding performance compared to traditional coding schemes. Next to this, it is demonstrated that the computational complexity in a video coding scheme can be shifted dynamically from the encoder to the decoder and vice versa by combining conventional and distributed video coding techniques. Lastly, this chapter discusses some currently important research topics of which it is expected that they can further enhance the performance of DVC, i.e., side information generation, virtual channel noise estimation, and new coding modes.

Keywords – distributed video coding, dynamic complexity, Wyner-Ziv coding

INTRODUCTION

In traditional video coding schemes, such as MPEG-2, H.264/AVC, or VC-1, it is the encoder that exploits the statistics of the source signal. As a result, encoding requires significantly more computational resources than decoding, which very well suits traditional application scenarios like broadcasting or video-on-demand, where video is compressed once and decoded many times.
However, emerging applications such as wireless low-power video surveillance, video conferencing with mobile devices, or video communications in sensor networks, require ultra low-complexity encoders, possibly at the expense of a more complex decoder.

Surprisingly, results from information theory established in the 1970s suggest that this should be possible without losing any coding efficiency. In the context of digital video coding, these insights have led to a new video coding paradigm called Distributed Video Coding (DVC), which is based on Distributed Source Coding (DSC), and characterized by low-complexity encoding and high-complexity decoding.

**Distributed Source Coding**

![Figure 1. Joint source coding (left) vs. distributed source coding (right).](image)

DSC is a coding paradigm based on two major results from information theory: the Slepian-Wolf theorem and the Wyner-Ziv theorem. Slepian and Wolf (1973) proved that two correlated random sequences generated by repeated independent drawings of a pair of discrete random variables $X$ and $Y$ can be coded as efficiently by two independent coders as by a joint encoder, provided that the resulting bit streams are jointly decoded (Figure 1). In particular, this result states that $R_X + R_Y \geq H(X,Y)$, $R_X \geq H(X \mid Y)$, and $R_Y \geq H(Y \mid X)$. This means that the sum of the rates of the sources $X$ and $Y$ can indeed achieve the joint entropy, just as for joint encoding (Figure 2).

![Figure 2. Achievable rate regions for the coding schemes from Figure 1.](image)

A special case of DSC is when a decoder makes use of so-called side information. Here, the source sequence $X$ is correlated with some side information $Y$ which is unavailable at the encoder, but available at the decoder (Figure 3). Since conventional encoding techniques can code $Y$ at a rate $R_Y = H(Y)$, the above results indicate that $R_X = H(X \mid Y)$ is achievable. This case will be the starting point for DVC architectures, as discussed later in this chapter.
The work of Slepian and Wolf, which involved lossless compression, was extended to lossy compression by Wyner and Ziv (1976). They considered compression with decoder side information. This time, however, a distortion $D = E[d(X, X')]$ between the original signal $X$ and the decoded signal $X'$ is allowed. Let $R_{X|Y}^{WZ}(D)$ be the achievable lower bound for the bit rate given a distortion $D$, and $R_{X|Y}(D)$ the rate required in case the side information is available at the encoder as well.

Given these notations, Wyner and Ziv (1976) proved that a rate loss $|R_{X|Y}^{WZ}(D) - R_{X|Y}(D)| \geq 0$ occurs when the encoder does not have access to the side information. More importantly, they also proved that the equality holds in the case of Gaussian memoryless sources and a mean squared error distortion metric $d$. Later, these results were extended to more general cases, proving that the equality also holds for source sequences $X$ that are the sum of arbitrarily distributed side information $Y$ and independent Gaussian noise $N$ (Pradhan, 2003), and that the rate loss for sources with general statistics and a mean squared error distortion metric $d$ is less than 0.5 bits per sample (Zamir, 1996).

**Distributed Video Coding and its Applications**

Distributed video coding tries to exploit the theoretical results from distributed source coding in the context of video compression. In doing so, DVC comes with a number of advantages and accompanying application scenarios. Firstly, using independent encoders and a joint decoder means in practice that the complexity burden is now at the decoder side instead of at the encoder (as in conventional video coding solutions). This in turn could lead to low production cost, low power consumption, and very small encoders. Application scenarios include video capturing with mobile devices or wireless low-power video sensors for surveillance.

Secondly, DVC comes with inherent error robustness. Since there is no prediction loop in the encoder of a distributed video codec, error propagation is not an issue as it is in conventional predictive video codecs like MPEG-2 or H.264/AVC. Furthermore, since DVC is based on error correcting codes (see later), it is straightforward to efficiently extend such a codec with extra protection against transmission errors.

Thirdly, DVC looks promising for the compression of multi-view video sequences. Usually there is a strong correlation between sequences generated by different cameras in a multi-view setup. Using the principles of DVC, it should be possible to efficiently compress these different sequences using separate encoders (i.e., the different cameras/encoders do not need to communicate with each other), provided that the resulting bit streams are decoded using a single joint decoder.
OVERVIEW OF DVC SOLUTIONS

The fundamental theoretical results of Slepian-Wolf and Wyner-Ziv only provide bounds for the rate (and distortion in case of Wyner-Ziv) of a DVC system, but these results do not provide insights in how to build such a system. In this section, we explain how to apply the concepts of DSC to video compression. Starting point is a global point of view and an indication of the main problems that need to be dealt with to build an efficient codec. This knowledge is then used to introduce the two pioneering architectures, i.e., the PRISM system developed at Berkeley by Puri and Ramchandran (2002), and the system proposed by Aaron et al. (2004) at Stanford University. And although many extensions have been proposed by other researchers (including the authors of this chapter), the main architectures have stayed more or less the same.

A General Architecture for DVC

A video sequence consists of a number of frames (temporal axis) and each frame can be divided into non-overlapping blocks, often referred to as macroblocks (spatial axis). If one wants to use distributed source coding principles for video compression of one video sequence, the sequence must be partitioned in a certain way, as to obtain the two correlated sources mentioned in the Slepian-Wolf and Wyner-Ziv theorems. For example, the sequence can be partitioned into I frames and W frames, using the temporal direction only (see Figure 1). The description below assumes this particular partitioning but remains valid for other partitioning strategies. The I frames are coded independently from other frames in the sequence, for example using intra-coding techniques available in H.264/AVC. On the other hand, W frames are coded by exploiting correlation between the frames. However, this correlation is exploited at the decoder only, i.e., the encoder codes each frame (W or I) independently from other frames. To achieve this, the decoder generates side information Y using one or more previously decoded frames (I' and/or W').

The main problem with the virtual channel is that the virtual noise statistics need to be known by both encoder and decoder: the Wyner-Ziv (WZ) encoder needs to know how many parity bits should be sent to the decoder, and the WZ decoder needs the conditional distribution \( P(W \mid Y) \) for efficient channel decoding (e.g., Viterbi-like decoding). However, the encoder only has access to W while the decoder can only access Y. To solve this problem, two solutions are frequently used in the literature. In the first solution, Y is estimated at the encoder and this information is used to estimate \( R_W \). Information about the conditional distribution \( P(W \mid Y) \) is then sent to the
decoder along with the error correcting bits. The disadvantage of this solution is that complexity is added to the encoder which is typically not desired in a DVC context. In the second solution, it is the decoder that estimates $P(W | Y)$ and calculates the number of parity bits that are needed to correct $Y$ reliably. Subsequently, a feedback channel is used to request the amount of parity bits from the encoder. However, the use of a feedback channel is impractical in video storage scenarios, and even in streaming scenarios, the use of a feedback channel should be limited to avoid excessive delays.

In short, the side information $Y$ plays a crucial role in a DVC system. Estimating $W$ at the decoder using previously decoded data is not straightforward due to complex motion, deformation, occlusion, and so on. In addition, modeling the virtual noise is difficult and results in a trade-off between additional complexity at the encoder and the use of a feedback channel.

To achieve different degrees of compression in a practical system, quantization is performed, usually preceded by a transformation step such as a discrete cosine transform (DCT) or a wavelet transformation. Therefore, a general DVC architecture can be drawn as depicted in Figure 4.

