Advanced Heterogeneous Video Transcoding

Eduardo Peixoto Fernandes da Silva

Queen Mary, University of London

School of Electronic Engineering and Computer Science

Supervisor: Prof. Ebroul Izquierdo

Thesis submitted to University of London in partial
fulfillment of the requirements for the degree of
Doctor of Philosophy

PhD Thesis

Abstract

Video transcoding is an essential tool to promote inter-operability between different video communication systems. This thesis presents two novel video transcoders, both operating on bitstreams of the current H.264/AVC standard. The first transcoder converts H.264/AVC bitstreams to a Wavelet Scalable Video Codec (W-SVC), while the second targets the emerging High Efficiency Video Coding (HEVC).

Scalable Video Coding (SVC) enables low complexity adaptation of compressed video, providing an efficient solution for content delivery through heterogeneous networks. The transcoder proposed here aims at exploiting the advantages offered by SVC technology when dealing with conventional coders and legacy video, efficiently reusing information found in the H.264/AVC bitstream to achieve a high rate-distortion performance at a low complexity cost. Its main features include new mode mapping algorithms that exploit the W-SVC larger macroblock sizes, and a new state-of-the-art motion vector composition algorithm that is able to tackle different coding configurations in the H.264/AVC bitstream, including IPP or IBBP with multiple reference frames.

The emerging video coding standard, HEVC, is currently approaching the final stage of development prior to standardization. This thesis proposes and evaluates several transcoding algorithms for the HEVC codec. In particular, a transcoder based on a new method that is capable of complexity scalability, trading off rate-distortion performance for complexity reduction, is proposed. Furthermore, other transcoding solutions are explored, based on a novel content-based modeling approach, in which the transcoder adapts its parameters based on the contents of the sequence being encoded.

Finally, the application of this research is not constrained to these transcoders, as many of the techniques developed aim to contribute to advance the research on this field, and have the potential to be incorporated in different video transcoding architectures.

Acknowledgements

First of all, I am particularly grateful to my advisor, Prof. Ebroul Izquierdo, and to Dr. Toni Zgaljic and Dr. Tamer Shanableh, who have worked closely with me on some of the topics of my thesis. This work could not have been completed without you.

I would also like to thank Paulo V. Borges, who helped so much in my first weeks in London, and my friends in the MMV Lab, in particular Giuseppe Passino, Stefano Asioli, Vlado Kitanovski, Michele Sanna, Naeem Ramzan, Krishna Chandramouli, Tomas Piatrik, Saverio Blasi, Sertan Kaymak, Heng Yang (Chris), Ivan Damjanovic, Markus Brenner, Fiona Rivera, Juan Caicedo, Konstantinos Bozas, Xavier Sevillano, Anil Aksay, Sander Koelstra, Muhammad Akram, Lasantha Seneviratne, and, of course, my "thesis friends", Navid Hajimirza, Virginia Fernandez Arguedas and Cristina Romero Macias. A special thanks to Saverio and Virginia, who helped a lot proof-reading the thesis and giving valuable insight on how to present the data.

I must also thank those who helped me to get to London in the first place, my M.Sc. advisor, Prof. Ricardo L. de Queiroz, and all the people from the GPDS lab, and with whom I had several discussions that helped to shape this thesis: Alexandre Zaghetto, Bruno Macchiavello, Renan Utida, Fernanda Brandi, E. Mintsu Hung, Diogo Garcia, Rafael Galvao, Tiago Alves and Camilo Dorea.

Finally, I must thank my family (who are too many to cite), and a very special thanks to my wife, Ana Carolina, and son, Leonardo.

Contents

List of tables

List of figures

List of Abbreviations

B_n^k	A image block at position k at frame n.
$mv_{n \to n-\beta}^k$	the motion vector for the k-th block at frame n that uses as reference the frame $n - \beta$
4CIF	Four Common Intermediate Format - Video resolution of 704×576 pixels.
AMP	Asymmetric Motion Partition
AVC	Advanced Video Coding
B	Bi-Predictive
BAMVR	Block-Adaptive Motion Vector Re-sampling
BD-bitrate	Bjøntegaard Delta bitrate
CABAC	Context-Adaptive Binary Arithmetic Coding
CBP	Coded Block Pattern
CIF	Common Intermediate Format - Video resolution of 352×288 pixels.
CU	Coding Unit
DCT	Discrete Cosine Transform
DPCM	Differential Pulse Code Modulation
EPZS	Enhanced Predictive Zonal Search
FDVS	Forward Dominant Vector Selection
FGS	Fine Granular Scalability
FPS	Frames per Second
FRExt	Fidelity Range Extensions
GOP	Group of Pictures
H.263	Standard for Video Coding

H.264/AVC ..... Standard for Video Coding

HEVC ..... High Efficiency Video Coding

HM4.0rc1 ..... Reference Software for the HEVC version 4.0

I ..... Intra

IBBP ..... Coding configuration consisting on coding the first frame as intra and all the remaining frames as groups of 3, being the first two B-frames and the third a P-frame

IDR ..... Instantaneous Decoder Refresh

IPCM ..... Intra Pulse Code Modulation

IPP ..... Coding configuration consisting on coding the first frame as intra and all others as P-frames

IPP1 ..... IPP configuration with 1 reference frame

IPP4 ..... IPP configuration with 4 reference frames

IPP5 ..... IPP configuration with 5 reference frames

ISO ..... International Organization for Standardization

ISO/IEC ..... ISO - International Electrotechnical Commission

ITU ..... International Telecommunication Union

ITU-T ..... ITU - Telecommunication Standardization Sector

JCT-VC ..... Joint Collaborative Team on Video Coding

JM 14.2 ..... Reference Software for the H.264/AVC version 14.2

LCU ..... Largest Coding Unit

LDF ..... Linear Discriminant Functions

MB ..... Macroblock

MC ..... Motion Compensation

MCTF ..... Motion Compensated Temporal Filtering

ME ..... Motion Estimation

MPEG-2 ..... Standard for Video Coding

MPEG-4 ..... Standard for Video Coding

MRVB ..... MV Composition supporting Multiple Reference frames and Variable Block sizes

MRVB-FDVS ..... MRVB based on FDVS

MRVB-TVC ..... MRVB based on TVC
MSSIM ..... Mean Structural Similarity
MV ..... Motion Vector
P ..... Predictive
PCM ..... Pulse Code Modulation
PSNR ..... Peak Signal to Noise Ratio
PT ..... Proposed Transcoder
PT-MVVD ..... Proposed Transcoder using MV Variance Distance
PT-RC ..... Proposed Transcoder using the RC Module
PTCM ..... Proposed Transcoder based on Content Modeling
PTCM-EDCT ..... PTCM using Energy of DCT Coefficients
PTCM-LDF ..... PTCM using Linear Discriminant Functions
PTCM-MVPV ..... PTCM using MV Phase Variance
PTCM-MVVD ..... PTCM using MV Variance Distance
PTCM-NDCT ..... PTCM using Number of DCT Coefficients
PU ..... Prediction Unit
Q ..... Quantization
QP ..... Quantization Parameter
RC ..... Reduced Complexity
RD ..... Rate Distortion
RT ..... Reference Transcoder
RT-EPZS ..... Reference Transcoder using EPZS
RT-FS ..... Reference Transcoder using Full Search
RT-H264 ..... Reference Transcoder using only H.264/AVC Motion Information
RT-HS ..... Reference Transcoder using Hexagon Search
RT-MVR ..... Reference Transcoder based on MV Reuse
SAD ..... Sum of Absolute Differences
SNR ..... Signal to Noise Ratio
SVC ..... Scalable Video Coding
T ..... Transform

TD ..... Temporal Decomposition

TVC ..... Telescopic Vector Composition

VC-1 ..... Standard for Video Coding

W-SVC ..... Wavelet-based Scalable Video Codec

YUV ..... Colour space defined in terms of one luma component (Y) and two chrominance components (UV)

Chapter 1.

Introduction

The last twenty years have seen a revolution in video technology, production and consumption. In the early nineties, video consumption was made mainly by television broadcasting, cinema and VHS tapes, all using analog technologies. Advances in semiconductors and computer hardware, both in terms of processing power, memory and storage capabilities, have changed this, bringing video consumption to the digital era. Today, television broadcasts are digital in most parts of the world, and, in some parts, the old analog technology is being switched off (for instance, in London). In addition, DVD's, Blu-rays, digital cameras, mobile phones, tablets and computers are all using digital video.

Video compression is essential to all these applications [48, 106]. Today, there are several video coding formats available: MPEG-2 [59], MPEG-4 [88], H.263 [61], H.264/AVC [62], Dirac [22], among many others. In general, content encoded with one format can only be decoded with the appropriate decoder of that format. Thus, in order to promote inter-operability between different systems that use different formats, a change of format is needed. This change of format is called *transcoding* [9, 135, 141]. Video transcoding can also be used to change specific properties of a compressed video, such as the bitrate, resolution or frame-rate, or used in order to add desired properties to the bitstreams, such as error correction capabilities or scalability capabilities.

This thesis presents two complete video transcoders, both operating on bitstreams of the current H.264/AVC standard. In addition, this thesis presents improvements on other techniques that can be used in different video transcoders. The first transcoder works on H.264/AVC bitstreams, transcoding to a wavelet-based scalable video codec (SVC) [124]. Scalable video coding is a recent technology that enables low complexity video adaptation according to transmission and display requirements, providing an ef-

ficient solution for video content delivery through heterogeneous networks [90]. Such a transcoder can potentially enlarge the range of applications for the wavelet-SVC codec, and some of the techniques developed here can be applied in the transcoding from other hybrid DPCM/DCT codecs, such as H.263 or MPEG-2, to the wavelet-SVC codec, or even within other transcoders.

The second transcoder targets the emerging video coding standard, so called High Efficiency Video Coding (HEVC) [58], which is being developed to replace the current H.264/AVC standard. The H.264/AVC standard is very successful and it has been widely adopted, both by physical media, internet streaming services and some broadcast and cable television services. The HEVC is being developed as its successor, hence, motivating the conversion between these two codecs.

1.1. The W-SVC Transcoder Scenario

Due to rapid developments in technologies for coding and transmission, the demand for video delivery to and access from end users through heterogeneous networks is constantly increasing. In such an environment, a real-time adaptation of the video is often needed due to highly varying transmission bandwidth or user's display device capabilities. This requirement can easily be met if the video stored on the server is encoded in a scalable way [90]. In this case a real-time low-complexity adaptation to a lower resolution, frame-rate and/or quality is possible by simple parsing of the compressed embedded bitstream. Thus, the video can easily be adapted to the required transmission bit-rate and user's device preferences. However, video stored on the server is often encoded using conventional, non-scalable, coders, such as H.264/AVC. In this case, low-complexity video adaptation is not possible since the video needs to be transcoded each time it does not meet transmission requirements. An alternative solution is to store multiple instances of the same video content on the server and then transmit the instance that most closely matches transmission requirements. This is, however, an inefficient solution from the perspective of video storage.

To tackle the above issue, a non-scalable video bitstream can be converted to a scalable bitstream by transcoding. In this case transcoding is required only once and the transcoded video can then be adapted as many times as required. The most straightforward way to transcode from one format to another is to cascade the required decoder and encoder. Although this naive approach results in high quality of the transcoded

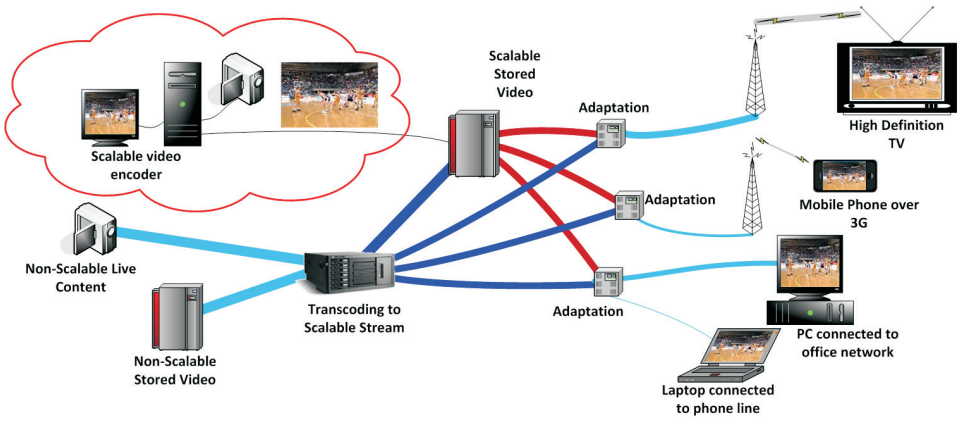


Figure 1.1.: Example of the W-SVC transcoder scenario.

video, the complexity of this conversion is also very high. One possible way to reduce the complexity of transcoding is to reuse motion information coming from the decoded video during re-encoding into another format [21]. In this thesis, this approach is used to transcode from non-scalable H.264/AVC bitstream to a scalable bitstream, produced by a wavelet-based scalable video encoder. The application scenario for such a transcoder can be seen in Fig. 1.1.

Transcoding between hybrid-based video coding structures has been extensively studied before [9, 135, 141], even targeting hybrid-based scalable codecs providing quality scalability [17, 33] and temporal scalability [10, 47]. However, hybrid-based to wavelet-based scalable video transcoding, or transcoders targeting fully scalable codecs, have not been fully investigated in the literature. Although a hybrid based approach was chosen for standardization of scalable video coding within MPEG [66], concurrently, a significant amount of research has been also carried out on wavelet-based scalable video coding. Several wavelet-based systems proposed recently have shown a very good performance in different types of application scenarios [8, 11, 60, 93, 94, 95, 124], while still being able to deliver some attractive features not supported by the standard, such as Fine Grain Scalability (FGS) and superior performance at specific coding and decoding rates.

The codec used for the research reported in this paper, denoted simply as W-SVC in the sequel, is an extended and improved version of the encoder reported by Sprljan et al. [124]. It supports quality (with FGS), spatial resolution and temporal scalability and any combination of them. Its main features include hierarchical variable size block matching motion estimation supporting motion vector scalability [91], flexible selection of filters for both spatial and temporal wavelet transforms at each level of spatio-temporal

decomposition [92, 122], user defined flexible decomposition path [89, 122], support for conventional frame-based coding and object-based coding, motion adaptive spatial filtering [123], bit-plane coding based on Embedded Zero Block Coding (EZBC) with binary arithmetic coding and low-complexity post compression rate-distortion optimization for bit-stream allocation [151].

In this thesis, H.264/AVC refers to the non-scalable part of the H.264/AVC recommendation [62] (thus without Annex G, that refers to the H.264/SVC codec [66]). Transcoding from H.264/AVC to W-SVC cannot be performed in a straight-forward way. Since the H.264/AVC codec supports reference frames selection in a very flexible way, and the W-SVC bitstream targets temporal scalability, not all information on the H.264/AVC bitstream can be directly reused in the target W-SVC. An additional key issue is that the motion information from H.264/AVC is optimized for a single rate-distortion point, and therefore it has to be modified to support the addressed scalability functionality. The proposed H.264/AVC to W-SVC transcoder consists mainly of two modules, one that is responsible on how the partitions are tested in the W-SVC, according to the partitioning found in the H.264/AVC bitstream, and the Framework for Motion Vector Approximation and Refinement [3, 4, 6]. Here, a flexible Motion Vector composition algorithm is introduced, that is able to cope with different coding configurations in the H.264/AVC stream, such as *IPP*, *IBBP* and *Hierarchical* configurations using multiple reference frames [3, 6]. The transcoder is also able to use larger macroblock sizes in the W-SVC codec, up to 64×64 pixels [5]. Furthermore, an optional module, the Reduced Complexity Module, can be used to significantly reduce the transcoder complexity, with a negligible impact on the decoded video quality [3, 4, 5, 6].

The proposed transcoder is generic in the sense that it can be used with most of the well-known wavelet-based coding architectures [8, 60, 95], particularly with any architecture that uses motion compensated temporal filtering (MCTF) [11, 31] at the same spatio-temporal resolution as the source codec. Furthermore, the transcoder framework can be adapted to any cascaded pixel domain transcoder, regardless if they are hybrid or wavelet codecs. In particular, the state-of-the-art Motion Vector Approximation algorithms presented here could be easily adapted to any transcoder where reference frame mismatch is an issue [10, 47, 55, 70, 71, 112, 117, 149]. Therefore, the transcoder is rather based on an open and easily adaptable model since the key strategies proposed for the adaptation of motion information are independent from the W-SVC coder and its underpinning algorithms.

1.2. The HEVC Transcoder Scenario

The emerging video coding standard, so called High Efficient Video Coding (HEVC) [58], is being developed by the ITU-T and JCT-VC groups to replace the current H.264/AVC standard [62]. Even before it is finalised, in a recent iteration, the HEVC outperforms the H.264/AVC by 30% to 50% [75].

The main goal of the HEVC codec is not to provide video compression with different features, such as error correction or scalability capabilities, but rather to significantly improve the rate distortion performance, compared to the current standard, H.264/AVC, in order to allow for new applications, such as beyond high-definition resolutions (so called $2K$, 3840×2160 pixels, and $4K$, 7680×4320 pixels) and improved picture quality, incorporating tools to support other colour spaces, samplings and pictures with higher dynamic range.

The HEVC is not yet a standard, but it is expected to become a standard in 2013. Therefore, the motivation for a H.264/AVC to HEVC transcoder is twofold: (i) to be ready to promote inter-operability for the legacy video encoded in H.264/AVC format, when new applications using the HEVC emerge; and (ii) to be able to take advantage of the superior rate-distortion performance of the HEVC. The first will be useful when the first applications are launched that use the new standard, while the second could be used straight away to migrate the abundant existent video content encoded in the H.264/AVC format.

In this thesis, first the performance of a modified motion vector reuse technique in a H.264/AVC to HEVC transcoder is discussed, in an attempt to identify the issues in transcoding to the new codec. Then, a transcoder based on a new method that is capable of complexity scalability, trading off rate-distortion performance for complexity reduction, is proposed [1]. The method is based on a new metric to compute the similarity of the H.264/AVC motion vectors, and it uses this metric to decide which HEVC partitions are tested on the transcoder. Finally, other transcoding solutions are explored, based on a content-based modeling approach, in which the transcoder adapts the transcoding parameters based on the contents of the sequence being encoded [2].

It is also important to notice that all techniques discussed in the H.264/AVC to HEVC transcoder are generic in the sense that its main ideas can still be used even if the HEVC changes some of its underlying algorithms, which is of particularly importance, since the HEVC is not yet finalised.

1.3. Thesis Contributions

The primary contributions of this thesis are the development of two efficient and competitive transcoders, from the H.264/AVC to W-SVC and from H.264/AVC to HEVC. The specific contributions can be divided in two parts, for each of the transcoders developed. For the W-SVC transcoder:

- development of a new Motion Vector Composition method, that is both reliable and accurate, independently from the coding structure used [3, 6];
- an efficient algorithm to optimise the reuse of the H.264/AVC partitioning in the W-SVC codec [4];
- design of an efficient framework for MV Approximation and refinement [4];
- development of new methods to further reduce the transcoder complexity, with a negligible loss in terms of decoded video quality [6];
- design of an efficient and flexible transcoder, that is able to cope with any coding configuration in the incoming H.264/AVC bitstream [6]; and
- use of different macroblock sizes to reduce the transcoder loss [5].

And for the HEVC transcoder:

- identification of the issues on transcoding to the new HEVC codec;
- development of an efficient transcoding algorithm, capable of exchanging transcoder complexity at the cost of rate distortion performance [1];
- proposal and analysis of different features used to map H.264/AVC to HEVC partitions [1, 2];
- design of a content-based algorithm, that is able to adapt to the current sequence being transcoded [2].

1.4. Thesis Organisation

This thesis is organised as follows:

Chap. 2 presents an overview on video coding and transcoding methods, as well as a survey on video transcoding. Important video coding concepts, such as motion estimation and compensation, and rate-distortion optimisation, are explained, as they will be important in the later chapters. Also, several well-known transcoding functionalities and techniques are presented, for both homogeneous and heterogeneous transcoding.

Chap. 3 presents an overview of the three codecs that are used in this thesis. On the H.264/AVC standard, the chapter covers mainly the tools used in the Main and High profiles, since these are the profiles that are more likely to be used in the transcoding scenarios. A more detailed study on the motion estimation part is presented, because this will play a major role in both transcoders. An overview of the HEVC codec, specifically the version $HM4.0rc1$, is also presented. Finally, the main tools of the Wavelet-based scalable video codec used are presented, also with a special attention to the motion estimation and mode decision modules.

Chap. 4 discusses one of the main transcoding techniques, the motion vector reuse. It also presents the state-of-the-art in motion vector approximation techniques, along with the proposed Motion Vector Composition method. A thorough evaluation of all these techniques, both independent from the specific codecs used in the transcoder, focusing on reliability and accuracy, and within the scope of the H.264/AVC to W-SVC, is presented.

Chap. 5 presents the proposed transcoder from H.264/AVC to the W-SVC. The main issues to be solved in the transcoder and how these issues are tackled are presented. It also presents both the experiments made to evaluate the transcoder and the results for these experiments, along with a complexity analysis for the transcoder, and a brief discussion of these results.

Chap. 6 presents the proposed H.264/AVC to HEVC transcoder. It discusses the performance and limitations of the motion vector reuse on this type of the transcoder, as well as a study on several transcoding options to the HEVC codec. It also presents an analysis of different features used for the content-based transcoding approach, along with the results and its discussion.

Chap. 7 concludes the work presenting the final considerations about the results so far. It also contains some directions for future research.

This thesis has been organised in a self-contained manner. The initial chapters present the fundamental aspects of the addressed technology and the corresponding state of the art, while the subsequent chapters present the proposed solutions, fully detailing the algorithms and both transcoders developed. The last chapter concludes the work, presenting the final findings and considerations for future research. Finally, the most important algorithms used on the thesis, but developed by other authors, are presented in the Appendix, alongside the full results for the experiments with the W-SVC transcoder and a description of the video sequences and the quality metrics used.

Chapter 2.

Video Coding and Transcoding Concepts

The study of video transcoding relies on a deep understanding of some aspects of video coding techniques. Throughout the thesis, different video codecs are used, including a codec that is very different from the usual hybrid DPCM/DCT model. For this reason, this chapter provides a background on video coding concepts, detailing the basic tools of a video codec and introducing the standard classification of prediction, transform and entropy coding.

Afterwards, it discusses the concepts of a video coding *format* and a video coding *standard*, that lays the foundation for the introduction of video *transcoding* and its fundamental techniques. The chapter ends presenting the types of transcoding, its different functionalities, and a survey of the state-of-the-art.

2.1. Video Coding

Video encoders exploit the video sequence redundancies, both between frames (inter-frame redundancy), within the frame itself (image redundancy) and within the data itself (data redundancy) in order to achieve greater compression performance. One way to classify a video encoder is to divide it in three parts: prediction model, image model, and entropy coding [106], each targeted to exploit one of these redundancies. In summary, the goal of the prediction model is to generate a prediction for the image, so that only the residual is encoded. The goal of the image model is to exploit the image redundancies, generally attempting to represent the image in a sparser representation

through the use of transforms. Finally, the goal of the entropy coding part is to exploit the data redundancy in order to encode the prediction (or the parameters needed to form a prediction), and the sparser representation formed by the image model in a more efficient way.

In most video codecs, each frame is divided in non-overlapping blocks, usually called macroblocks (MB) [104, 106]. The macroblock is the unit to which all the operations are applied. The encoding process starts by encoding the first macroblock (generally, the top left macroblock in the frame, as the blocks are usually encoded in raster scan order), applying the three modules (prediction, transform and entropy coding) and, when this macroblock is finished, proceeding to the next macroblock. Video codecs that follow this approach (division in macroblocks, followed by prediction, DCT transform and entropy coding) are said to follow the *hybrid DPCM/DCT* approach [106]. The typical operations that are used to encode one macroblock are discussed next.

2.1.1. Prediction Model

The most typical prediction model used for video coding has its roots on Differential Pulse Coding Modulation (DPCM) [107]. For video compression, this technique is applied as follows: consider a subdivision of the frames in the video sequence in blocks of $N \times N$ pixels (with $N > 0$, with $N = 16$ being typically used in various codecs), where B_n^k is the k-th block of the n-th frame. When encoding a given block B_n^k, the encoder attempts to form a prediction of this block using the blocks that were already encoded. Then, a residual is formed, given by:

$$R_n^k = B_n^k - P_n^k \tag{2.1}$$

where R_n^k is the residual and P_n^k is the prediction for the block B_n^k. The closer the prediction resembles the target block, the lower the residual, to the point where if the prediction perfectly matches the current block, no residual needs to be transmitted. The rationale is that the amount of bits to encode R_n^k is much lower than what is needed to encode B_n^k, and thus compression is achieved. It is also necessary to signal which prediction, among all possible previously encoded blocks, was used (or to signal how to form this prediction), but this is usually a small amount of side information that needs

to be transmitted, compared to the amount of bits required to encode the original block B_n^k.

The most common classification for this prediction is dependent on the location of the blocks that are used to form the prediction: when they are inside the current frame, this is called intra prediction, and when they are located in different frames, it is called inter prediction. The latter plays an important role through this thesis, so it is reviewed more thoroughly.

Intra Prediction

In intra prediction, the blocks used to form the prediction are restricted to the already encoded blocks in the current frame. In general (and particularly through this thesis), the blocks are encoded in raster scan order, so the intra prediction can use the blocks above and to the left of the current block. This can be seen in Fig. 2.1(a). Different techniques can be used to form the prediction: it can be formed by the average value of the pixels in the border, or by using some kind of bilinear interpolation from the neighbouring pixels [106]. An example of intra prediction can be seen in Fig. 2.1(b), where the prediction is formed by propagating the pixel values in the left border of the current block.

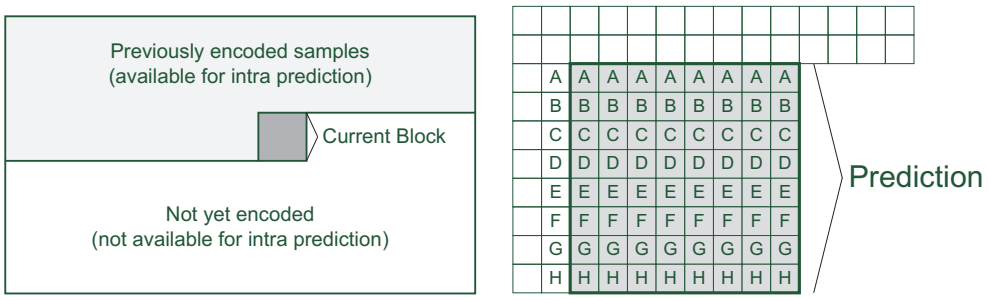


Figure 2.1.: Intra prediction: (a) blocks available for intra prediction; and (b) example of intra prediction.

Inter Prediction

In inter prediction, the prediction is formed using blocks from different frames other than the current frame [67, 106]. These frames can be either past or future frames. For

a given block B_n^k, located at pixel coordinates x_0 and y_0 (the top-left pixel of the block) and of size $w \times h$, the prediction is formed using the equation:

$$P_n^k(x,y) = f'_{n-\beta}(x_0 + x - dx, y_0 + y - dy) \tag{2.2}$$

where x and y represent the pixel coordinates with $0 \leq x < w$ and $0 \leq y < h$, $f'_{n-\beta}$ represents the $n-\beta$-th reconstructed frame, dx and dy represent the displacement vector and P_n^k represents the prediction. Given that the previously encoded frames are already available, the prediction is completely described by three parameters: the displacement vector (dx, dy) and the reference frame index $n-\beta$. These parameters are usually considered together, in an entity called *motion vector* (MV). Here, and throughout this thesis, a motion vector is referred as $mv_{n\to n-\beta}^k$, which represents the displacement vector for the k-th block at frame n that uses as reference the frame $n-\beta$. This is shown in Fig. 2.2, using as an example $\beta = 1$. Notice that the frames used for prediction are the reconstructed frames, not the original frames, in order to avoid drift (the desynchronization between the encoder and the decoder caused by the use of different predictions).

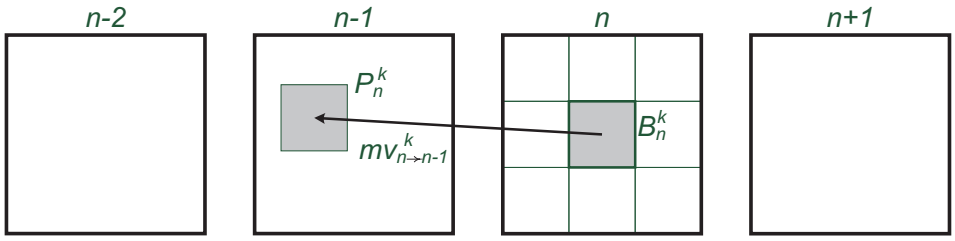


Figure 2.2.: Example of inter prediction.

In order to further improve the quality of the prediction, it is common to generate an interpolated frame at a higher resolution, and then use motion vectors at fractional pixel coordinates, in a process called sub-pixel motion estimation [49]. Many codecs use interpolation at quarter-pixel level, as a trade off between increasing the side information associated with transmitting the motion vectors and improving the prediction [62, 89, 121]. The interpolation filter is usually fixed, so it does not have to be transmitted as side information.

The inter prediction process is divided in two parts: motion estimation, which is the process by which a motion vector is selected, and motion compensation, which is

the process that actually generates the prediction given a motion vector. The optimal motion estimation method is the full search, which tests all motion vectors in a given search window (which is usually an encoder parameter), selecting the one that yields the lowest cost. This is optimum in the sense that it can find the best motion vector, but it is also very slow, as it involves computing the cost for a large number of motion vectors. Several fast motion estimation methods have been proposed in the literature [132, 153, 154], that attempt to reduce the number of motion vector candidates that are tested, thus reducing the complexity. These methods are significantly faster than full motion estimation, but they are also sub-optimal in the sense that they cannot always find the optimal motion vectors, often converging to a local minimum. Two of the most common fast motion estimation methods are described in the Appendix A, as they are used in this thesis.

In order to select among all possible motion vectors in a motion vector candidate list, there needs to be a metric to evaluate them. One could attempt to find the best possible prediction (i.e., the one that minimizes the residual), using a given distortion metric. Some distortion metrics that are common are the sum of absolute differences (SAD) and the sum of square differences (SSD), given by equations 2.3 and 2.4, respectively.

$$SAD = \sum_{i=0}^{w} \sum_{j=0}^{h} \left| B_n^k(i,j) - P_n^k(i,j) \right| \tag{2.3}$$

$$SSD = \sum_{i=0}^{w} \sum_{j=0}^{h} \left(B_n^k(i,j) - P_n^k(i,j) \right)^2 \tag{2.4}$$

One possible algorithm would be to compute the cost of each motion vector in the list and select the one with the lowest cost. However, this accounts only for the quality of the prediction, not for the additional side information that needs to be transmitted (i.e., the motion vectors).