**Figure 4. General architecture for distributed video coding**

Assume again that the video sequence is partitioned into intra frames $I$ and WZ frames $W$. As before, $I$ frames are intra coded and sent to the decoder where they can be decoded independently from other frames. At the encoder, each $W$ frame is transformed ($T$), quantized ($Q$) and channel-coded. At the decoder, the side information $Y$ is generated and transformed into $Y^T$. The channel decoder selects the most likely quantization bin for $W$, denoted as $W', Q$, using the side information $Y^T$ and the correlation noise statistics $P(W | Y^T)$ provided by the virtual noise estimation module. Next, reconstruction is performed. This step is equivalent to inverse quantization but it also takes $Y^T$ into account: $W' = E[W | Y^T, W']$. Finally, the result is inverse transformed.

Although the theorems of Slepian-Wolf and Wyner-Ziv date back to the seventies, it was only quite recently that practical systems have been developed, first by Pradhan and Ramchandran from the University of California at Berkeley, and later on by Aaron and Girod at Stanford...
University. Many researchers have proposed extensions to both systems in the years that followed, but the basic architecture has stayed more or less the same. In the next subsections we will first discuss the PRISM system proposed by Ramchandran et al., and then we will discuss the Stanford codec.

**PRISM**

Pradhan and Ramchandran (1999) proposed a technique called DIstributed Source Coding Using Syndromes (DISCUS) and they created a framework for video coding which they called PRISM: Power-efficient, Robust, high-compression, Syndrome-based Multimedia coding (Puri, 2002). PRISM adopts a spatial partitioning approach to obtain the correlated sources of the WZ theorem. At the encoder, each macroblock B is first classified into one of several classes, based on the squared error difference between B and the co-located macroblock in the previous frame. If both macroblocks are strongly correlated, B is not coded and signaled as SKIP. In this case, the decoder can simply take the co-located macroblock in the previous frame without the need for channel coding. On the other hand, if there is very little correlation, B is intra-coded because it is likely that the generated side information Y will not be a very accurate prediction.

In all other cases, B is WZ-coded as follows. Each block is first transformed using a DCT transformation and the coefficients are scanned in zig-zag order. The side information Y will be correlated with B for the low frequency coefficients, but the correlation between the higher frequency coefficients is usually low. Therefore, the higher frequency coefficients are intra-coded, i.e., they are quantized and run-length Huffman (entropy) coded. Hence, only the low frequency coefficients are actually WZ coded. In PRISM, syndrome codes are used as channel codes, and both the code and quantization step size are determined by the estimated correlation noise. First, the remaining low frequency coefficients are base quantized and syndrome coded. Next, to achieve the target distortion, the quantization is further refined, and the index of the refinement interval inside the base interval is transmitted to the decoder. In addition, a CRC check is calculated which serves as a signature of the quantized codeword sequence.

At the decoder, all possible (half-pixel accurate) side information blocks are listed (within a given search range). From this set of candidate blocks, the best predictor is selected using the Viterbi algorithm and the received syndrome bits. If this best predictor matches the CRC, syndrome decoding terminates and the block is returned as output. Otherwise, the next best predictor is chosen and so on.

In summary, no conventional motion estimation is performed in PRISM, but instead all candidate side information blocks are listed and tried one by one until the output matches the CRC. Also, since the co-located macroblock in the previous frame is used to perform the initial classification of the macroblock at the encoder, each frame is not coded independently from other frames. Hence, the correlated sources of the Wyner-Ziv theorem are in fact the already classified intra and WZ blocks.

**Stanford Codec**

Aaron et al. use a frame-based approach and partition the video sequence into I frames (or key frames) and W frames. First, they developed a pixel-domain codec (Aaron, 2002) which was later
on extended to the transform domain (Aaron, 2004). The overall architecture is the same as the frame-based example described in the general overview above (also see Figure 4).

This architecture is used as a basis for DVC systems by the majority of the research community, including the authors of this chapter. Therefore, this architecture will be used in the next section to explain the functional building blocks of a DVC system in more detail.

**Other Systems**

Many researchers have proposed new techniques, including techniques to improve the generation of side information by using bidirectional motion refinement and spatial smoothing (Ascenso, 2005), by exploiting both temporal and spatial correlation (Adikari, 2006a; Tagliasacchi, 2006a), and by using multiple side information streams (Adikari, 2006b). Rate-distortion analysis has been provided for motion extrapolation (Li, 2007), motion compensated interpolation (Tagliasacchi, 2007a) and hash-based side information generation (Tagliasacchi, 2007b). The feedback channel has been studied (Pedro, 2007) and practical request stopping criteria have been formulated (Tagliasacchi, 2007c) as well as how to eliminate the feedback channel (Morbée, 2007). Alternative channel codes have been studied such as LDPC codes (Aaron, 2006; Liu, 2006) and overlapped quasi-arithmetic codes (Artigas, 2007a). Also enhanced reconstruction techniques have been proposed (Vatis, 2007).

DVC techniques have also been used in other scenarios, such as traditional video coding with forward error correction using WZ coding (Baccichet, 2006; Bernardini, 2007; Rane, 2004) and multi-view coding (Flierl, 2006; Tosic, 2007; Yang, 2007). An interesting hybrid system has been proposed by Mukherjee (2006), who uses H.264/AVC to code subsampled frames. At the decoder, the decoded subsampled frame is used to generate the side information for the full resolution frame. Subsequently, the frame is decoded using WZ bits.

**FUNCTIONAL BLOCKS IN A DVC SYSTEM**

To illustrate the internal working of a DVC system, we provide a block-by-block overview of our DVC system, depicted in Figure 4. This system is based on the architecture proposed by Aaron et al. (2004), but it uses different methods for side information generation and virtual noise estimation. In the following we will discuss each building block in some more detail: intra coding, transformation and quantization, bitplane extraction, turbo coding, side information generation, virtual noise estimation, and, finally, reconstruction.

**Intra Coding**

Frames are intra coded and decoded following the H.264/AVC specification (Wiegand, 2003) for intra-coded pictures. Both coding and decoding can be done completely independently from any other frame. At the decoder, the intra decoded frames are stored in a buffer so they can be used as reference frames later.

**Transformation and Quantization**
For transformation, the H.264/AVC integer transform (Malvar, 2003) is used as a computationally efficient approximation of the DCT. Each transform coefficient is uniformly quantized in the same way, using one of six available quantization patterns, resulting in quantized coefficients consisting of 8-3 bits.

In particular, in H.264/AVC, the DCT is approximated as a forward transformation $T_1$ followed by a scaling part $S_1$ (Figure 5). Analogously, the IDCT is approximated by a backward scaling $S_2^{-1}$ and backward transformation $T_2^{-1}$. The scaling steps $S_1$ and $S_2^{-1}$ are integrated within the quantization, resulting in $Q_1$ and $Q_2^{-1}$, respectively. An important consequence is that, due to this combination of transformation and quantization, the backward transformation $T_2^{-1}$ and the backward quantization $Q_2^{-1}$ are each individually no longer the exact inverse operations of the forward transformation $T_1$ and quantization $Q_1$. In the next paragraph, we will explain how to take this into account when implementing H.264/AVC transformation and quantization in the DVC codec.

In the DVC codec, at the encoder, $W$ frames are transformed using $T_1$ and quantized using $Q_1$. At the decoder, $Y$ is transformed using $T_1$ before being used as side info by the turbo decoder. The turbo decoder decodes the frame and returns for each transform coefficient $W_k^T$ the quantization bin $q_k^T$ in which it is contained. Using these decoded quantization bins and the transformed side info $Y^T$, the reconstruction acts as an inverse quantizer and calculates the most likely value $w_k^T$ for each bit. However, since reconstruction really acts as an inverse quantizer for $Q_1$, and since $S_2^{-1}$ is not actually the inverse operation of $S_1$, the decoded values are not properly scaled and need to be rescaled by both $S_1$ and $S_2^{-1}$ before executing the backward integer transform $T_2^{-1}$. Naturally, the successive scaling by $S_1$ and $S_2^{-1}$ can be combined so that in practice only one scaling operation $S$ needs to be performed.