Rate Distortion Optimization on Motion Estimation

In order to also account for the rate, a technique called rate distortion (RD) optimization was proposed [98, 126], and it is widely used in encoder implementations. Using this

technique, the cost function is modified in order to account for the rate of the side information. The modified cost function is given by:

$$J = D + \lambda \cdot R \tag{2.5}$$

where J is the new cost, in rate distortion sense, D is a distortion metric (both SAD, SSD or others can be used), R is the rate required to transmit the side information and λ is the Lagrange multiplier [98]. The λ parameter controls the weight that both the rate and the distortion will have in the final cost. A small value of λ prioritizes minimizing the distortion (note that, if $\lambda = 0$, the cost function is equal to the distortion), while a larger value prioritizes minimizing the rate. The value of λ is given by the encoder, and it usually changes according to the distortion metric used and with the video quality selected [106].

2.1.2. Image Model

The goal of the image model is to compress the residual generated by the prediction model in a more efficient way. To achieve this, it uses techniques that exploits the correlation that is still present in the residual. Also, a small loss may be introduced, in order to achieve higher compression ratios.

Transform Coding

The most common way to exploit the image correlation is by the use of transforms. The goal of a transform is to separate the image data, which is in the pixel domain, into compact components in the transform domain, with minimal inter-dependence [106]. The data in the transform domain is generally easier to compress, and less susceptible to quantization errors than the data in the pixel domain. While the DCT transform [105] is the most widely used, many other transforms have been proposed to be used in image and video coding, such as the Karhunen-Loève transform [105], Singular Value Decomposition [105], lapped transforms [36], the wavelet transform, with different filters [96] and even a modified integer DCT transform [83]. In general, these transforms are reversible and follow the pattern:

$$\mathbf{Y} = \mathbf{A}\ \mathbf{X}\ \mathbf{A^T} \tag{2.6}$$

$$\mathbf{X} = \mathbf{A^T}\ \mathbf{Y}\ \mathbf{A} \tag{2.7}$$

where $\mathbf{X}$ is the image block, in the pixel domain, $\mathbf{Y}$ is the transformed block, in the transform domain, and $\mathbf{A}$ is the transform matrix. The transformed block $\mathbf{Y}$ is then quantized [107] into the signal $\mathbf{Q}(\mathbf{Y})$, which is able to represent only a smaller number of values in the range of the original signal $\mathbf{Y}$. Since the values in $\mathbf{Q}(\mathbf{Y})$ have a smaller range than $\mathbf{Y}$, they can be represented using a smaller number of bits and, therefore, compression is achieved. Note that the quantization process is irreversible, i.e., once the signal $\mathbf{Y}$ is quantized into the signal $\mathbf{Q}(\mathbf{Y})$, the original pixel values cannot generally be reconstructed perfectly, which classifies the quantization as a lossy process. The coarseness of the quantizer is usually controlled by a quantization parameter (QP), that controls the quantization step size. The smaller the step size, the more accurate $\mathbf{Q}(\mathbf{Y})$ will represent $\mathbf{Y}$, however, a larger number of bits will be needed to represent $\mathbf{Q}(\mathbf{Y})$. On the other hand, the larger the step size, the less accurate $\mathbf{Q}(\mathbf{Y})$ will represent $\mathbf{Y}$, and a smaller number of bits will be needed to represent $\mathbf{Q}(\mathbf{Y})$.

2.1.3. Entropy Coding

Entropy coding is the process that exploits the redundancy in the data itself (the entropy) to compress the data. The original data can be recovered perfectly, making this a lossless process. There are several algorithms in the literature, each more suitable to a kind of data. Video codecs typically use a entropy encoder that is specifically designed for the data it is attempting to compress. For instance, the quantized transform coefficients $\mathbf{Q}(\mathbf{Y})$ are typically encoded using an arithmetic encoder or a variable length encoder [107], while other parameters such as the motion vectors are encoded using a different table of variable length codes [106].

2.1.4. Encoder and Decoder Sequences

A simplified architecture of a video encoder that follows the DPCM/DCT approach is shown in Fig. 2.3. When encoding a given macroblock B_n^k, the first step is to perform

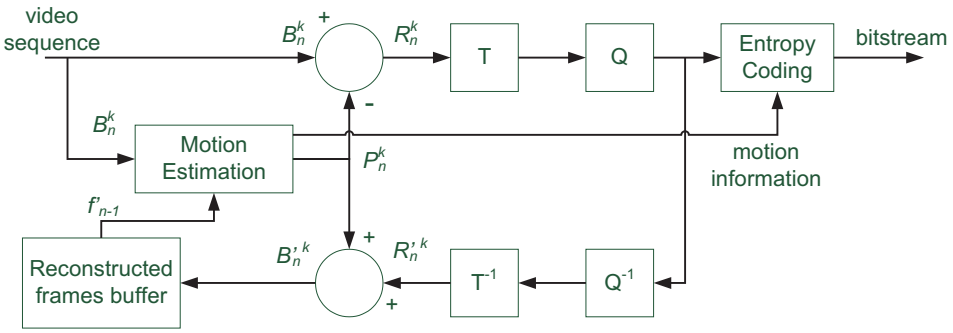


Figure 2.3.: Simplified architecture for a video encoder.

motion estimation. The output of the motion estimation process are the prediction for that block, P_n^k, and the motion information (in the form of motion vectors and reference frames). The latter is ready to be used in the entropy coding module and inserted in the bitstream. As seen in the previous sections, the residual is computed, as $R_n^k = B_n^k - P_n^k$. The residual is then ready for the transform and quantization modules, and the quantized coefficients are transmitted in the bitstream. Afterwards, there is the encoder reconstruction loop, where the operations of inverse quantization and transform are applied to the quantized coefficients, generating the reconstructed residual $R_n'^k$. This residual is added to the prediction, generating the reconstructed block $B_n'^k = R_n'^k + P_n^k$, which is then added to the reconstructed frame buffer to be ready to be used in the motion estimation for the next frame. Again, note that the reference frames used to generate the prediction P_n^k at the encoder are the reconstructed frames. The reason for this will be made clear in the following.

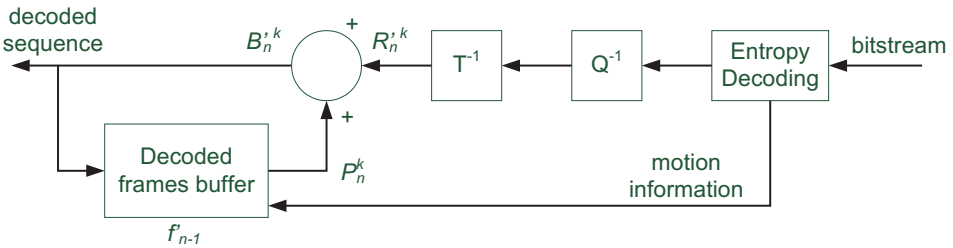


Figure 2.4.: Simplified architecture for a video decoder.

A simplified architecture of the decoder is shown in Fig. 2.4. First, the bitstream is decoded, and the operations of inverse quantization and transform are applied, generating the reconstructed residual $R_n'^k$. Also, the motion information is decoded and used to generate the prediction P_n^k, using the reconstructed frame buffer. They are then added, generating the reconstructed block $B_n'^k = R_n'^k + P_n^k$. This block is then ready to

be displayed, and also ready to be used as reference for the next frame. At the decoder, only the reconstructed frames are available. Thus, in order for both the encoder and the decoder to use the same prediction P_n^k, the encoder also needs to use the reconstructed frames.

2.1.5. Video Coding Formats

The previous sections showed a few of the techniques that can be used in a video encoder, in particular one that follows the DPCM/DCT model. However, each of these techniques can be implemented in a different way. For instance, the size of the blocks to generate the residual can vary, the motion estimation can allow for half-pixel or quarter-pixel precision for the motion vectors, or the encoder can use one of many transforms. A video coding format is a set of definitions that allow the encoder and the decoder to agree on which techniques are used, and on how to signal the parameters and compressed data inside the bitstream.

Some of the video coding formats are made into industry standards, that defines a format or syntax to the compressed bitstream (and a method to decode it) that can be used by different manufacturers. This way, a video that was encoded using a camera from one manufacturer can be decoded using the decoder made by another manufacturer, if both the encoder and the decoder implement the same standard.

There are several different video coding formats, for example: MPEG-2 [59], H.263 [61], H.264/AVC [62], HEVC [58], Dirac [22], the W-SVC [124], among others. Among these examples, some of them are international standards (MPEG-2, H.263 and the H.264/AVC), one of them is a draft for a new standard (HEVC) and some (Dirac and W-SVC) are not international standards.

In general, a bitstream that was generated using a given format can only be decoded by a decoder that is compliant with that format, and will not work with any other decoder. In order to promote inter-operability between different formats, video transcoding is essential. A short background on video transcoding technology is given in the following section.

2.2. Video Transcoding

By definition, transcoding is the process that converts from one compressed bitstream (called the source or incoming bitstream) to another compressed bitstream (called the target or transcoded bitstream) [9, 135, 141]. Several properties may change during transcoding: the video format [70, 112], the bitrate of the video [15, 127], the frame rate [71, 117], the spatial resolution [119, 146], the coding tools used (i.e., one bitstream might use B frames, while the other might not, or scalability layers are added to the target bitstream) [33], and even the insertion of new information on the video, such as watermarking [143], hidden data [110] or a layer for error resilience [113].

There are also several possible application scenarios for transcoding. One example is to deliver a high quality video content through a more restricted wireless network to be accessed by mobile phones. In this case, the spatio-temporal resolution may have to be reduced to fit the device playing capabilities and also the bitrate may have to be reduced to suit the network bandwidth. Another example is to broadcast a video content compressed in H.264/AVC format through a digital television system that uses MPEG-2 as the video format. In this case, even though the compression performance of H.264/AVC is higher than that of MPEG-2, the video must be transcoded to enable communication.

What is common among the application scenario of transcoding is that one or more characteristics of the video bitstream need to change to allow communication between the two systems [9]. To allow the inter-operation of multimedia content, transcoding is needed both within and across different formats. The usual operations on transcoding are shown in Fig. 2.5.

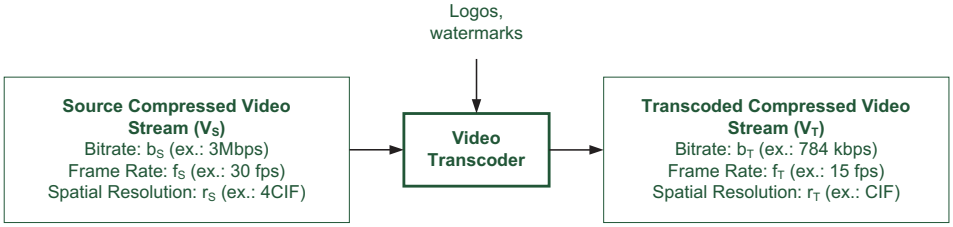


Figure 2.5.: Video transcoding operations.

In transcoding, it is always possible to use the *cascaded pixel-domain approach*, where the source bitstream is completely decoded and then re-encoded in the target conditions [135]. Through this thesis, this is defined as the *trivial transcoder*. While this approach

usually achieves high quality of the transcoded sequence and can be used for any target conditions, it is not efficient from the point of view of complexity. As Ahmad et al has summarised [9], there are three basic requirements in transcoding:

- The information in the source bitstream should be exploited as much as possible;
- The quality of the transcoded video should be as high as possible, or as close as possible as if the original video was encoded in the target format; and
- The transcoder complexity (delay, processing power and memory requirements) should be kept at minimum, targeting real-time implementation.

The transcoder performance (specially the quality of the decoded sequence) is bounded by the quality of the source stream. Compared to the quality that could be achieved if the original video sequence was encoded directly in the target specifications, instead of being encoded by the source encoder first, the transcoded sequence usually presents a lower rate distortion performance. The performance difference between the transcoded sequence and what could be achieved if the original sequence was encoded in the target specifications is referred here as the transcoder loss. One of the goals in the transcoder design is to minimize this loss.

The two main categories of transcoders are: *heterogeneous transcoding* (i.e., between different formats) and *homogeneous transcoding* (the conversion of bitstreams within the same format). Homogeneous transcoding is commonly used to change the bitstream in order to adapt it to a new functionality, such as a different bitrate or spatio-temporal resolution. Heterogeneous transcoding can also provide the functionalities of homogeneous transcoding, such as reduction of bit rate and change of spatio-temporal resolution, but it is mainly defined by the change of format. While this thesis focuses on heterogeneous transcoding, a brief introduction to the main techniques used for homogeneous transcoding is given next. Since some of these techniques may be adapted to heterogeneous transcoding, it is useful to study these techniques in addition to those of heterogeneous transcoding.

2.2.1. Homogeneous Transcoding

This kind of transcoder performs the conversion of bitstreams within the same format. Typically, it can reuse more of the information available in the source bitstream, since the

same tools used to encode the source stream are likely to be available for the transcoded bitstream [135].

There are several techniques for homogeneous transcoding, and they are often combined to achieve certain target conditions. For the sake of simplicity, they will be treated separately, classified as the target requirements: (i) bit rate reduction transcoding; (ii) spatial resolution reduction transcoding; and (iii) temporal resolution reduction transcoding. All techniques are discussed in general terms for an homogeneous transcoder for a hybrid DPCM/DCT codec. Also, a short discussion of the application of these techniques to the current H.264/AVC standard is given.

Bit rate reduction transcoding

The goal of this transcoder is to reduce the bit rate of the transcoded bitstream (compared to the source bitstream) while maintaining the highest possible quality (since the bitrate is being reduced, the quality is also likely to be reduced, however, the transcoder goal is to minimise this loss) and also keeping the complexity at the minimum. There are two main types of bit rate reduction transcoding: open-loop system and closed-loop system [135]. The former has the advantage of very low complexity, but it suffers from degraded quality due to drift, while the latter is more complex, but yields a better performance.

The open-loop architecture shown in Fig. 2.6. Comparing this architecture to the architecture of a encoder and decoder shown in Figs. 2.3 and 2.4, it can be easily seen that the source bitstream is not fully decoded. Instead, it works at macroblock level, decoding the DCT coefficients of the residual for that macroblock, performing the inverse quantizing, and then re-quantizing the coefficients using a coarser quantizer step to reach the desired bit rate [127]. This is done in the DCT domain, so even the transform operations are avoided. Finally, it encodes the new residual. This is shown in Fig. 2.6. An alternative to this design is the selection of coefficients, instead of re-quantization, discarding the higher frequency coefficients of the residual [39].

Since this architecture does not involve even DCT or IDCT operations, it has a very low complexity. The biggest drawback of this architecture is the drift [146]. After the modification of the DCT coefficients of the residual, the decoder of the transcoded bitstream will not have access to the same prediction used at the source encoder. Note that the transcoded bitstream have the quantized coefficients for $R_n^{''k}$, instead of the

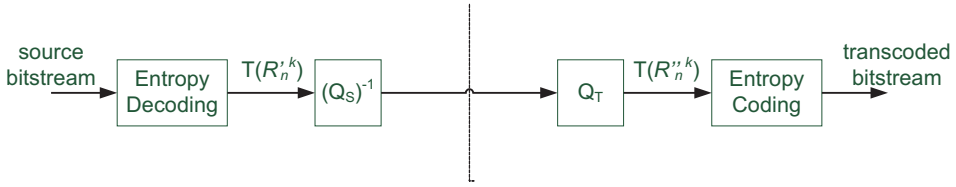


Figure 2.6.: Simplified architecture for a open-loop transcoder.

coefficients for R'^{k}_{n}. Thus, when this new residual is used at the decoder of the transcoded bitstream, that decoder will not be able to generate the prediction P^{k}_{n}, instead generating a degraded prediction $P'^{k}_{n} \neq P^{k}_{n}$. Thus, the predictions computed at the decoder of the transcoded bitstream have a lower quality, and hence a degradation in the quality of the predicted frames will occur.

In order to reduce the drift problems present in the open-loop system, closed-loop systems were proposed [14, 127]. The main difference between them is that, in the closed-loop system, there is a reconstruction loop in the transcoder to correct the residual, in an attempt to avoid drift. In this system, there is increased complexity due to the DCT, IDCT and motion compensation operations. A simplified structure is shown in Fig. 2.7. Note that, in this architecture, the difference cause by the different prediction $P'k_n$ is being compensated, thus eliminating the drift. Alternatively, the motion compensation in closed-loop system can be performed in the DCT domain [14, 26], which could yield the same quality as compensating in the spatial domain [14, 87, 128].

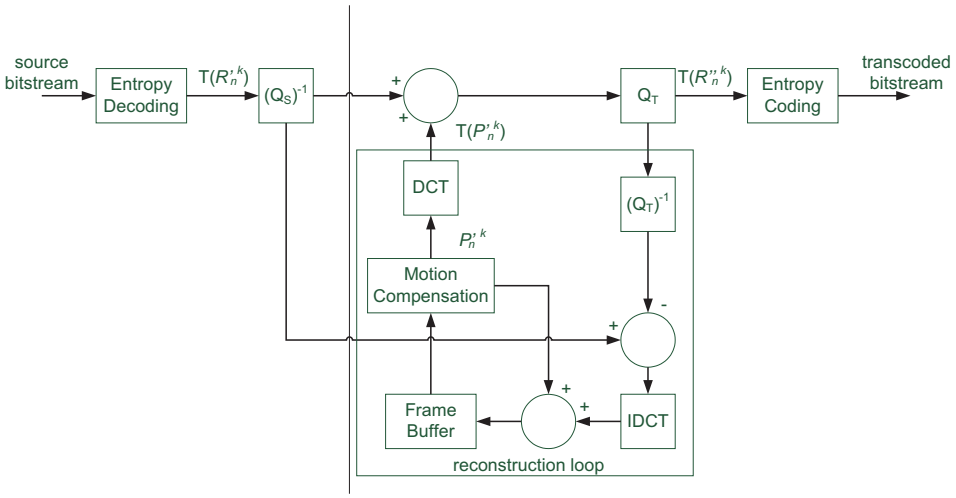


Figure 2.7.: Simplified architecture for a closed-loop transcoder.

In transcoding applications for the H.264/AVC codec, it was found that the open-loop architecture introduces an unacceptable amount of drift, particularly if applied

to intra frames [73]. For this reason, a closed-loop architecture that performs error compensation using the residuals was proposed, but its performance is still much lower than the cascaded pixel domain transcoder (which performs error compensation using the reconstructed frames) [73]. Another work [76] combines the three approaches, performing the cascaded pixel domain transcoder for the intra frames (since there are usually a small number of intra frames in the bitstream, the added complexity is small), and using a closed-loop architecture with drift compensation (using the residual blocks) for the P frames and an open-loop architecture without any drift compensation for the B frames. The results of this hybrid architecture are significant better than applying the closed-loop approach for all frames and the complexity level of these two approaches is similar, however, the cascaded pixel domain architecture still outperforms the hybrid approach by a larger margin in terms of decoded video quality [76].

Spatial resolution reduction transcoding

In this type of transcoding, the spatial resolution of the transcoded stream is lower than the resolution of the source stream. This also produces a bit rate reduction. Some key techniques are the reuse of data at the macroblock level and mapping the motion estimation to the new spatial resolution [21, 115]. Downsampling in the DCT domain is also possible [26, 120]. To avoid drifting errors, an intra-refresh architecture can also be used [146]. These techniques will be analysed separately.

When downsampling the video sequence, many macroblocks in the source stream are mapped to a single macroblock in the transcoded stream, in a technique called macroblock mode mapping (MB mode mapping). As an example, when downsampling by a factor of 2 in both spatial dimensions, and considering a macroblock of 16×16 (which is usual in DPCM/DCT codecs), 4 macroblocks are mapped in a single macroblock, as it can be seen in Fig. 2.8. In this case, if the target codec allows motion vectors for 8×8 blocks, a $1:1$ mapping can be performed. Otherwise, the transcoder has to decide which motion vector to use in the transcoder, and a $4:1$ mapping must be used. Many techniques were proposed to map these motion vectors: median [21], weighted average and mean [115] are some of them. Several of these methods were also investigated in the context of arbitrary downsampling, where a $M:N$ mapping needs to be performed (where M and N depend on the source and target resolutions) [79].

The spatial resolution reduction can also be made in the DCT domain, in a technique referred as DCT-Domain down conversion. The main idea of this technique is to keep

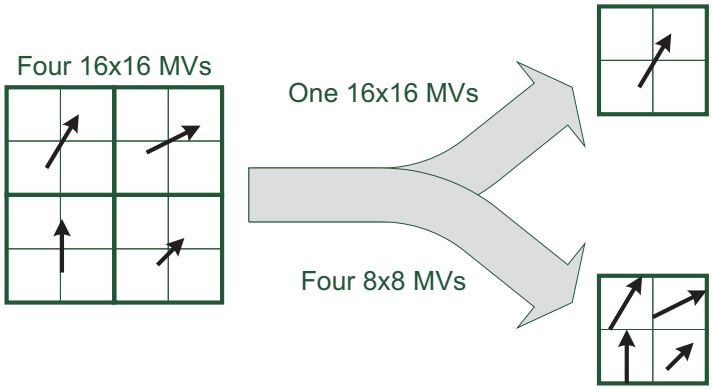


Figure 2.8.: Illustration of macroblock and motion vector mapping.

only the low-frequency coefficients of the macroblocks in the source stream and then recompose them in the macroblock for the transcoded stream. If the DCT used is 8×8, the macroblock is 16×16 and a downsampling by a factor of 2 is considered, then this would keep only the 4×4 coefficients in the first quarter of each macroblock. A composition technique was proposed to combine the coefficients [26], and some more refined filters were also proposed [120].

To avoid drift errors that may arise due to requantization, motion vector mapping, DCT down sampling and other sources, an intra-refresh technique [135] may be used. In this technique, some number of inter macroblocks in the source stream are transcoded as intra mode. For these blocks, all drift is avoided. Selecting which macroblocks are going to be converted to intra mode is a key factor in the efficiency of this technique. One possibility is to use some information (the residual energy, the motion vectors and the picture statistics) to drive this choice, selecting the macroblocks that are most likely to contribute to a larger drift error [146].

Many transcoding applications targeting spatial resolution reduction using the H.264/AVC standard have been reported [129, 152]. A transcoder based on mode mapping was proposed [152], where only a $4:1$ mapping is investigated using a cascaded pixel domain architecture. Another work [129] proposes to re-estimate the motion vectors for the new spatial resolution using an area weighted median filter. This is used in the context of H.264/AVC to H.264/AVC transcoding with spatial resolution reduction that also uses the cascaded pixel domain approach. In this work, downsampling by an arbitrary factor is also investigated.

Temporal resolution reduction transcoding

In this type of transcoding, some frames of the source stream are dropped and the transcoded stream has a lower frame-rate than the source stream. As in the spatial reduction, this also causes a reduction in bit rate. The biggest problem with this type of transcoding is that the video stream is usually strongly dependent on the past frames (by using predictive frames), and this frame-dropping is likely to break this chain of dependency. This is shown in Fig 2.9. There are two major techniques that can be used in this case: motion vector re-estimation [57, 149] and residual re-estimation [26, 45].

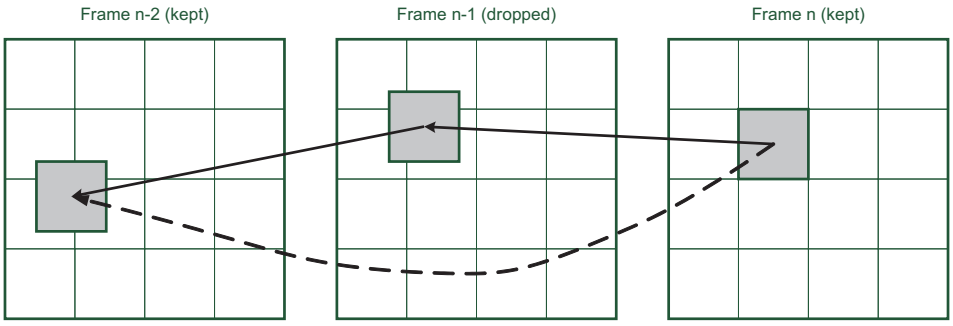


Figure 2.9.: Frame dropping and motion vector re-estimation.

One technique for re-estimation (or approximation) is the motion vector composition [112, 149] (sometimes called motion vector propagation - the term motion vector composition is used through this thesis). It tries to find where, in the dropped frame, a given motion vector points to, and then, in the next step, it attempts to find where this area points to in its reference frame, which was not dropped. This is shown in Fig 2.10(a). A problem arises when the reference block encompasses more than one macroblock. Some techniques that may be used to decide for the final motion vector are a simple majority or bilinear interpolation. In the first, the motion vector in the largest overlapping area is used, while in the second the motion vectors are weighted using the overlapping area for each of them. This process can be repeated if multiple frames are dropped.

Another technique for re-estimation is to scale the motion vector [142], usually assuming the motion as linear. In this technique, the motion vector is simply scaled by a factor proportional to the number of dropped frames. For the example in Fig. 2.10(b), the motion vector used is twice the original motion vector.

When the motion vector changes in the motion vector re-estimation, the residual also has to change or drift will occur (since changing the motion vector will change the

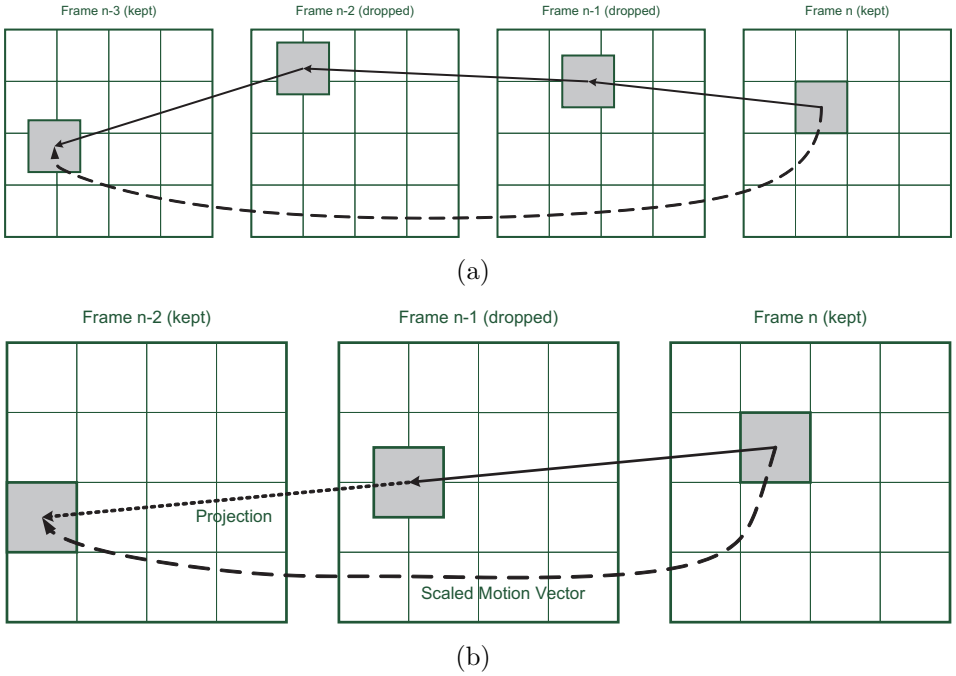


Figure 2.10.: Motion vector re-estimation techniques: (a) Motion Vector Composition; and (b) Motion Vector Scaling.

prediction). This may be done in the pixel domain in a straightforward way, but some DCT-domain motion compensation techniques may also be used [45, 26].

Several transcoders targeting temporal resolution reduction have been proposed for the H.264/AVC codec [71, 80, 117], particularly targeting applications such as fast forward video playback. Since some of these algorithms are important for the work on this thesis, they are reviewed in more detail on Chapter 4.

2.2.2. Heterogeneous Transcoding

This kind of transcoder performs the conversion of bitstreams across formats, for instance, from MPEG-2 to H.264/AVC. It may also provide the functionalities of homogeneous transcoding, like bit rate reduction and spatio-temporal resolution reduction, and some techniques developed for homogeneous transcoding may also be used [9, 135, 141].

The biggest difference from the architecture of an homogeneous transcoder to an heterogeneous transcoder is the presence of a syntax conversion module in the latter. Also, since the two codecs may use different tools (or may use the same tools with

different settings), the encoder and decoder motion compensation loops in heterogeneous transcoder are more complex than in homogeneous transcoders [40, 112].

In many works, heterogeneous transcoding is achieved using a complete decoder from the source stream and a reduced encoder for the transcoded stream, that reuses information present in the source bitstream to speed up the transcoding, in a similar fashion as the cascaded pixel domain transcoder. This is shown in Fig. 2.11.

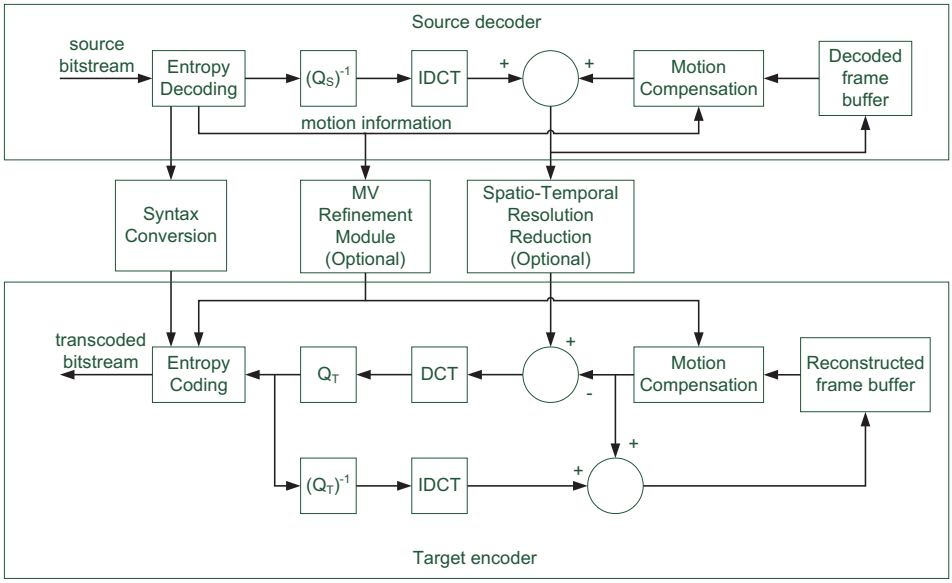


Figure 2.11.: Generic architecture for an heterogeneous video transcoder.