![Figure 5: The H.264/AVC transformation and quantization scheme.](image)

**Bitplane Extraction**

Bitplane extraction is performed after quantization. First, we shift the AC coefficients so that all quantized values are positive. Next, coefficients are grouped in so-called coefficient bands. For example, all DC coefficients are grouped into one coefficient band. Then, for each band, the bits at corresponding positions are grouped into bitplanes. For example, the most significant bit of the DC coefficients form one bitplane. After this step, each of the bitplanes is fed as a binary input word to the turbo coder, which will encode it.
Bitplane coding comes with two advantages. Firstly, the codewords that are inputted into the turbo encoder are fixed length. More precisely, the length of the codewords corresponds with the number of 4x4 (transformation) blocks per frame. Secondly, decoding can start with the most significant bitplanes and less significant bitplanes could be skipped.

**Turbo Coding**

Turbo codes are used as channel codes in the described system. In short, turbo coding consists of two fundamental ideas: a code design that produces a code with random-like properties, and a decoder design that makes use of soft-output values and iterative decoding. In the following, first a brief description of the turbo encoder is given, followed by a short discussion of the turbo decoder. After that, a discussion of the puncturing process follows. For more detailed information on turbo coding, we refer the interested reader to Lin and Costello (2004).

Turbo encoding

A turbo encoder (Figure 6) consists of two (in most cases identical) systematic convolutional encoders. More precisely, systematic feedback encoders are used. This means that there is a closed loop in the encoder’s memory circuit, because of which the parity bits depend not only on a finite number (equal to the number of memory blocks) of previous input bits, but on all previous input bits. Each such systematic feedback encoder normally produces two output bits per input bit: a systematic bit, which equals the input bit, and a parity bit. In DVC however, the systematic bits are discarded, and thus the DVC turbo encoder generates two output (parity) bits per input bit $w_k^q$.

An interleaver $\pi$ is put in front of the second encoder. This pseudo-random interleaver (which needs to be known at the decoder also) permutes the input sequence before feeding it to the second encoder, ensuring that both encoders generate different parity bits. This introduces some kind of randomness to the code, which highly attributes to the high performance.

Convolutional encoders require very little computational and hardware resources. The presence of the interleaver in the turbo encoder requires the whole input sequence to be stored in memory, which can be quite long, but this is still nominal compared to, for instance, YUV frames.
that need to be stored in memory. Thus, the turbo encoder can be implemented very efficiently with small hardware requirements, which perfectly fits the low encoder complexity paradigm of DVC.

**Turbo decoding**

At the turbo decoder (Figure 6), optimal decoding of the concatenated code would computationally be too complex. Therefore a suboptimal iterative process is applied involving two soft-input, soft-output (SISO) convolutional decoders passing information to each other. Each SISO decoder calculates the Logarithm of the A Posteriori Probability (LAPP) ratio for each encoded bit $w_k^0$. From the LAPP ratios, extrinsic information (i.e., information calculated from data not accessible by the other SISO decoder, namely the parity bits $p^{(j)}$ corresponding to the current SISO decoder) is extracted and passed as a priori information to the other SISO decoder. Once a given convergence criterion is satisfied, the iterative decoding process stops. The LAPP ratios calculated in the last iteration are fed into the decision module, where a final decision for each bit is made.

So, each SISO decoder has to calculate the LAPP ratio $L(w_k^0) = \ln \left( \frac{P(\hat{w}_k^0 = 1 | r)}{P(\hat{w}_k^0 = 0 | r)} \right)$, where $r$ represents all information known at the SISO decoder, i.e., both the transformed side information $\gamma^T$ and the corresponding parity bits $p^{(1)}$ or $p^{(2)}$. As can be seen, a positive value of the LAPP ratio denotes a higher probability that the input bit was 1, whilst a negative LAPP ratio indicates an input bit 0.

![Figure 7: Trellis diagram for a simple systematic convolutional code. On the vertical axis, the encoder states are given; on the horizontal axis, the time steps $k$. A dotted line corresponds with an input bit 0, a full line with an input bit 1. The systematic output bits are identical to the input bits; the corresponding parity output bits are written as labels on the edges.](image)

Based on the structure of the trellis diagram of the code (Figure 7), the a posteriori probabilities $P(w_k^0 | r)$ can be rewritten (Lin and Costello (2004)) so that
\[
L(w^0_k) = \ln \left( \frac{\sum_{s'} p(s_{k-1} = s', s_k = s, r)}{\sum_{s'} p(s_{k-1} = s', s_k = s, r)} \right),
\]

where \( S^1 \) (resp. \( S^0 \)) represents all edges in the trellis corresponding to a state transition \( s_{k-1} \rightarrow s_k \) considering an input \( w^0_k = 1 \) (resp. \( w^0_k = 0 \)).

It has been shown (Lin and Costello (2004)) that the probability \( p(s_{k-1} = s', s_k = s, r) \) can be written as \( \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s) \) with \( \alpha, \beta \) and \( \gamma \) formally defined as \( \alpha_{k-1}(s') \triangleq p(s_{k-1} = s', r_{k-1}^{(s')}) \), \( \beta_k(s) \triangleq p(r_k^k | s_k = s) \) and \( \gamma_k(s', s) \triangleq p(s_k = s, r_k | s_{k-1} = s') \).

Intuitively, \( \alpha_{k-1}(s') \) is the probability of getting from the start of the trellis to state \( s' \) at time \( k-1 \). It is often called the forward metric, since it can be computed recursively using \( \gamma_k \) as follows: \( \alpha_k(s) = \sum_{s' \in \alpha_{k-1}} \alpha_{k-1}(s') \gamma_k(s', s) \). Analogously, \( \beta_k(s) \) is the probability of getting from state \( s \) at time \( k \) to the end of the trellis. It is called the backward metric and can be written as \( \beta_{k-1}(s') = \sum_{s \in \alpha_k} \beta_k(s) \gamma_k(s', s) \).

\( \gamma_k(s', s) \), often called the branch metric, is the probability of making the state transition from state \( s' \) at time \( k-1 \) to state \( s \) at time \( k \), all given the received values \( r \). It is calculated as a function of the a priori probability \( L^\text{prior}_j \) and the probability \( p(r_k^k | s_{k-1} = s', s_k = s) \) which is the conditional probability that \( r_k = (r_k^T, p_k^{(s')}) \) is received when the state transition \( s' \rightarrow s \) has occurred. This probability is calculated at the decoder based on the estimation of the virtual noise.

**Puncturing and feedback channel**

The turbo code as described above does not compress the input sequence. On the contrary, two parity bits are generated for each input bit. However, not all parity bits are needed at the decoder to correct all errors in the side information. Therefore, we need a way to adapt the amount of parity bits that are transmitted to and used at the decoder. This is done by a procedure called puncturing.

In short, at the encoder, instead of directly transmitting all parity bits to the decoder, they are saved in a buffer. Then, the decoder asks for a certain amount of parity bits from the buffer through a feedback channel. If the amount of bits is not big enough to correct the side information, the decoder asks for more bits from the buffer. This way the decoder estimates the smallest number of bits (i.e., minimal rate) required to transmit the current bitplane.

Several online techniques to decide whether or not the decoding was successful have been described in the literature (Tagliasacchi, 2007c; Kubasov, 2007a; Artigas, 2007b). Recently, the authors of this chapter (Škorupa, 2009) showed that better results can be achieved by using the sign-difference ratio as a stopping criterion. This criterion counts the number of sign differences between the a priori and the extrinsic value at the SISO decoder. If that number reaches zero,
decoding is considered successful. Otherwise, more turbo decoding iterations are performed. If a predefined maximum number of turbo decoding iterations are performed without triggering the stopping criterion, decoding is considered unsuccessful and more parity bits are requested.

Furthermore, to save the number of requests through the feedback channel, the initial rate is estimated from the error probabilities estimated during virtual noise estimation (see later).