In Fig. 2.11, a decoder for the source format decodes the sequence and extracts some information of this bitstream, such as the motion vectors and mode information for the macroblocks. Then, these motion vectors and modes are post-processed according to the format accepted by the target format. To achieve better quality, an optional motion vector refinement may be performed. The transcoder encodes the sequence using these modes and motion vectors, reducing the complexity when compared to the trivial transcoder (which tests all modes and motion vectors for the target format). The details of the post-processing of the motion vectors and modes depend on the tools available for the target format [9, 135].

Recent Work on Heterogeneous Transcoding

This section discusses some of the work that has been done on heterogeneous transcoding, specifically on cascaded pixel domain transcoders, which is the architecture chosen to

implement the two transcoders developed in this thesis, presented in Chapters 5 and 6. This architecture has been used in transcoding H.263 bitstreams targeting other codecs, such as H.264/AVC [18, 46] or VP-6 (a proprietary codec used for internet video, in particular Adobe Flash Player) [51]. It has also been extensively used in MPEG-2 to H.264/AVC transcoders [42, 130, 137, 140], H.264/AVC to MPEG-2 transcoders [114, 118] , MPEG-4 to H.264/AVC [72], H.264/AVC to MPEG-4 [55] and even from other non-DPCM/DCT codecs to H.263 [101, 102] or H.264/AVC [85].

A simple way of classifying the contributions of these works is to separate them in algorithms for mode mapping, algorithms for motion vector approximation and algorithms for motion vector refinement. The goal of the mode mapping algorithms is to use information on the incoming bitstream in order to avoid testing all macroblock modes for the target codec, while the goal of the motion vector approximation algorithms is to maximize the reuse of the motion vectors in the incoming bitstream in order to avoid costly motion estimation operations in the target encoder. Finally, the goal of the motion vector refinement algorithms is to improve the reused and approximated motion vectors so that a good prediction can be achieved. Some of these algorithms are discussed next.

Algorithms for Mode Mapping

In order to reuse the mode used to encode a particular macroblock in the incoming bitstream, a range of algorithms have been proposed. A simple mode mapping algorithm was proposed in the context of a H.264/AVC to MPEG-2 transcoder [70]. In this algorithm, the H.264/AVC macroblock types are classified in three categories, skipped, inter and intra, for macroblocks encoded in SKIP mode, inter or intra modes, respectively. Then, in the transcoder, only the modes associated with these classes are tested. Another simple algorithm, used in the context of VC-1 to H.264/AVC transcoding, was proposed [100]. In this work, since the VC-1 codec offers a smaller number of modes than the H.264/AVC (for instance, only blocks sizes of 16×16 and 8×8 are used for motion compensation, and there is no skip mode for a macroblock), the transcoder uses both the macroblock type and the size of the transform used in VC-1 (the codec allows for four transform sizes, 4×4, 4×8, 8×4 and 8×8), proposing some rules based on heuristics, summarized in the form of a look-up table, to decide which modes are tested in the H.264/AVC.

The block mode statistics are used in a MPEG-4 to H.264/AVC transcoder [72]. In this work, several test sequences are transcoded using a trivial transcoder in order

to gather the macroblock mode conversion statistics. This information is then used to generate a look-up table, which is used during transcoding to decide which H.264/AVC modes are tested according to the MPEG-4 mode. A similar approach was used in a H.264/AVC to MPEG-4 transcoder [55].

In other works, the idea of using the block mode statistics is expanded, using machine learning algorithms to map the modes in the incoming bitstream and decide how the modes in the target codec are tested, in the context of MPEG-2 to H.264/AVC transcoding [41, 42, 52]. All these works are built around similar ideas: first, a few frames of test sequences are transcoded using a trivial transcoder. For these frames, some features are computed and stored for each macroblock, along with the optimal mode used to encode said macroblocks. Then, a machine learning algorithm is used to generate an algorithm to map features computed using the incoming bitstreams into modes to be tested in the target codec. The training is performed offline, with the goal to develop a single, generalized, mapping that can be used for transcoding any MPEG-2 video. In the first of these works [41], the features used include the MPEG-2 macroblock coding mode, the coded block pattern, and the means and variances for each 4×4 residual block, for a total of 37 features. In the other works [42, 52], the list of features was expanded to include the MPEG-2 DCT coefficients, neighbouring macroblock information, coded block pattern, the MPEG-2 motion vectors, the mean and variance of the 4×4 residual blocks, and the variance of the means and mean of variances for each group of means and variances, for a total of 131 features. A similar approach was presented by some of the same authors in the context of a Wyner-Ziv to H.264/AVC transcoder [85]. In this work, three features are used to generate the mapping algorithm, being the SAD of the residual computed in the Wyner-Ziv decoding process, the length of the motion vector generated by the Wyner-Ziv decoding process, and information from the Wyner-Ziv reconstruction process, and the same offline training process is used.

Algorithms for MV Approximation

Motion vector approximation is used when the reference frame for the motion vector in the incoming bitstream does not match the reference frame used for the target codec. This could happen for one of two reasons: either because the used reference frame is dropped, in temporal resolution reduction transcoding (in particular, targeting fast-forward video playback); or because the coding structure in the target codec is different from the coding structure in the incoming bitstream. The main technique to deal with

this is the Motion Vector Composition, and there are two main algorithms for this: Forward Dominant Vector Selection (FDVS) [147, 149] and Telescopic Vector Composition (TVC) [112]. These algorithms are reviewed in more detail on Chapter 4.

Algorithms for MV Refinement

In most cases, the transcoder attempts to improve the approximated or reused motion vector, so that a higher rate-distortion performance can be achieved. Several refinement methods have been proposed. Motion vector refinement using the full search algorithm with a very small search window have been proposed, both with a search window of 1 pixel [55] and 2 pixels [100, 117]. A variation of this approach is the Horizontal and Vertical Search algorithm (HAVS) [149], in which, instead of searching all points within the search window, it searches first for the minimum point in the horizontal line, then for the minimum point on the vertical line.

In some algorithms, a dynamic search window is used, so that the size of the search window used for motion estimation is derived from some information found in the incoming bitstream. A dynamic search window that is set according to the length of the incoming motion vector and the mode of the incoming macroblock was proposed [42]. In other work [70], the search range is not square, being a rectangle defined by the length of each motion vector component. Also, algorithms based on fast motion estimation methods have been proposed, such as a refinement based on Diamond search and a search window of 3 pixels [10], and an algorithm based on the Enhanced Prediction Zonal Search (EPZS) [81], used in the context of a MPEG-2 to H.264/AVC transcoder, in which the MPEG-2 motion vectors are considered to reduce the number of predictors used in the EPZS algorithm, and also to simplify the Diamond search used in the EPZS search. Both the Diamond search and the EPZS are explained on Appendix A.

2.3. Conclusions

This chapter presents basic concepts on video coding, in particular for a hybrid DPCM/DCT codec. The main modules - prediction model, image model and entropy coding - are briefly reviewed. It then discusses video coding formats, including some recent formats that are widely used nowadays. It finishes with a survey on the main

functionalities of video transcoding, and a review on techniques used on heterogeneous video transcoders.

The next chapter gives an overview of the three codecs used in this thesis. It is important to understand the tools used in (and available for) each of these codecs in order to understand the issues they originate in the transcoders.

Chapter 3.

The Codecs Used

This thesis presents transcoding solutions to two different transcoders, both using the H.264/AVC standard as the source bitstream, which is the current international standard for video coding [62], being used by many applications.

One of the transcoders targets a Wavelet-based scalable codec, W-SVC [124]. This codec follows a very different architecture from the traditional hybrid DPCM/DCT, using spatial and temporal Wavelet transforms, along with embedded coding, to achieve full scalability.

The other transcoder targets the new emerging standard, HEVC [58], which is being developed as the next international standard. The HEVC follows the traditional DPCM/DCT architecture, sharing many similarities with the H.264/AVC. However, there are substantial differences between the two codecs, as it is seen on the next sections.

While a complete review of each of these codecs would be out of the scope of this thesis, this chapter provides an overview of each of these three codecs, giving special attention to the most important modules for each transcoder.

3.1. The H.264/AVC Standard

The H.264/AVC codec is the latest ITU-T Recommendation and ISO/IEC International Standard for video coding [62]. It provides higher coding efficiency than the previous standards [99, 139], such as H.263 [61] and MPEG-2 [59], and it was developed targeting applications such as broadcasting, video on demand, high quality optical medias, among

others. Since its release, it has been adopted by many applications, such as Blu-ray discs, internet video streaming, digital television broadcasting, and others. Similar to previous ITU-T and ISO standards, such as H.263 and MPEG-2, it also follows a hybrid DPCM/DCT architecture [106]. However, compared to these standards, it has enhancements on all aspects of video coding, as prediction of pictures, coding efficiency and robustness and error resiliency [125, 139].

As a standard, the H.264/AVC is divided in profiles and levels [62, 106]. Each profile defines a set of requirements and coding tools that can be used within this profile. The most important profiles are the Baseline, Main, Extended and High (this was added later, on the Fidelity Range Extension, FREXt, amendment [63]). The levels define performance limits within a profile, placing limits on parameters such as sample processing size, picture size, coded bitrate and memory requirements [106].

This section explains the fundamental tools used in H.264/AVC Main and High Profiles, which targets high coding efficiency and high video quality. It also contains a comprehensive explanation on the motion estimation and macroblock partitioning modules, since these are going to play a fundamental role on the transcoders presented in the next chapters.

3.1.1. Encoding Sequence

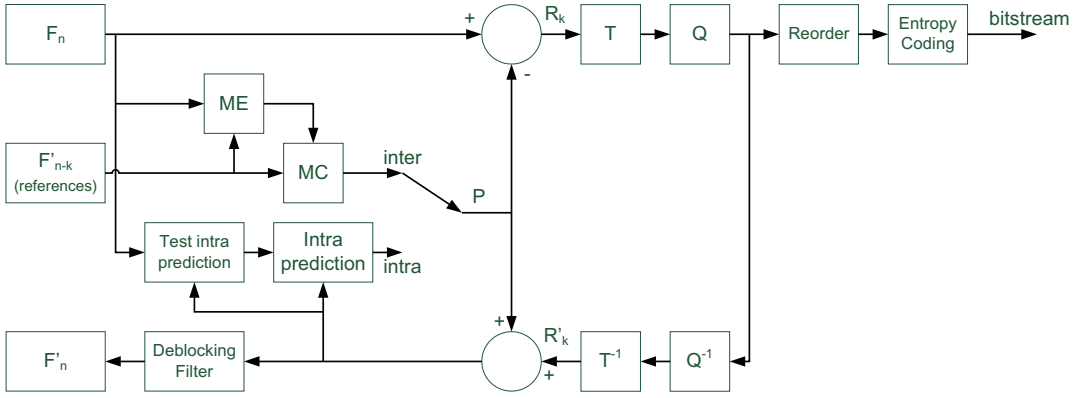
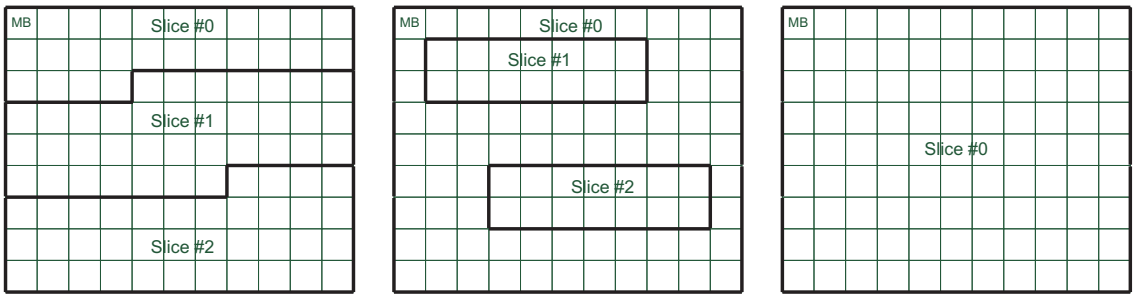


Figure 3.1.: H.264/AVC encoder diagram.

The basic flow of encoding is shown in Fig. 3.1. A more complete description of the steps performed when encoding a macroblock with the H.264/AVC codec is given in Appendix B, with an emphasis on the lossy steps of encoding. The H.264/AVC follows the hybrid DPCM/DCT model seen on Chapter 2, with well-defined prediction and

image models, as well as entropy coding. As most standards, it only defines the decoder and the bitstream syntax - the encoder implementation is open, as long as it produces a compliant bitstream. However, most encoders are likely to mirror the steps of the standard decoder [106]. A common encoder implementation is given in the following paragraphs.

The macroblock is the basic building block of the codec, with a fixed size of 16×16 luminance pixels (the actual number of chrominance pixels depend on the sampling format). The macroblocks are grouped into slices, which are defined as a sequence of macroblocks. A frame is defined as a collection of slices. Slices are self-contained, in a way that there is minimal inter-dependency between coded slices [106] (except for the inter prediction). Each slice can be correctly decoded without use of data from other slices (such as the state of the entropy encoders, which are content adaptive, or motion vectors for motion vector prediction, for example). Usually, the macroblocks are processed in raster scan order to form different slices. The standard also offers a more flexible way to group macroblocks in a slice, called Flexible Macroblock Ordering (FMO) [106, 138]. However, since FMO is not defined in the Main and High profiles, it will no longer be considered in this work. Also, it will be assumed just one slice per frame, which is a usual coding configuration if maximum rate-distortion performance is sought. Fig. 3.2 shows examples of possible slice groups when FMO is not used (Fig.3.2(a)), when FMO is used (Fig 3.2(b)) and the usual configuration of one slice per frame (Fig 3.2(c)), which is the configuration used in this thesis.



(a) (b) (c)

Figure 3.2.: Example of subdivision of a frame in slices (a) when FMO is not used; (b) when FMO is used; and (c) one slice per frame. In the figures, each small square represents a macroblock (MB).

The codec allows for several coding modes for a macroblock (see Sec. 3.1.4). The macroblock type decides how the prediction is formed for this macroblock (for instance, if it is intra or inter prediction, and how the block is partitioned to generate the prediction).

Typically, the encoder considers all coding modes for a given macroblock (according to the current codec configuration), computing an associated cost to each coding mode. Then, a mode decision module decides which is the best mode to encode this macroblock, usually using a rate distortion criteria similar to the one seen in Sec. 2.1.1.

After the mode is chosen, the codec computes the residual for this macroblock (except if the chosen mode is SKIP or Intra PCM - these cases are discussed in Sec. 3.1.4). The choice of the macroblock type may restrict some of the other codec options - for instance, the 8×8 transform cannot be used if the mode smallest prediction unit is smaller than 8×8 pixels, and the SKIP mode forces the residual to be zero, resulting in a reconstructed block that is equal to the prediction. After the residual is computed, it is then transformed and quantized. The quantized coefficients are reordered and entropy encoded, generating the bitstream for this macroblock. In addition, the encoder performs the inverse quantization and transform operations and add the prediction to the reconstructed residual, resulting in the reconstructed macroblock $B_n'^k$, which is then added to the current reconstructed frame buffer. Once all macroblocks in a slice are encoded, a deblocking filter is applied to the frame to reduce the blocking artifacts [78], and the output of the filter is added to the frame buffer of reconstructed frames, which will be used as reference for inter-prediction in the next frame. An important encoder decision is the coding structure, which decides which frames are encoded as intra or inter, and the type of inter prediction that is allowed for the inter frames. This is discussed in Sec. 3.1.3.