For optimal performance it is desirable that the transmitted parity bits are spread over the whole parity sequences $p^{(1)}$ and $p^{(2)}$. At the same time, strong structure in the puncturing pattern seems to degrade the performance of the code. To address these rather contradictory needs the following puncturing procedure is employed. The parity sequences are divided into blocks with a fixed length. When the decoder requests a certain amount of parity bits, the number of bits to be transmitted from each block is calculated, and these bits are chosen in a pseudo-random fashion in each block.

**Side Information Generation**

The side information generation module is responsible for creating a prediction of the Wyner-Ziv frame to be decoded based on already decoded I or W frames. Since the frame to be predicted is not available at the decoder, traditional motion compensated prediction (using block matching) is not possible. Therefore, interpolation or extrapolation techniques are used. So far, best results are achieved using a technique called motion compensated interpolation (see below). Extrapolation techniques were also proposed in the literature (Borchert, 2007), and are especially useful when long GOP sizes are used.

Motion compensated interpolation is a motion estimation technique that is based only on information contained in the reference frames, and does not use any information from the frame that is being predicted. Therefore, it can be applied in the DVC decoder, where the frame being predicted is not available. Originally, the technique was proposed by Ascenso et al. (2005). Later on, several extensions and improvements were added. Of these improvements, the ones proposed by Ascenso et al. (2006) and Klomp et al. (2006) are incorporated in the described codec. The technique consists of five steps, which are each briefly discussed below.

**Mean filtering of reference frames**

First of all, both past and future reference frames are filtered using a mean filter with a 5x5 square kernel. This step will help the motion estimation to choose motion vectors that better match the true motion field instead of simply choose the best motion vectors in the rate-distortion sense. This is important because the motion vectors will be interpolated later on.

**Forward motion estimation between reference frames**

After mean filtering the reference frames, a traditional block matching algorithm is used to estimate the motion between future and past reference frames. A modified version of the Mean Absolute Difference (MAD), given below, is used as cost function. In this cost function a penalizing term is introduced which adds an increasing cost when motion vectors go to more extreme positions in the search range.
\[ \text{cost}(d_x, d_y) = (1 + 0.05 \sqrt{d_x^2 + d_y^2}) \cdot \text{MAD}(d_x, d_y) \]

The forward motion estimation is conducted using full pel precision. For each of the resulting motion vectors \((d_x, d_y)\), the intersection point with the frame to be predicted is calculated based on the distance from the frame to be predicted to both reference frames.

**Obtaining bidirectional motion vectors for the frame to be predicted**

Next, for each motion block in the frame to be predicted, the motion vector with its intersection point the closest to the middle of the block is selected from the obtained list of motion vectors between the reference frames. The selected motion vector is split up into a forward and a backward motion vector, taking into account the distance from the frame to be predicted to the past and future reference frames.

From this stage on motion vectors with half pel precision are used. Therefore, both past and future reference frames are interpolated using the six tap Wiener filter defined in the H.264/AVC specification (Wiegand, 2003), with filter coefficients \((1, -5, 20, 20, -5, 1) / 32\).

**Bidirectional motion refinement**

![Diagram](image)

*Figure 8: The refinement search range for a motion vector is derived based on the motion vectors of its neighboring motion blocks (Ascenso, 2006).*

In this stage, the obtained bidirectional motion vector is further refined assuming a linear trajectory between the future and past reference frames. The refinement vectors for the forward and backward motion vectors are symmetric, and the optimal refinement vector is determined by minimizing the MAD between the past and future reference blocks. The refinement search range is derived based on the motion vectors of its neighboring motion blocks as shown in Figure 8.

A hierarchical coarse-to-fine approach is used: a first iteration uses large block sizes \((16\times16)\) to track fast motion reliably. In a second iteration higher precision is achieved using smaller block sizes \((8\times8)\).

**Spatial smoothing of the motion vectors**
The spatial coherence of the motion vectors is then improved using a weighted median filter. For each macroblock $MB_i$, its motion vector $MV_i$ is replaced with the weighted median of $MV_i$ itself and the motion vectors of the neighboring macroblocks (i.e. up to eight motion vectors $MV_k$). The weight for each motion vector $MV_k$ is defined as the mean squared error between the past and future reference blocks when $MV_k$ is applied to the macroblock $MB_i$.

**Motion-compensated interpolation**

Finally, the prediction for each pixel is obtained by applying conventional bidirectional motion compensation using the refined and spatially smooth motion vectors.

**Virtual Noise Estimation**

The decoder generates the side-information frame $Y$ as a prediction of the original frame $X$. The knowledge of the correlation between $X$ and $Y$ has a great impact on the coding performance. Since the difference between $X$ and $Y$ is called virtual noise, the estimation of the correlation properties is often referred to as a *virtual noise estimation*. Because we use a feedback channel, we do not need to estimate the correlation at the encoder. At the decoder, an accurate correlation model is needed for optimal reconstruction and for efficient turbo decoding.

Brites and Pereira (2008) proposed a method to estimate the correlation statistics from the information available at the decoder. In short, the online noise estimation can be described as follows. At the decoder, the side information is generated as an interpolation from two motion compensated reference frames (see before). If the difference between these two frames is small, it is assumed that the motion estimation was accurate and that the generated side information is likely to be very similar to the original signal. On the other hand, if the difference between motion-compensated reference frames is big, it is assumed that the motion estimation failed and the generated side information is not an accurate prediction of the original signal. The authors of this chapter have proposed some refinements to this method (Škorupa, 2008a), and recently also proposed to take quantization noise into account when estimating the virtual noise (Slowack, 2009).

The online noise estimation provides a correlation between the corresponding transformed coefficients (or between the corresponding samples in pixel domain) from the side information and the original frame. This correlation is used in the reconstruction function as is. To use it in the turbo decoding, bit-error probabilities for each bit in each particular bitplane can easily be inferred from it.

**Reconstruction**

The turbo decoder outputs the quantization bins in which each coefficient is located. Given these quantization bins and the side information $Y^T$, the reconstruction step selects the most likely value for the original coefficient $w_k$.

Reconstruction is the equivalent of inverse quantization in a traditional video codec. However, in DVC we have calculated side information, which is a non-quantized approximation of the
original frame. We can use this extra information to achieve a more accurate reconstruction compared to conventional inverse quantization.

Denote the lower bound of the decoded quantization bin \( L \) and its higher bound \( H \). A basic approach for reconstruction, as used in the Stanford codec, could be the following:

\[
    w_k' = \begin{cases} 
    L & \text{if } y_k \leq L \\
    y_k & \text{if } L < y_k < H \\
    H & \text{if } H \leq y_k 
    \end{cases}
\]

However, this approach is suboptimal. For an optimal reconstruction (Kubasov, 2007b), \( w_k \) should be reconstructed as the expectation \( \mathbb{E}[W_k \mid q_k', y_k] \) of the random variable \( W \) given the quantization interval \( q_k' \) and the value of the side information \( y_k \).

Since the parameters of the conditional distribution \( P(W_k \mid Y_k = y_k) \) have been calculated or estimated to model the virtual noise, \( \mathbb{E}[W_k \mid q_k', y_k] \) is the center of the mass of \( P(W_k \mid Y_k = y_k) \) inside the bin \( q_k' \). Hence:

\[
    w_k' = \frac{\int_L^{q_k'} w \cdot P(W_k = w \mid Y_k = y_k) dw}{\int_L^{q_k'} P(W_k = w \mid Y_k = y_k) dw}
\]

**FLEXIBLE DISTRIBUTION OF COMPUTATIONAL COMPLEXITY THROUGH HYBRID PREDICTIVE-DISTRIBUTED VIDEO CODING**

One of the challenges of video streaming scenarios is coping with the bandwidth requirements of the networks and the heterogeneity of the devices. Devices with different characteristics are performing video compression and streaming, ranging from high-end servers to PDAs and mobile phones. In addition, since many systems are multi-tasking systems, available computational complexity is often non static. Furthermore, some devices are battery-constrained, and if computational complexity is decreased, techniques such as dynamic voltage scaling can be used to extend battery lifetime (He, 2005).

As such, besides rate and distortion, available computational complexity is considered an important parameter in a video coding system. This prompted methods for rate-distortion-complexity (RDC) optimization, which have been developed primarily for predictive video coding with encoder-side motion estimation (Kaminsky, 2007; Ates, 2006; Stottrup-Andersen, 2004). However, only encoder complexity is considered, and there is no way for the decoder to take over some of the workload. For DVC systems, some complexity analysis has been provided, e.g. by Liu et al. (2007). However, reducing decoder complexity is not considered.
We can summarize by stating that current solutions for RDC optimization do not allow motion estimation to be shifted between encoder and decoder, but only allow encoder complexity to be decreased. Dynamic aspects such as multi-tasking, variable power-supply and session mobility need systems that adapt better to the changing conditions, by distributing complexity between encoder and decoder according to the amount of resources available at both devices. At one time instance, the encoder device could have more resources available than the decoder device but at a later point in time this could be the other way around.

Therefore, we let the encoder and decoder share the complex task of motion estimation, by developing several modes for coding inter frames. While the basic video coding architecture developed in the previous section remains the same, we introduce a mode-dependent part responsible for motion estimation. In addition, we extend the codec by coding the residual between the original frame \( W \) and a prediction \( Z \) generated at both encoder and decoder, instead of coding the original frame \( W \). Such a residual approach has proved to be advantageous also for DVC (Aaron, 2006).

The mode-dependent part features several modes of operation for coding inter frames: the predictive mode with all motion estimation at the encoder side, the DVC mode with all motion estimation at the decoder side, and the hybrid modes which share motion estimation between encoder and decoder. Two variants for implementing the hybrid modes are identified. A first approach is to apply a spatial partitioning technique and predict some of the macroblocks in a frame at the encoder while estimating the remaining macroblocks at the decoder. In the second approach, the motion search algorithm is split in two parts, i.e., the encoder calculates coarse motion vectors which are further refined by the decoder.

We provide complexity analysis for the mode-dependent part, by counting the number of pixel read and write operations. These theoretical results will be matched later on by practical measurements, illustrating how complexity is distributed between encoder and decoder in each mode. Finally, since we define the modes on a frame-by-frame basis (i.e., the coding mode for coding a particular frame does not depend on previous coding modes), we can combine different modes per set of frames, and develop a strategy for achieving fine-grain sharing of motion estimation complexity between encoder and decoder.

**Description of the codec**

At the encoder, the frame sequence is partitioned into key frames \( I \) and Wyner-Ziv (WZ) frames \( W \), as shown in Figure 9. \( I \) frames are coded using H.264/AVC intra coding. Decoded intra frames \( I' \) are available anyway after intra coding due to mode decision and rate-distortion optimization, hence, \( I' \) frames are stored in the *decoded I frame buffer*. For each \( W \) frame, a certain prediction \( Z \) is generated (that will also be generated at the decoder side). The residual \( R \) between \( W \) and \( Z \) is calculated and \( Z \) is stored in the *prediction frame buffer*. \( R \) is WZ coded using the WZ coder described earlier.

At the decoder, key frames are decoded into \( I' \) and stored in the *decoded I frame buffer*. For each \( W \) frame, side information \( Y \) is generated as well as the mutual prediction \( Z \). The residual \( Y^R \) between \( Y \) and \( Z \) is transformed and used as side information for the WZ decoder. \( Z \) is added
to the result \( R' \) to obtain the decoded frame \( W' \). For future reference, \( W' \) is stored in the decoded \( W \) frame buffer.

Figure 9: Codec architecture consisting of a WZ codec, an intra codec, a mode-dependent part and a mechanism for buffering frames.

Mode-dependent part in the predictive video coding mode
In predictive video coding mode, motion estimation is performed solely at the encoder. Each inter frame \( W \) is partitioned into macroblocks of size 8-by-8, and motion vectors are calculated for each block using bidirectional motion estimation. More precisely, the prediction frame buffer and decoded I frame buffer are consulted and the closest past frame \( P \) and future frame \( F \) are retrieved. For each macroblock \( MB_i \) in \( W \), the best match of full pixel precision in \( P \) is determined using a search window of size \( S \), where the size of the search window indicates the number of macroblocks in the search space. Only the luma component is used in motion estimation. The best match is found by minimizing a lagrangian cost function:

\[
Cost_i(\vec{v}) = D_i(\vec{v}) + \lambda R_i(\vec{v}),
\]

where the distortion metric \( D_i(\vec{v}) \) is the Sum of Squared Errors (SSE) between \( MB_i \) and the macroblock in \( P \) defined by the motion vector \( \vec{v} \). \( \lambda \) is the Langrange multiplier and \( R_i(\vec{v}) \) represents the rate to code \( \vec{v} \). Motion vectors are coded by first predicting them from their neighbors as in H.264/AVC, and coding the residual between \( \vec{v} \) and its prediction using signed exponential Golomb coding, resulting in \( R_i(\vec{v}) \) bits.

After motion estimation between \( W \) and \( P \), for each block in \( W \), the macroblock in \( P \) with the lowest cost is interpolated with every candidate block in \( F \), and a similar cost function is used for determining the best match. This results in two motion vectors for each macroblock in \( W \). Since the motion vectors calculated at the encoder will also be available at the decoder, they are used to generate the mutual prediction \( Z \) through bidirectional motion compensation, performed on all color components.
At the decoder, Z is constructed using the received motion vectors and reference frames P and F (retrieved from the prediction frame buffer and/or decoded I frame buffer). No motion estimation is performed by the decoder, therefore, the side information Y is taken equal to Z. As such, the residual $Y^R$ between Y and Z that is used by the turbo decoder contains only zeros.

We will now analyze encoder and decoder computational complexity in terms of the number of pixel read and write operations. An overview of this complexity analysis is provided in Table 1. At the encoder, unidirectional motion search between W and P using a search window of size S requires S comparisons between a certain pixel in W and a certain pixel in P. Frames are represented in the YUV color space, with H(orizontal) by V(ertical) pixels for the Y (i.e. luma) component and H/2 by V/2 pixels for each chroma component U and V. Hence, since motion estimation is only performed on the luma component, $2SHV$ pixel read operations are performed. Motion estimation between W and F reads one pixel from P, interpolates it with a pixel in F, and compares it with a pixel in W. This requires one additional pixel read operation per pixel, i.e., a total of $3SHV$ pixel read operations. Constructing Z means interpolating best matches from P and F, requiring two pixel read operations and one pixel write operation, per pixel, for all components. Hence, $9/2HV$ pixel read/write operations are executed.

At the decoder, the motion vectors are used for constructing Z only, requiring $9/2HV$ operations.
Table 1: Pixel read and write operations executed at encoder and decoder, per W frame.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Encoder-side</th>
<th>Decorder-side</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pred.</td>
<td>ME (W, P) 2SHV 220.8</td>
<td>Construct Z 9HV/2 0.5</td>
</tr>
<tr>
<td></td>
<td>ME (W, F) 3SHV 331.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Construct Z 9HV/2 0.5</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>HV (5S + 9/2) 552.4</td>
<td>Total 9HV/2 0.5</td>
</tr>
</tbody>
</table>

| DVC    | LP filtering 2HV(L+1) 2.0 |
|        | ME (F and P) 2SHV 220.8 |
|        | Wiener interpolation 144HV 14.6 |
|        | Refinement (16x16) S^{DVC}_{R*} 512HV/M 7.3 |
|        | Refinement (8x8) S^{DVC}_{R*} 128HV/M 1.8 |
|        | Spatial smoothing 64HV 6.5 |
|        | Bidir. interpol. 18HV 1.8 |
|        | Down-sampling 3HV 0.3 |
| Total  | - - Total HV(2L+2S+105S_{DVC}/2) 262.2 |