3.1.2. Decoding Sequence

The basic flow of decoding is shown in Fig. 3.3. The bitstream for the macroblock is decoded, and the inverse operations of reordering, quantization and transform are applied, generating the residual $R_n'^k$. This residual is added to the prediction P_n^k (intra or inter), which is computed using motion compensation on the previously decoded frames or intra prediction using the previously decoded macroblocks. Once all macroblocks are decoded, the deblocking filter is applied and the frame is ready to be displayed.

3.1.3. Slice Types and Coding Configurations

Each slice can be encoded using the following slice types:

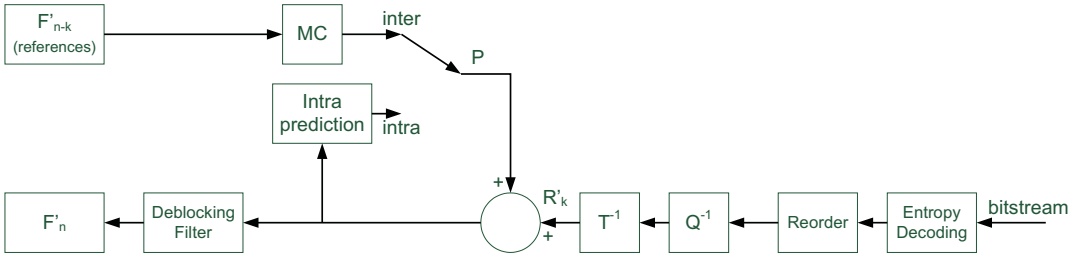


Figure 3.3.: H.264/AVC decoder diagram.

- I (intra): all macroblocks within this slice are encoded in intra mode (i.e., they are predicted only using information from previously coded macroblocks within the same slice).
- P (predictive): can contain intra and P macroblocks. P macroblocks are predicted using only one motion compensated prediction signal from a "LIST 0" reference frame.
- B (bi-predictive): can contain intra, P and B macroblocks. B macroblocks are predicted using up to two motion compensated prediction signals, one from a "LIST 0" reference frame and the other from "LIST 1".

In addition to these modes, the extended profile includes SP (switching P) and SI (switching I) modes [68, 106]. These modes are not defined in the Main and High profiles, and hence are not further discussed here. There is an additional type of coded frame called Instantaneous Decoder Refresh (IDR). This type of frame contains only I or SI slices, and it completely resets the lists of reference frames. All subsequent frames can be completely decoded without the use of any information prior to the last IDR frame.

In the H.264/AVC codec, the slice mode is encoded in the slice header, providing full flexibility on the macroblock mode choice for each slice. Previous standards, such as H.263 and MPEG-2, define a group of pictures (GOP) that specifies the order in which intra and inter frames (including P and B frames) are arranged. The sequence would then be encoded by dividing the frames in different GOPs, repeating the frame structure. In the H.264/AVC, however, there is no restriction to the GOP. Although it is usual for encoder's implementations to define such a GOP for easier encoding, the standard itself does not require it. In the place of a fixed GOP structure, the codec has two lists that stores reference frames for motion estimation and compensation, namely, "LIST 0" and "LIST 1". Also different from previous standards, both lists may contain past and future frames, in display order. P macroblocks may use one motion vector pointing to a

frame in "LIST 0", while B macroblocks may use one motion vector pointing to a frame in "LIST 0", one pointing to "LIST 1", or two motion vectors, each one pointing to a frame in each list, forming a bi-directional prediction.

In order for these coding structures to work, the codec provides tools to manage how the encoded frames are added to these lists. Even though each list may contain up to 16 reference frames, normal coding conditions use a smaller number (ranging from 1 to 5). There are two types of reference frames in the list, short term references and long term references [43, 106]. By default, once the encoding of the current frame is finished, it is reconstructed and added to the appropriate list as a short term reference. This frame is added to the first position on the list, and the frame that was in the last position is discarded and it is no longer used as a reference frame. Long term references, on the other hand, have to be explicitly added and removed. These operations are signaled in the bitstream, so that the decoder can have access to the same prediction lists as the encoder, avoiding drifting problems. An example can be seen in Fig 3.4, where the "LIST 0" is used with a maximum of 2 reference frames.

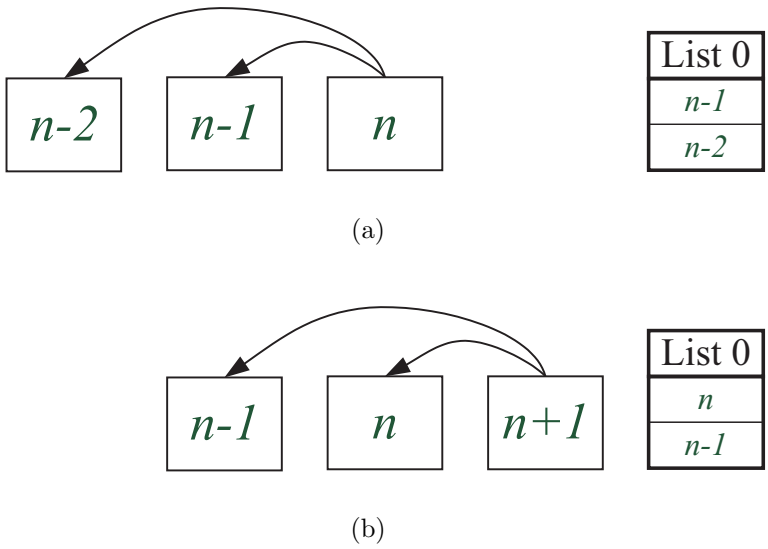


Figure 3.4.: Example of a reference frame list for H.264/AVC. In this example, the "LIST 0" is used with a maximum of 2 reference frames, when: (a) encoding frame n; and (b) encoding frame $n+1$.

While the standard itself is fully flexible on the choice of slice type, some coding configurations are more often used in the literature. Through this thesis, five coding configurations are used:

- *IPP* with 1 reference frame, referred as *IPP1*. This is a common low-delay configuration, where only the first frame is encoded as intra, and all the others are encoded as P-frames with one reference frame (the previous frame). An example can be seen in Fig. 3.5(a).
- *IPP* with 4 reference frames, referred as *IPP4*, where only the first frame is encoded as intra, and all the others are encoded as P-frames with four reference frames (the previous four frames).
- *IPP* with 5 reference frames, referred as *IPP5*.
- *IBBP* with 2 reference frames for the B-frames (1 in each direction) and 5 for the P-frames, referred as *IBBP*. This is a common high performance configuration, where the first frame is encoded as intra, and afterwards every third frame is encoded as a P-frame, with 5 reference frames, and the two frames in between are encoded as B-frames, with 2 reference frames (the previous and the future I or P frame). In this configuration, B-frames are not used as reference. This configuration needs to encode the frames in a different order from the display order. An example can be seen in Fig. 3.5(b).
- *Hierarchical* with 1 reference frame for each direction and n levels, referred as *Hierarchical n-levels*. An example of hierarchical configuration with 3 levels is given in Fig. 3.5(c). In this configuration, the display order is also different from the coding order.

3.1.4. Macroblock Types and Partitioning

For the sake of simplicity, the macroblock types in the H.264/AVC standard are classified in four types here: (i) SKIP; (ii) Intra PCM; (iii) intra macroblocks, and (iv) inter macroblocks. The last two types (intra and inter macroblocks) are encoded using the DPCM/DCT model seen on Chapter 2, where a prediction is formed, followed by transform, quantization and entropy coding. The other two modes are encoded differently.

IPCM

In this mode, the pixel values are transmitted directly, without the use of the process of transform and quantization. It allows the encoder to represent the values of an

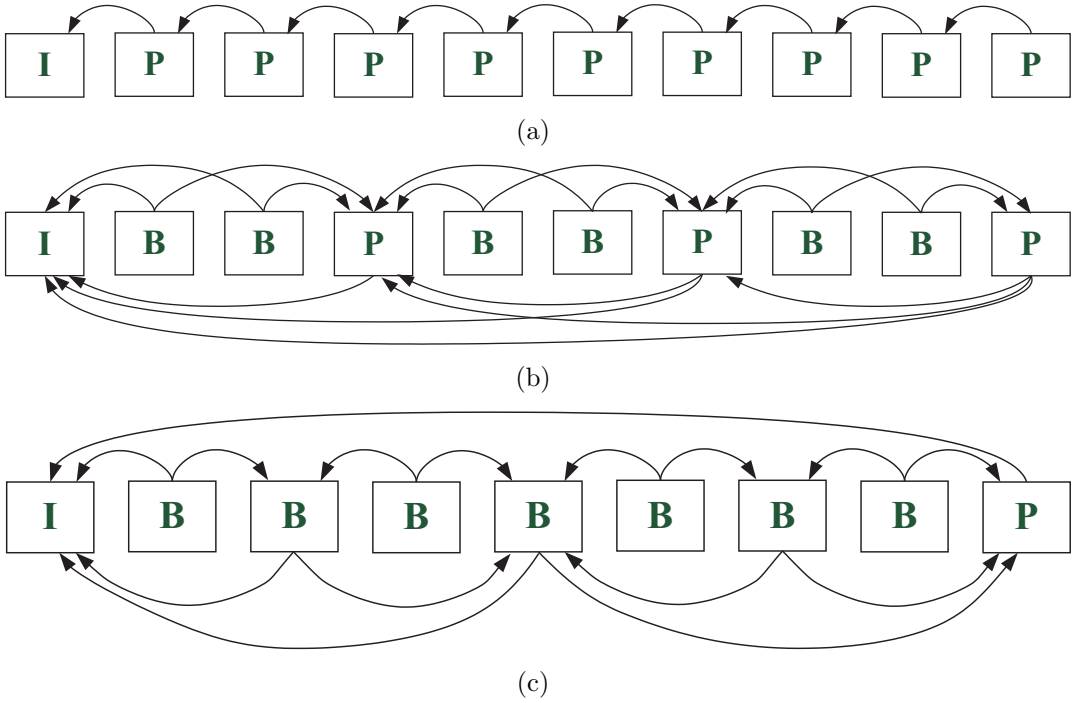


Figure 3.5.: Example of the used coding configurations: (a) *IPP1*; (b) *IBBP1-5*; and (c) *Hierarchical* 3-*levels*. The arrows in the figures point from the source frame to the reference frames that can be used for inter prediction.

anomalous macroblock without significant change on the bit rate. This mode also allows the encoder to place a hard limit on the number of bits that are needed to encode a macroblock without any constraints on decoded image quality [139].

SKIP Modes

The H.264/AVC standard allows two skip (or direct) modes, one for P slices and other for B slices (B slices may use both modes when deciding the mode for a macroblock).

In the skip mode for P slices, called $PSKIP$, no additional information is transmitted and the prediction is used as the reconstructed macroblock. Even the motion vector is not transmitted, instead, it is predicted using the motion vectors from the neighbouring areas [106]. The reference frame used is the first frame in the "LIST 0" (usually, the last encoded frame). This mode allows large unchanged areas of the frame to be encoded very efficiently [139].

For the B slices, the skip mode can be chosen independently by each 8×8 sub-macroblock, and it is also called Direct Mode. Similarly to the $PSKIP$ mode, the

motion vector is predicted in a pre-defined way, and no other signal is transmitted. However, it uses reference frames from both "LIST 0" and "LIST 1", forming a bi-directional prediction. The biggest difference from the $PSKIP$ mode, apart from the fact that it uses two motion vectors, is that there are two ways, spatial and temporal, to form the predicted motion vector, and this is signaled in the slice header (hence, they are always the same within the slice). Throughout this thesis, only the spatial mode is used.

Intra Macroblocks

The intra mode is supported by all slice types. In H.264/AVC, in addition to the Intra PCM mode, two other types of intra modes are possible: intra 16×16 and intra 4×4. The FREext amendment has added the intra 8×8 mode (which is only available in the High profile) [125]. The codec allows for four types of intra prediction for the intra 16×16 mode, and nine types for the intra 8×8 or intra 4×4.

In the three intra macroblock types, a prediction is formed using the already encoded neighbouring pixels of the current block. After the prediction is formed, the residual is computed, followed by transform, quantization and entropy coding.

Inter Macroblocks

For inter macroblocks, the partitioning define the blocks that are used for motion compensation. Inter macroblocks can be partitioned in 4 different sizes: 16×16, 16×8, 8×16 and 8×8. If the 8×8 mode is chosen, each 8×8 sub-macroblock can be further partitioned in 8×8, 8×4, 4×8 and 4×4. The possible partitions of a macroblock are shown in Fig. 3.6.

Each partition may choose motion vectors independently, however, for each 8×8 sub-macroblock partition, the same reference frames is used, regardless of the sub-partitioning.

The motion vectors in H.264/AVC can use up to quarter-pixel accuracy. To generate the interpolated pixels at half-pixel accuracy, a simple six-tap Finite Impulse Response (FIR) filter is used, which is applied independently in both horizontal and vertical directions [139]. The quarter-pixel samples are linearly interpolated using the two adjacent samples, and it is also applied independently in both directions.

For B macroblocks that have two motion vectors the prediction is computed as a simple average of the reference blocks described by each motion vector. After the prediction is formed, it is used as the prediction for this macroblock the same way the prediction for P macroblocks (or the prediction for B macroblocks with a single motion vector). The standard also supports Weighted Prediction [23, 106]. In this mode, each prediction is scaled (or weighted) by a scaling factor ω and added to generate the final prediction. These weights can be implicit, based on the relative temporal positions of the reference frames, or explicit, defined in the slice header. However, throughout this thesis, explicit weighted prediction is not used, and the weights used for bi-directional prediction are always $\omega = \frac{1}{2}$.

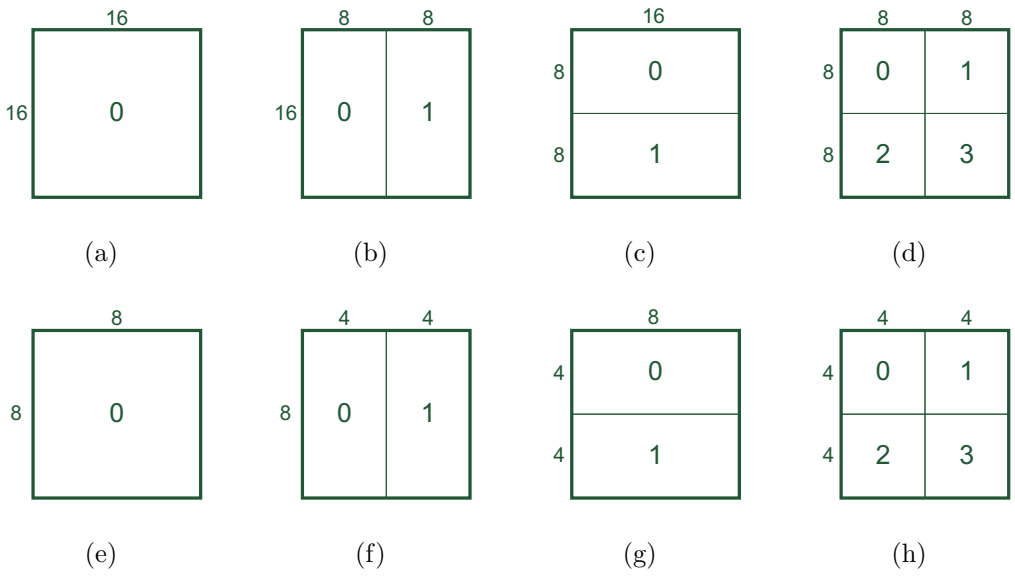


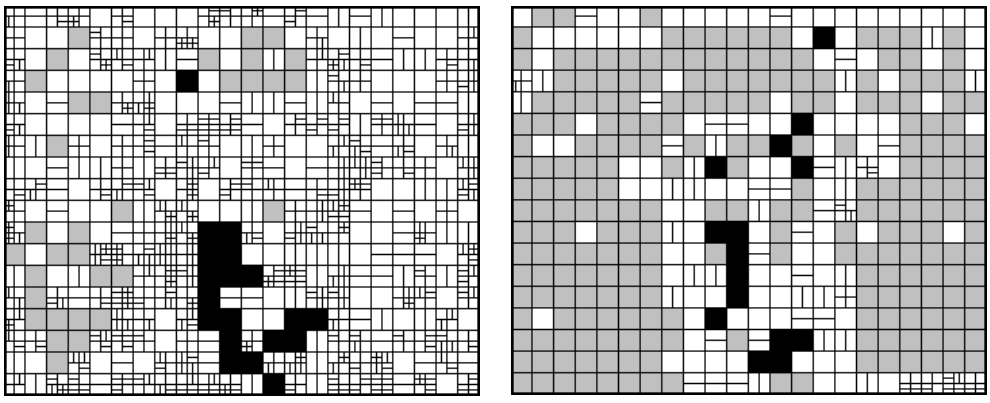
Figure 3.6.: Possible partitions for a macroblock: (a) 16×16; (b) 16×8; (c) 8×16; and (d) 8×8; and for a 8×8 sub-macroblock: (e) 8×8; (f) 8×4; (g) 4×8; and (h) 4×4.