| Spat.  | LP filtering 2HV(L+1) 2.0 |
|        | ME for S_1 HV(5S + 9/2)/2 276.2 |
|        | Wiener interpolation 144HV 14.6 |
|        | Refinement (16x16) S^{SPAT}_{R*} 256HV/M 10.1 |
|        | Refinement (8x8) S^{SPAT}_{R*} 64HV/M 2.5 |
|        | Spatial smoothing 64HV 6.5 |
|        | Bidir. interpol. 18HV 1.8 |
|        | Down-sampling 3HV 0.3 |
|        | Construct Z 9HV/2 0.5 |
| Total  | HV(2.5S + 6.75) 276.7 Total HV(2L+5S_{SPAT}+171.5) 31.9 |

| Subs.  | LP filtering 2HV(L+1) 2.0 |
|        | ME Subs. P and F 5HV/4 0.1 |
|        | Wiener interpolation 144HV 14.6 |
|        | Refinement (16x16) S^{SUB1}_{R*} 512HV/M 20.3 |
|        | Refinement (8x8) S^{SUB2}_{R*} 128HV/M 1.8 |
|        | Bidir. interpol. 18HV 1.8 |
|        | Down-sampling 3HV 0.3 |
|        | Construct Z 9HV/2 0.5 |
| Total  | HV(110 + 5S)/16 35.2 Total HV(2L+8S_{SUB1}+2S_{SUB2}+171.5) 262.2 |

Mode-dependent part in the DVC mode

In the DVC mode, no motion estimation is performed by the encoder. The mutual prediction Z equals the closest frame, past or future, that can be retrieved from the prediction frame buffer and decoded I frame buffer.

At the decoder, side information Y is generated for each inter frame W using the closest past frame P and closest future frame F, retrieved from the decoded W frame buffer and decoded I frame buffer only, since decoded frames have better quality than mutual prediction frames. Y is generated using the techniques described earlier in the detailed overview of the DVC architecture. Firstly, for better capturing the true motion field, the luma component of P and F is LP filtered by replacing each pixel by the average of a group of L = 3-by-3 pixels having this pixel as a center. This requires a total of 2HV(L+1) operations. Note that from here on, we will only provide complexity analysis for steps that are not similar to techniques analyzed earlier in this work. We refer to Table 1 for a complete overview.
Next, unidirectional block-based motion estimation is performed between the filtered versions of P and F with $\Delta_{P,F}$ pixel precision using an extended search window of size $\Delta_{P,F}$ S. $\Delta_{P,F}$ denotes the distance between P and F, with $\Delta_{P,F} = 1$ if the frames are adjacent to each other.

After obtaining the motion vectors from F to P, for each macroblock in Y the motion vector intersecting the block closest to the block center is chosen and treated as a bidirectional motion vector, i.e., the intersection point splits the motion vector into a forward vector from W to F and a backward vector W to P.

Next, both the full quality reference frames P and F, and the LP-filtered versions are upsampled to half pixel resolution using a 6-tap Wiener interpolation filter. The number of operations executed by this Wiener filter can be calculated as follows. For each frame, $3/2HV$ pixels from the low resolution frame need to be copied to the high resolution frame, requiring $9/2HV$ operations. The remaining $9/2HV$ pixels are calculated using the 6-tap Wiener filter. Hence, to filter the two original frames and two LP-filtered versions, a total of 144HV operations are required.

Using the half pixel LP-filtered reference frames, the motion vector of each block in Y is refined in two passes: first using reference blocks of size 16x16 and next using reference blocks of size 8x8. At all times, we assume that the motion vector is linear between P and F, and that it goes through the block center. The refinement window $S_{DVC}^R$ for a certain block is defined by the motion vectors of the neighboring blocks, as seen before. Due to this fact, $S_{DVC}^R$ is not constant in size. $S_{DVC}^R$ will be small (on average) if the motion vector field is smooth. On the other hand, $S_{DVC}^R$ will be large if there are a lot of discontinuities in the motion vector field. We obtained a value for $S_{DVC}^R$ to use in our complexity analysis experimentally, using the setup described further in the results section. By averaging over all sequences and rate points, a value of $S_{DVC}^R = 16$ has been obtained. Hence, the complexity of the refinement process is given by $S_{DVC}^R * 512HV/M$ operations in the first pass and $S_{DVC}^R * 128HV/M$ operations in the second pass.

After motion refinement, the motion vectors are spatially smoothed by weighted vector median filtering of the motion vector for MB$_i$ and the motion vectors of neighboring macroblocks applied to MB$_i$. Concerning computational complexity, for each of the (approximately) eight neighboring motion vectors two macroblocks need to be read, requiring a total of 16HV pixel read/write operations for this step.

Finally, the calculated motion vectors are used for bidirectional motion compensation, and the result is down-sampled to obtain the side information frame Y.

**Mode-dependent part in the hybrid predictive-distributed video coding modes**

In the hybrid modes, motion estimation is shared between encoder and decoder. Two variants for sharing motion estimation can be identified, based on spatial partitioning on the one hand and splitting of the motion estimation algorithm on the other hand.

**Hybrid video coding using spatial partitioning**
Motion estimation can be shared between encoder and decoder by applying a spatial partitioning strategy, and predicting a subset of the macroblocks in an inter frame \( W \) at the encoder, while predicting the remaining blocks at the decoder side.

A checkerboard pattern is used to partition the macroblocks of \( W \) in two subsets \( S_1 \) and \( S_2 \). At the encoder, the macroblocks in \( S_1 \) are predicted using bidirectional motion estimation, as in the predictive mode. No motion estimation is performed at the encoder for \( S_2 \). To construct the mutual prediction frame \( Z \), each block in \( S_2 \) is assigned the motion vector of its left neighbor or if it does not exist, the vector of its right neighbor. This technique adds very little complexity to the encoder, but enables us to construct a mutual prediction frame \( Z \) through bidirectional motion compensation.

At the decoder, the closest past frame \( P \) and the closest future frame \( F \) are retrieved from the buffers containing decoded frames. These reference frames are first LP filtered (as in the DVC mode). For each block in \( S_2 \), an initial motion vector estimate is generated by using the motion vectors calculated for \( S_1 \). First, for a block in \( S_2 \), the motion vectors of its neighbors in \( S_1 \) are retrieved. Consider for example a non-border block \( H \) with a certain neighbor \( A \). Since \( A \) is in \( S_1 \), it has a backward motion vector \((d_x^b, d_y^b)\) and forward motion vector \((d_x^f, d_y^f)\). This vector has been obtained through rate-distortion optimization at the encoder side, and so it does not necessarily represent the actual motion path. Therefore, we obtain an initial motion vector estimate for \( H \) by treating the backward and forward motion vector of \( A \) separately. As such, we extend the backward motion vector to \( F \) and the forward motion vector to \( P \). This is done for all neighbors \( A, B, C, \) and \( D \), resulting into eight linear motion vectors. Each of these motion vectors is applied to \( H \), and the one minimizing the Sum of Squared Errors (SSE) between the reference blocks is chosen.

This initial estimate is then used as a starting point for half-pixel motion refinement, as in the DVC mode. However, in this case a refinement window of fixed size 5-by-5 \((S^{\text{SPAT}}_R = 25)\) showed better results.

Subsequent to half-pixel refinement, bidirectional motion compensation is performed and the frame is down-sampled to construct the side info frame \( Y \). No spatial smoothing is performed, since information from neighboring blocks is already taken into account during initialization of the motion vector.

**Hybrid video coding by splitting the motion search algorithm**

A second way to combine predictive video coding techniques and DVC is to split up the motion search algorithm, i.e., restrict the decoder search space for each macroblock instead of restricting the number of macroblocks for which decoder-side motion estimation needs to be performed. This means that the encoder calculates coarse motion vectors which are further refined by the decoder. Due to the generality of this definition, hybrid modes can be constructed in several ways. In this section we use a subsampling approach.

At the encoder, frames are subsampled by averaging four pixel values at high resolution for calculating one pixel value at low resolution. Next, bidirectional motion search is performed, as
in the predictive mode, using a down-scaled search window of size $S/4$. As such, encoder-side computational complexity is reduced drastically. Subsequently, the low resolution motion vectors are coded and sent to the decoder. To create the mutual prediction frame $Z$, the motion vectors are upscaled and used for bidirectional motion compensation.

At the decoder, $P$ and $F$ are retrieved from the buffers with decoded frames and LP filtered. The decoded motion vectors are upscaled and used as an initial estimate for motion refinement. Due to subsampling to double pixel resolution at the encoder, we constrain the window for half pixel refinement and set it to a fixed size of 5-by-5 ($S^{\text{SUB1}}_R = 25$) in the first pass and 3-by-3 ($S^{\text{SUB2}}_R = 9$) in the second pass. As such, the vector is never refined more than three half pixels (which is less than two pixels, i.e., the accuracy of encoder-side motion estimation). No spatial smoothing step is performed afterward, but bidirectional motion compensation and down-sampling follows directly.

Instead of using a subsample approach, other techniques can be used. For example, a heuristic motion search algorithm such as a Three Step Search (TSS) can be split up in executing one or two steps at the encoder side while executing the remaining steps at the decoder.

**Results**

We present RD results for each mode as well as complexity results by comparing our model of complexity to measurements performed on a test system. Using these results, a method for choosing appropriate coding modes for each frame will be proposed later on.

**Rate-distortion performance of the system**

Tests have been conducted on the Mother and Daughter sequence, but the similar conclusions can be drawn for other sequences as well. An IWWW GOP structure is used, and for each sequence 101 frames are coded (25 GOP's + one closing frame). Inter frames are hierarchically coded, meaning that the sequence $I_1 W_1 W_2 W_3 I_2$ is coded and decoded in the following order: $I_1 I_2 W_2 W_1 W_3$. Four different quantization patterns are used, quantizing each coefficient from 3 to 6 bits respectively. Quantization of intra and inter frames is chosen in such a way so that the quality of the decoded frames is constant.

The H.264/AVC reference software (JM 13.2) was used to create two reference RD curves for each sequence: one using a (hierarchical) IBBI GOP structure, and one using only intra-coded frames (III).

The results (see Figure 10) show that the compression performance of the different modes is comparable. This is an advantage, since switching between modes (i.e., coding frames using possibly different modes) does not have a significant impact on the performance of the system.
Validation of complexity analysis
The complexity for each of the modes has been calculated using the number of pixel read and write operations (see Table 1). How these results should be mapped to the number of CPU cycles or milliseconds spent for coding/decoding each inter frame W depends on hardware details such as calculation speed, cache behavior etc. However, the scaling factor used to convert the number of pixel read/writes to the specific metric desired should be more or less the same for each mode. Hence, theoretical and practical metrics should show the same relationships between encoder and decoder complexity. In other words, if theoretical analysis indicates that the encoder in the predictive mode is twice as complex as the encoder in the spatial mode, then this should be observed in practice also. Therefore, we normalize theoretical complexity and measured complexity to the total complexity in the predictive mode (encoder and decoder) (see Table 2). Practical measurements have been performed on our test system, averaging values for all sequences and rate points. The results clearly show that our model for measuring complexity using the number of pixel read/writes is accurate.

Table 2: Comparing calculated and measured complexity shows that our model is fairly accurate.

<table>
<thead>
<tr>
<th></th>
<th>Calculated</th>
<th></th>
<th>Measured</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Encoder</td>
<td>Decoder</td>
<td>Encoder</td>
<td>Decoder</td>
</tr>
<tr>
<td>Predictive mode</td>
<td>&gt; 99.5%</td>
<td>&lt; 0.5%</td>
<td>99%</td>
<td>1%</td>
</tr>
<tr>
<td>DVC mode</td>
<td>&lt; 0.5%</td>
<td>47%</td>
<td>&lt; 0.5%</td>
<td>48%</td>
</tr>
<tr>
<td>Spatial mode</td>
<td>50%</td>
<td>6%</td>
<td>50%</td>
<td>8%</td>
</tr>
<tr>
<td>Subsample mode</td>
<td>6%</td>
<td>7%</td>
<td>7%</td>
<td>8%</td>
</tr>
</tbody>
</table>

Video coding with controllable complexity
The previous sections illustrated how several modes can be constructed with different distributions of complexity between encoder and decoder, with comparable rate-distortion performance of the modes. Here we will develop a strategy for choosing which coding mode to use for coding each inter frame, in order to meet encoder and decoder complexity constraints. An
example is provided to illustrate the different configurations that can be achieved by combining modes.

We assume that techniques are available to estimate or calculate the complexity available at encoder and decoder, and that these values are accurate for coding the following $K$ inter frames $W$. Denote the total complexity available at the encoder to code these $K$ frames as $K C_E$ and denote the total complexity available at the decoder as $K C_D$. We will define a method to calculate the optimal linear combination of modes that meets this constraint. The method will be optimal in the sense that the total complexity of the system is minimized.

To code one frame using mode $m_i$, a complexity budget of $M_i E$ is needed at the encoder, and a budget of $M_i D$ is needed at the decoder. From the $K$ inter frames, $\alpha_i$ frames will be coded using mode $i$. Hence, this optimization problem can be formulated as follows:

Minimize: $\sum_i \alpha_i (M_i E + M_i D)$

subject to: $\sum_i \alpha_i = K$, $\sum_i \alpha_i M_i E \leq K \cdot C_E$, $\sum_i \alpha_i M_i D \leq K \cdot C_D$, and $0 \leq \alpha_i \leq K$, $\forall i$.

Remark also that one could favor encoder or decoder complexity decrease, by introducing a weighing factor $\delta$ in the cost function: $\sum_i \alpha_i (M_i E + \delta M_i D)$.

This problem can be solved, for example, exhaustively or by using integer linear programming techniques. In the following section, we will use a graphical exhaustive method to find a solution.

We will illustrate how to choose coding modes by means of an example for a particular set of parameters ($K$, $C_E$, $C_D$, and $\delta$). While this is only one out of many configurations, similar reasoning can be used for other parameters. As an example, we will decide which coding modes to use for coding the following $K = 3$ inter frames. We use the results from our complexity analysis (Tab. 1), and express available complexity in terms of the number of pixel read/write operations that can be performed for coding these $K$ frames. Assume for example that $C_E = 325 * 10^6$, $C_D = 225 * 10^6$, and $\delta = 1$.

Each out of three inter frames can be coded using one out of four modes, resulting into 20 possible ways to code the GOP. Each of these 20 solutions can be described by the tuple $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ indicating how the three inter frames are coded: using $\alpha_0$ times the predictive mode, $\alpha_1$ times the spatial mode, $\alpha_2$ times the subsample mode, and $\alpha_3$ times the DVC mode. Given the complexity of each mode (Tab. 1), each tuple has an associated average encoding complexity per frame $\frac{\sum \alpha_i M_i E}{K}$ and an average decoding complexity of $\frac{\sum \alpha_i M_i D}{K}$. We can use these two values as coordinates for representing the 20 solutions in a plane (see Figure 11). It is easily verified that solutions featuring only two modes $i$ and $j$ lie on a straight line connecting the solutions where $i$ is used exclusively and where $j$ is used exclusively.
From these 20 points, some are suboptimal in the sense that there exists always a better way to code the GOP, i.e., with lower or at most equal encoder and decoder complexity. These points are indicated in gray. The other points (indicated in black) are so-called pareto-optimal. These 12 pareto-optimal points provide the range for distributing complexity between encoder and decoder. We can see that (1,0,0,2) and (2,0,0,1) are not pareto-optimal, which indicates that using the hybrid modes enables more efficient distribution of complexity than combining only the DVC mode and the predictive mode. In addition, since all modes are present in the optimal set, this figure shows that no mode is redundant.

Given the encoder and decoder constraints, illustrated by the gray rectangle, from the pareto-optimal points the point minimizing the cost function is chosen. This means that in this case all three frames should be coded using only the hybrid subsample mode (0,0,3,0). Other solutions satisfying the complexity constraints are suboptimal in this context, since they do not minimize the cost function.