As seen in Sec. 2.1.1, a finer partitioning provides a better prediction, but also requires the transmission of more side information (the partitioning, motion vectors and reference frames for each partition). In order to decide the partitioning, an optimization strategy is applied. However, as others ITU-T and ISO/IEC standards, only the reference decoder is defined in the H.264/AVC standard itself [62]. Thus, this optimization strategy is dependent on the encoder that is used.

Typically, H.264/AVC encoders use a rate distortion optimization model similar to the one discussed in Sec. 2.1.1 to find the optimal motion vector for a given partition [82, 126]. Once all partitions for a given mode have been tested, a mode decision module

selects the mode that will be used to encode the macroblock. The mode decision module may use the actual number of bits required by that mode using the current entropy encoder and the distortion of the reconstructed macroblock as the rate R and distortion D, however, in all cases, a cost function of the form $J = D + \lambda R$ is used.

In the H.264/AVC, the Lagrange multiplier λ is often used as a function of the quantization parameter (thus, $\lambda(QP)$) [106]. This means that the encoder attempts to optimize the rate-distortion for a fixed bitrate and spatio-temporal resolution. For instance, low QP values (associated with high quality, and thus high bit rate) have a lower λ value, indicating a optimization biased towards the reduction of the distortion, and high QP values (associated with low bit rate, and thus low quality) have larger λ values, indicating the importance of the rate. This leads to changes on the mode decision (and, thus, the motion vectors), as smaller partition modes are more often chosen in high quality bitstreams, if compared to the occurrence of these modes in low quality bitstreams. An example of this behaviour can be seen in Fig. 3.7. In this example, the first 5 frames of the *Foreman CIF* sequence were encoded using the H.264/AVC codec with *IPP5* configuration, and two different quantization parameters (QPs). It can be seen from the figures that using the first QP (18, shown in Fig. 3.7(a)), the frame partitioning resulted in much smaller partitions, while for the second QP (36, shown in Fig. 3.7(b)) the frame partitioning resulted in much larger partitions. This behaviour is relevant for the H.264/AVC to W-SVC transcoder (Chapter 5), since this optimization is different for scalable codecs.



(a) (b)

Figure 3.7.: Example of macroblock partitioning depending on the QP for the third frame of the *Foreman CIF* sequence using : (a) $QP = 18$; and (b) $QP = 36$. In the figure, gray blocks represent the use of the SKIP mode, black represent intra blocks and white represent inter blocks.

3.1.5. Residual Coding and Loop Filtering

In the H.264/AVC standard, the residual can be both from intra and inter macroblocks. The coding path for the residual is similar to what is found in previous standards, with transform, quantization, reordering and entropy coding [48, 104] and seen on Chapter 2. To encode the residual, several transforms can be used, dependent on the modes and whether the residual is for luminance or chrominance components. However, the two most important transforms are a modified 4×4 integer Discrete Cosine Transform (DCT) [83], and a modified 8×8 integer DCT, that is used only in the High profile [125].

The quantizer step is determined by a quantization parameter, which can be set in a macroblock basis. H.264/AVC uses a scalar quantizer and the post-scaling factor of the transform coefficients are applied in the quantization process [106]. Finally, the standard offers two types of entropy coding to encode residual coefficients: a Context-Adaptive Variable Length Coding (CAVLC) [106] and a Context-Adaptive Binary Arithmetic Coding (CABAC) [84, 106]. The choice of the entropy coder is not relevant for the transcoder, so throughout this thesis CABAC is used, as it yields better compression.

After all macroblocks are encoded, the codec uses a deblocking filter in order to reduce the blocking artifacts [78], and the output of the filter is added to the frame buffer of reconstructed frames, which will be used as reference for inter-prediction in the next frame. This filter is applied to the reconstructed block, but it is not applied equally to all pixels in the frame. Instead, it is applied only to the boundary between partitions. It uses information on how the partitions are encoded (for instance, whether it was encoded in intra or inter modes, if it contains coded coefficients, or if it uses different motion vectors) to select the appropriate filter strength. The idea is that the filter should be stronger at borders where it is likely that the blocking distortion is higher, and where this blocking distortion was likely introduced by the encoding process, and not found in the original sequence. The information used to decide how the deblocking filter is applied is available at the decoder, so no other information needs to be transmitted.

3.2. The W-SVC Codec

The W-SVC codec [124] is a scalable codec that uses a discrete wavelet transform and a technique called motion compensated temporal filtering (MCTF) [31] to achieve full

scalability. The architecture of the W-SVC is very different from that of H.264/AVC and other DPCM/DCT codecs. Even elements that are present in both codecs, such as the motion estimation and entropy coding, are implemented differently, because of the peculiarities of each codec.

This section explains the fundamentals of Scalable Video Coding (SVC) and the basic structure of the W-SVC codec. It also contains a comprehensive explanation on the motion estimation module of the codec, since this is going to play a fundamental role on the transcoder.

3.2.1. Scalable Video Coding

Typically, scalability is divided in three main types [90]: spatial (resolution), SNR (signal to noise ratio, i.e., quality) and temporal (frame rate). A fully scalable video codec should provide all three main types of scalability, and also provide a competitive rate distortion performance compared to conventional codecs (i.e., non-scalable codecs). Fig. 3.8 shows an example of temporal scalability, Fig. 3.9 shows an example of spatial scalability and Fig. 3.10 shows an example of quality scalability.

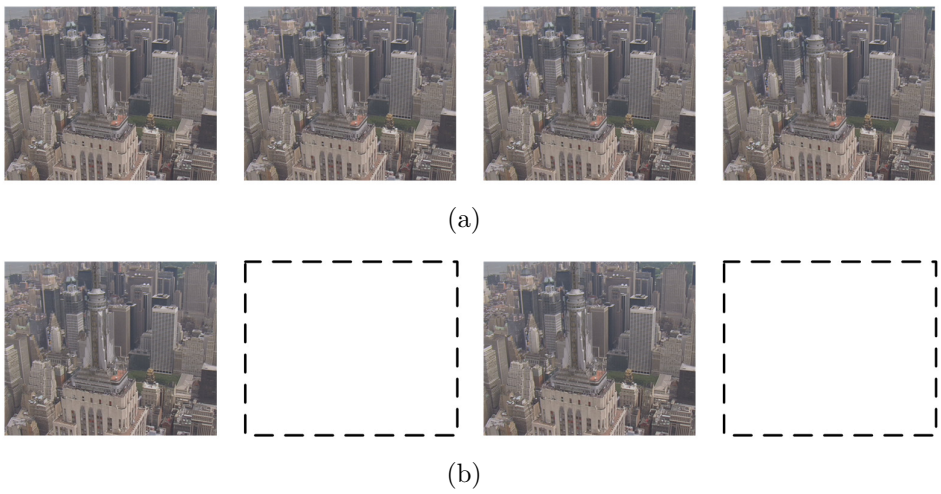
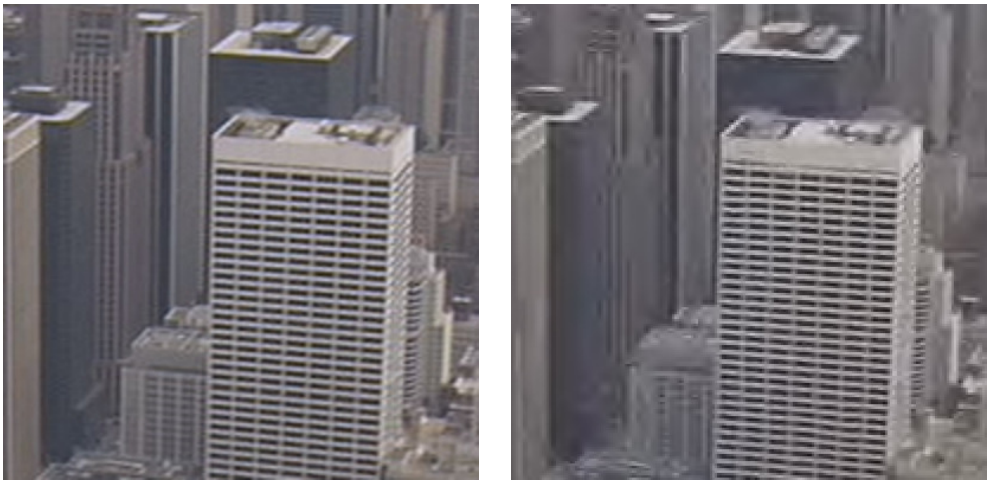


Figure 3.8.: Example of temporal scalability: (a) Original sequence; (b) Sequence in a lower frame rate. The marked frames are not available in the lower frame rate sequence.

In a scalable video codec, the video sequence is encoded into an embedded stream [90]. To address each individual application, the video transmitter uses a rate allocation mechanism, called extractor, which extracts the necessary data from this embedded

(a) (b)

Figure 3.9.: Example of spatial scalability: (a) Sample frame at original resolution; (b) Sample frame at a lower resolution.



(a) (b)

Figure 3.10.: Example of quality scalability: (a) Sample frame at original quality; (b) Sample frame at a lower quality.

stream. This extractor works at the bit-level, only selecting which bitstreams will be sent using appropriate flags in the stream. Therefore, the extractor is a very low complexity tool, and the stream it generates has the quality, resolution and frame rate required. Also, since extraction is a very low-complexity operation, it can be performed in any intermediate network node, even at the receiver side, if needed. This is shown in Fig. 3.11. It is important to notice that the scalability is embedded in the bitstream itself, therefore, the same bitstream can be extracted multiple times. However, information lost by a previous extraction cannot be restored in a subsequent extraction of the same bitstream.

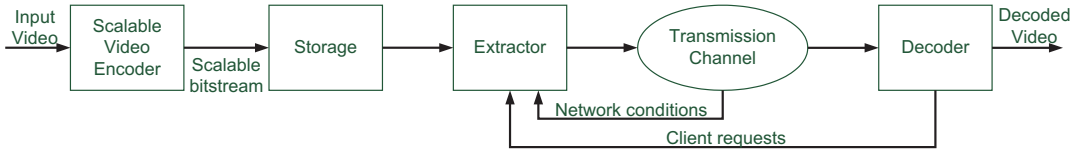


Figure 3.11.: SVC application scenario.

3.2.2. Encoding Sequence

A high-level block diagram of the W-SVC codec [124] is shown in Fig. 3.12. First, the motion estimation module finds the optimal motion vectors for the current frame being encoded. However, differently from DPCM/DCT codecs, traditional motion compensation is not performed. Instead, the input video is subjected to Motion Compensated Temporal Filtering (MCTF) [31], which aims to reduce the correlation between consecutive frames and provide a basis for temporal scalability. Temporally decorrelated frames are subjected to spatial decomposition, which reduces the correlation in the spatial direction and provides a basis for spatial scalability. This particular coding architecture is called the $t + 2D$ architecture [122] since spatial decomposition is preceded by temporal decomposition. Different decomposition orders, such as $2D + t$ and $2D + t + 2D$ architectures [122], can also be used in the W-SVC codec. Here, the focus will be on the $t + 2D$ architecture, since it is the simplest of all architectures and provides the best performance in case of temporal and quality scalability [89].

In the W-SVC codec, MCTF is implemented via lifting scheme [35]. The purpose of the motion estimation module is to estimate the motion between the frames so that the filtering can be performed in the direction pointed by the motion vectors. Many transforms can be chosen in the codec, defined in the encoding parameters, like the popular 9/7, 5/7, 5/3 and 1/3 wavelet transforms [96]. Texture coding is performed

by Embedded Zero Block Coding (EZBC) [53], which encodes wavelet coefficients in an embedded manner in order to remove the redundancies remaining after the wavelet decomposition and provide the basis for quality scalability. Finally, the resulting data is mapped into the scalable stream in the bitstream organization module, which creates a layered representation of the compressed data [150].

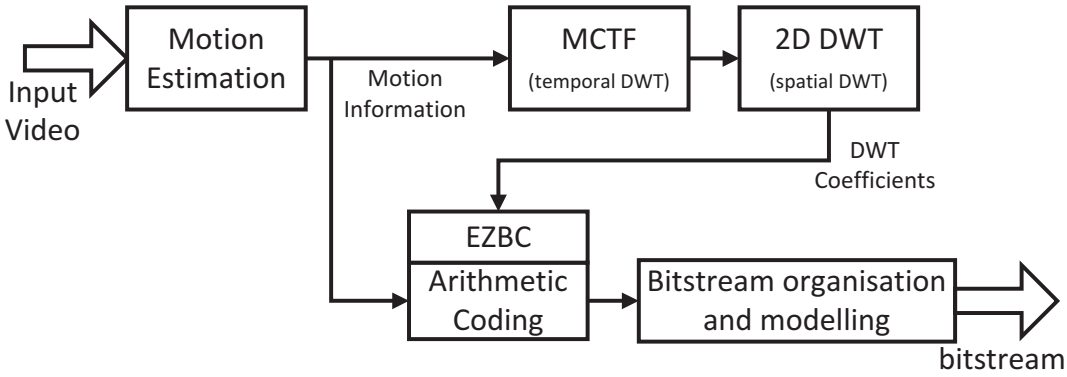


Figure 3.12.: Framework of the W-SVC codec using the $t + 2D$ architecture.

3.2.3. Decoding Sequence

As with the H.264/AVC and the HEVC codecs, the W-SVC decoder mirrors the encoder. However, the main difference is that the bitstream available for the decoder may be different from the bitstream produced by the encoder, if any extraction process was applied to the bitstream. At the decoder, first the bitstream is parsed and the arithmetic decoder decodes the coefficients. Then, an inverse DWT is applied in the spatial direction, followed by the inverse MCTF, which finally produces the reconstructed frames. Since the bitstream available to the decoder may be different from the bitstream produced by the encoder, some of the wavelet subbands may be lost in the extraction process, and they are not recovered here. Instead, the video sequence is decoded at a different spatio-temporal and quality resolution. The decoder architecture can be seen in Fig. 3.13.