From this example several advantages for using our system can be identified. Firstly, our system provides a solution in case neither encoder nor decoder have enough resources to perform all motion estimation (in our example, neither the predictive mode (3,0,0,0) nor the DVC mode (0,0,0,3) satisfy $C_E$ and $C_D$). Secondly, due to the fact that there is no dependency between the modes, any frame can be coded using any mode at any time. As a consequence, it is possible to adapt rapidly (with a maximum delay of $K$ frames) to varying complexity constraints that can be imposed by devices with variable power supply, or by systems featuring multi-tasking.

In this example we assumed a very short time during which encoder and decoder complexity constraints remain constant. In practice, however, complexity constraints are likely to be constant over a larger period of time. As $K$ becomes larger, distribution of complexity can be performed more subtle, since the pareto-optimal set will contain more solutions. For example, $K = 10$ results into 40 pareto-optimal solutions.

![Figure 11: Encoder and decoder complexity constraints define a set of possible combinations for coding the GOP (indicated by the gray rectangle). From this set the point minimizing the cost function can be selected.](image-url)
FUTURE RESEARCH DIRECTIONS

Despite the recent advances in the field there is still a substantial gap between the performance of DVC and traditional video coding. This should not be a surprise considering steadily improving traditional video coding in the past 20 years and the first DVC architectures proposed only in 2002. However, it is crucial that DVC soon become competitive to traditional coding in the terms of coding efficiency, because only then it can draw enough attention from the industry and in turn keep its attention from the research community.

In the rest of this section we will discuss several fields of interest in DVC where the biggest improvements to the coding efficiency can be achieved. These areas are subject to a considerable research effort lately, but because the problems at hand which are far from trivial it is difficult to predict when or whether DVC will achieve a performance competitive with traditional video coders.

Side-information generation

The generation of side information is the cornerstone of a DVC system. Increasing the accuracy of the side information decreases the number of errors that have to be corrected, and thus effectively reduces the bit rate. At the same time, through the active use of side information in the reconstruction process, its accuracy has a direct impact on the distortion in the decoded video. Therefore it is crucial to generate side information with as high as possible accuracy.

The current state-of-the-art techniques for side-information generation are based on the motion-compensated interpolation. As such these techniques are facing difficulties when dealing with effects like non-linear motion or occlusions. These problems are even more pronounced when longer GOPs are used. Because the efficiency of generating the prediction with the current techniques deteriorate at GOP length as short as 8 (Pereira, 2007), in order to make DVC competitive in the practical scenarios it is important that the problem of long GOP is addressed. This can be done either by improving the interpolation techniques or by developing an effective extrapolation scheme (Borchet, 2007). While there are results indicating that long GOP size should not be feared, it is still not common to use significantly long GOPs in current literature.

Despite the obvious problems of performing the motion estimation at the decoder without knowledge about the original video signal in DVC, this design brings one considerable advantage—no motion vectors have to be signaled in the video stream. This opens the door for techniques which are out of limits of traditional block-based standards like shape-adaptive motion estimation or dense motion fields. Clearly, in DVC it is much easier to use the motion vector for an arbitrary part of the frame. Techniques for estimating the optical flow (Beauchemin, 1995) produce a motion vector for each pixel in the frame, which, in contrast to the block-based approach, is more suitable for adapting to the shape of the moving objects and to certain types of motion like rotation, zooming or deformation. As promising as these techniques might sound, so far, no significant results have been reported in this field.
Another two advantages of DVC are that the prediction signal can be adjusted at any time during the decoding process and that the channel decoder can make use of some knowledge about the accuracy of the prediction. The former allows the decoder to regenerate better side information after the frame is partially decoded and use this more accurate side information through the rest of the decoding process. The latter advantage allows multiple prediction signals or probabilistic models for the motion field (Varodayan, 2008).

To sum up, DVC offers unlimited flexibility in generation and usage of the prediction signal. This stands in strong contrast to traditional video coding, where the usage of the prediction signal is prescribed by standards. It is now the role of the research community to grasp the offered possibilities and use them to improve the coding efficiency of DVC.

**Estimating virtual channel noise**

One of the assumptions in the Slepian-Wolf and Wyner-Ziv theorems is that the joint probability distribution of the two correlated sources is given. In other words, the rate regions described in the theorems are achievable only if the knowledge about the correlation between the sources is available. That means that in DVC one has to estimate the correlation between the original signal and the side information, i.e., estimate the noise in the virtual channel. Moreover, deviations from the true correlation will decrease the coding performance.

As explained earlier in this chapter, knowledge about the virtual-channel noise is required at the encoder for rate control. A common way to avoid the necessity of noise estimation at the encoder side is using a feedback channel. While not hindering the coding performance, the feedback channel is not practical or even possible in certain application scenarios. On the other hand, doing the noise estimation at the encoder would require generating side information which would countermand the low-complexity encoding. To avoid generating side information at the encoder one can estimate certain properties of the noise from different (motion) characteristics of the video sequence—mostly used are the difference between successive frames, and the difference between the frame and an interpolation of neighboring frames (Martinez, 2008). All these methods sacrifice some RD performance and encoder complexity in order to remove the feedback channel.

The biggest challenge in noise estimation at the decoder side is the lack of the original video signal. Therefore, the estimated correlation between the original signal and the predicted signal will always be a mere approximation of the true correlation. Moreover, accurate correlation estimation has to deal with dynamically changing properties of the virtual noise. Clearly, the accuracy of the prediction depends on the amount and complexity of motion in the video sequence. Because motion can change abruptly in both temporal and spatial directions, so can the properties of the correlation between the original video and the side information. Advances in this field have proved that considerable improvements can be achieved by better modeling of the virtual noise at the decoder (Brites, 2008; Škorupa, 2008a).

Because the noise is determined by the side-information generation, one can expect that as techniques for generating side information are improving, the noise estimation will (have to) evolve as well.
New coding modes
As we discussed earlier, statistical properties of a video signal are strongly spatially and temporally varying due to motion in the sequence. Traditional coders deal with these variations by switching between various coding modes on a block basis. Indeed, coding gain achieved between successive generations of coders is contributed greatly by increasing the number of coding modes. In this light it is obvious that the frame-based approach with only two modes (I and W) as described earlier will not perform optimally and the need for different coding modes becomes clear.

While the PRISM codec offered a block-based approach with several coding modes, it—somewhat unexpectedly—lost attention in the scientific community and the codec of Stanford University has become the most important architecture for DVC nowadays. But also in the latter architecture new modes can be introduced. This, however, requires abandoning the frame-based approach or tampering with the channel code. An example of such an approach is to allow intra-coded blocks also in WZ frames (Tagliasacchi, 2006b). A second approach was taken by the authors of this chapter (Mys, 2009). Here, a notion similar to skipped blocks in traditional coding was introduced by adjusting the puncturing and the turbo decoder appropriately. In sequences with low motion, i.e., sequences where skip mode would be used extensively by a traditional coder, this concept can reduce the bit rate up to 52%.

Introducing new coding modes may require significant changes to the codec design, especially for architectures based on the Stanford codec. This, however, may turn into an advantage to DVC, because only constant probing for a change can assure that DVC will keep evolving and improving.

CONCLUSION

In this chapter, we addressed the concept of distributed video coding which is currently emerging as a new video coding paradigm allowing the construction of ultra-low complex video encoder at the expense of a more complex decoder. The theoretical foundations of DVC were discussed briefly after which an overview was given of existing DVC solutions and architectures. One of these architectures was used as reference for a more in-depth discussion of the functional building blocks of a DVC system. As computational complexity plays an important role in the context of DVC, the latter DVC system was extended with a number of coding modes allowing to dynamically shift the complexity between encoder and decoder, facilitating the requirements of emerging video communication applications. Finally, we provided an outlook to some future research directions for which it is believed that advances in these domains will contribute to the overall coding performance of DVC systems.

REFERENCES