3.2.4. Motion Compensated Temporal Filtering - MCTF

While there are others possible architectures that use wavelet transforms in video coding, MCTF [31, 97] is the one that offers better advantages. It offers high freedom scalability without drift problems, which are common in other architectures [116].

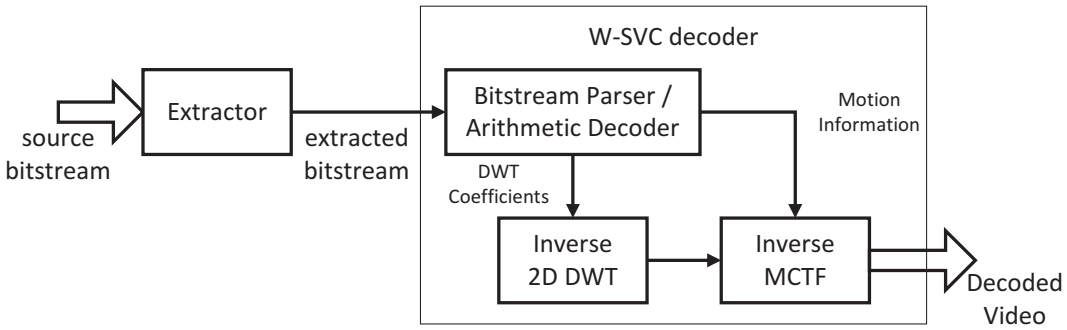


Figure 3.13.: Framework of the W-SVC decoder using the $t+2D$ architecture. The extractor is shown here, but it is not part of the decoder.

A simple way to apply the wavelet transform in the temporal dimension would be to apply the filter in the co-located pixel of each frame, so, for instance, apply the filter in the signal: $\{f_0(i,j), f_1(i,j), ..., f_N(i,j)\} \mid \forall i,j$. Notice that, if the filtering is performed via lifting scheme [35], the only samples of the signal needed to apply the filter for a particular sample $f_n(i,j)$ are its neighbours $f_{n-1}(i,j)$ and $f_{n+1}(i,j)$. In MCTF, instead of applying the filter using the co-located pixels of each frame, the filtering is performed on the direction of motion. So, consider that the motion for a particular pixel (i,j) is represented by $\left(d^i_{n\to n-1}, d^j_{n\to n-1}\right)$, between the frames n and $n-1$, and $\left(d^i_{n\to n+1}, d^j_{n\to n+1}\right)$, between the frames n and $n+1$. Then, when applying the filter to the sample $f_n(i,j)$, MCTF will use as neighbours the samples $f_{n-1}\left(i - d^i_{n\to n-1}, j - d^j_{n\to n-1}\right)$ and $f_{n+1}\left(i + d^i_{n\to n+1}, j + d^j_{n\to n+1}\right)$. In order to perform the inverse transform to recover the original signal, the decoder needs the wavelet coefficients and the motion descriptors $\left(d^i_{n\to n-1}, d^j_{n\to n-1}\right)$ and $\left(d^i_{n\to n+1}, d^j_{n\to n+1}\right)$, and the original signal can be perfectly reconstructed.

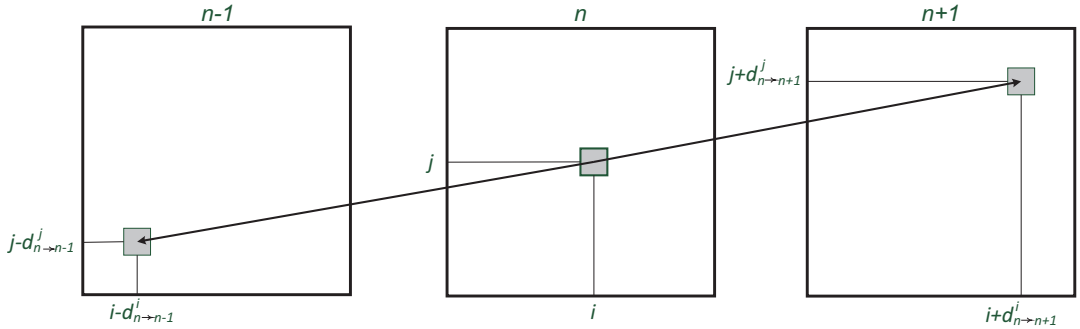


Figure 3.14.: Applying MCTF via lifting scheme.

In order to specify the descriptors, and not to generate too much side information, the motion is usually described for a block of pixels, instead of a single pixel (as in the

previous example), in a way similar to the motion estimation described in Sec. 2.1.1. However, in MCTF, the motion estimation is performed in the original frames, and not in the reconstructed frames, and the motion compensation is carried out during the filtering. Other techniques may be used together with MCTF, in order to improve the overall performance, such as sub-pixel interpolation and bi-directional wavelet filters [96] (i.e., filters that have both a prediction and an update step when implemented via lifting).

In MCTF, subsequent frames are transformed in low-pass frames (L) and high-pass frames (H). The process can be repeated on the low-pass frames, generating LL and LH frames, and so on. Fig. 3.15 shows the temporal decomposition for a group of 8 frames. Only the frames needed for perfect reconstruction and the motion information are kept and encoded.

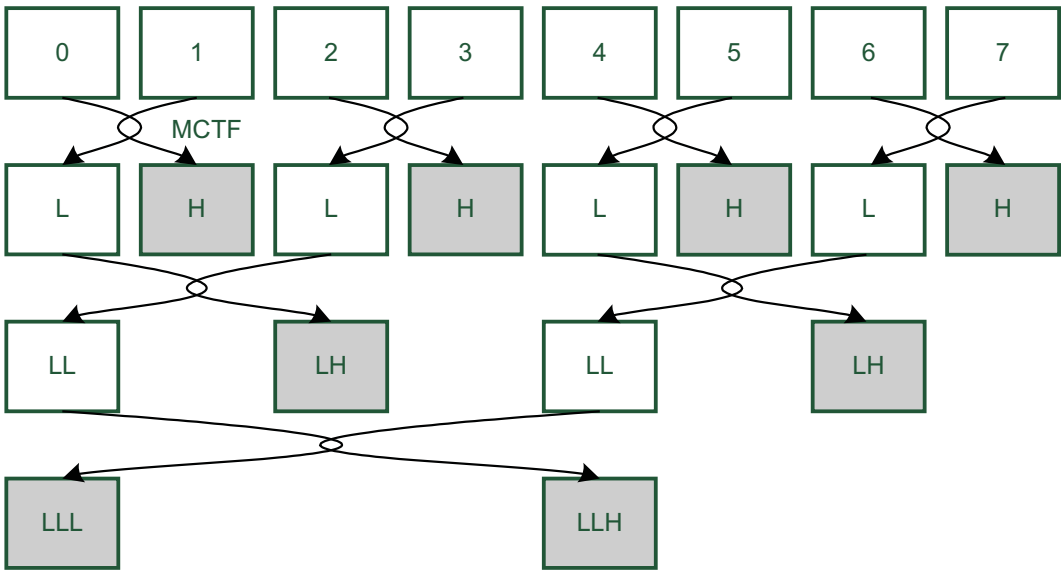


Figure 3.15.: MCTF for a group of 8 frames. Only the motion information and the highlighted frames are needed for perfect reconstruction. The low-pass frames are denoted as L and the high-pass frames are denoted as H.

The two most usual ways to use MCTF are the $t+2D$, also known as spatial-domain MCTF, and $2D+t$, also known as in-band MCTF [122]. In the first approach, $t+2D$, the temporal decomposition is applied first, and then the spatial decomposition is applied in the resulting coefficients. In $2D+t$, the spatial decomposition is performed first, and then the temporal decomposition is performed on the transformed frames. These approaches are shown in Fig. 3.16. The W-SVC codec can use both approaches, and also a third approach, $2D+t+2D$ [122].

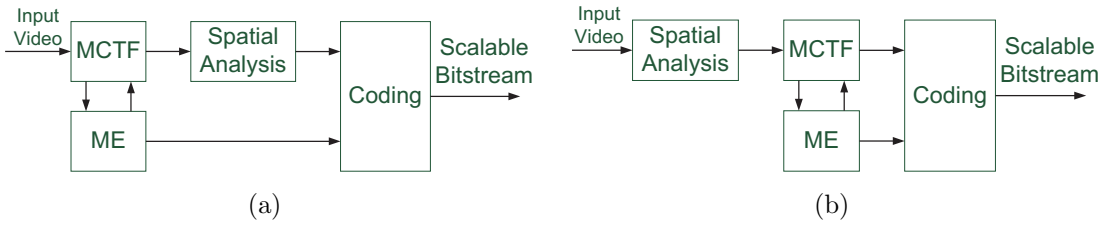


Figure 3.16.: Video encoder structures for MCTF (a) $t + 2D$; and (b) $2D + t$.

3.2.5. Motion Estimation

In order to describe the motion, the W-SVC codec uses a block based motion estimation module, describing the motion for each block in the frame using motion vectors. It uses a quadtree motion estimation that is similar, in some ways, to that found in the H.264/AVC standard [139]. However, there are some differences on the techniques and parameters themselves, and also on the optimization.

In the W-SVC codec, the macroblock size is itself an encoder parameter (which should be a power of two). Another encoder parameter is the maximum number of partitioning levels allowed. The codec partitions the macroblock in a recursive way: if the macroblock size is $n \times n$, and the parameters allow for k partitioning levels, then at the first partitioning level the macroblock would have 4 blocks of $\frac{n}{2} \times \frac{n}{2}$, and each of these blocks could be further partitioned (provided $k > 1$). So, the minimum block size would be $\frac{n}{2^k} \times \frac{n}{2^k}$.

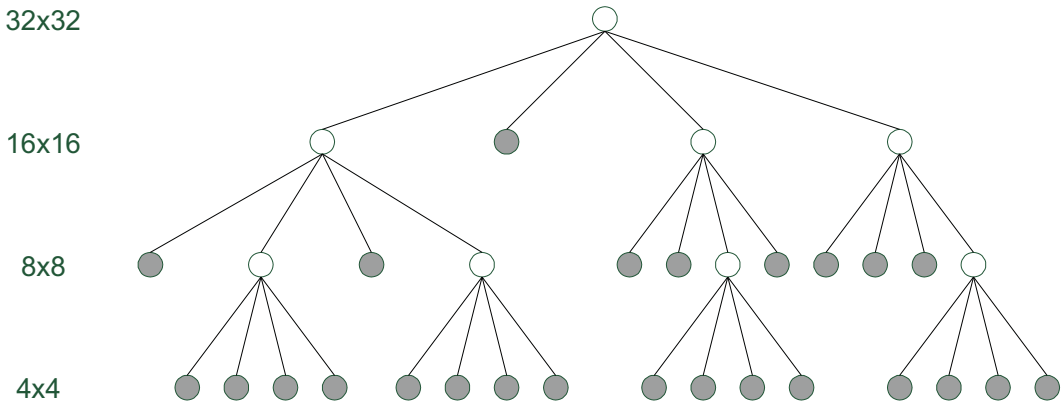


Figure 3.17.: Example of partitioning tree. In this example, the initial block size was 32×32, and the smallest block size allowed is 4×4.

On this example, the macroblock size was set to $n = 32$ and the number of partitioning levels was set to $k = 3$. Thus, block sizes of 32×32, 16×16, 8×8 and 4×4

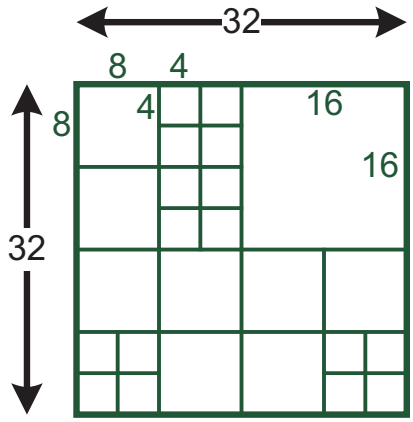


Figure 3.18.: Example of partitioning of a block. This is an alternative view of the situation shown in Fig. 3.17.

Table 3.1.: Available modes for each block regarding the selection of reference frames.

	Available motion modes			
Wavelet filters	Intra	Forward	Backward	Bi-Directional
$x/2$ (e.g.: Haar)	Yes	Yes	No	No
x/n, $n > 2$ (e.g.: 5/3)	Yes	Yes	Yes	Yes

are allowed. Unlike H.264/AVC, in which the modes are pre-defined, in the W-SVC the partitioning is more flexible, depending on these encoding parameters. Usually, the four blocks that appear after the block is split are called the children, while the block itself is called the parent. Also, a leaf node is each block that was not further split and thus, for each leaf node, the motion information is encoded and transmitted, such as motion vectors and the mode. In Fig 3.17, the leaf nodes are highlighted. The same situation is also shown in Fig. 3.18, using an alternative view.

3.2.6. Selection of Reference Frames

Depending on the wavelet filters chosen for temporal decomposition, each leaf node can choose between 4 modes: intra, forward, backward and bi-directional prediction. This can be seen in Table 3.1.

However, due to the temporal scalability of the codec, there are rigid limitations on the reference frames that the block matching procedure can use, and they are the same for an entire frame. Each block may use up to two reference frames, one past and

one future frame. This is illustrated in Fig. 3.19, for the case of using three temporal decomposition levels.

This is similar to the hierarchical structure described in Fig. 3.5 for the H.264/AVC codec. In the figure, each frame is labeled as L_n, which indicates that that frame belongs to the n-th layer. The arrows point from the frames being predicted to the reference frames in the process of MCTF. Note that this rigid structure straightforwardly enables temporal scalability - since all frames in the last layer (in the example, L_3) are high-pass frames in the first subband division of the first level of MCTF (seen in Fig. 3.15, also for 3 levels), they can be discarded and the structure could still be decoded at half the frame rate, since these frames are never used as reference for frames in lower layers.

The particular structure shown in Fig. 3.19 allows for 4 levels of temporal scalability. The sequence can be decoded at $f_R \in \left\{1\ ,\ \frac{1}{2}\ \frac{1}{4}\ \frac{1}{8}\right\}$, where f_R is the frame rate. If more levels of temporal scalability are needed, more levels of temporal decomposition should be used in the hierarchical structure.

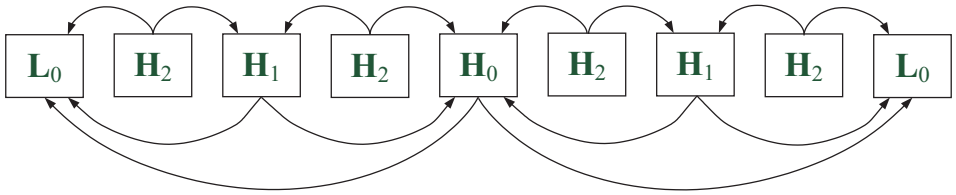


Figure 3.19.: Hierarchical structure for selecting the reference frame with 3 levels of temporal decomposition.

3.2.7. Sub-Pixel Motion Estimation

The precision of the motion vectors is also an encoder parameter. While the optimum precision is quarter-pixel, other precisions can be used, such as 1/2 and 1/8. Although 1/8 may be worse than 1/4 due to the overload of motion vector information that needs to be transmitted [89], it is useful in the codec when spatial scalability is applied.

The codec uses a sinc interpolation to get the pixel value (or coefficient value) [29]. This is applied separately in both vertical and horizontal directions. The contribution of the nearest sample is proportional to $\frac{sin(t)}{t}$, where t is the distance between the interpolated sample and the original sample. To avoid visual artifacts, a Hamming window is used to define the filter.

3.2.8. Optimisation

The goal of the rate distortion optimization in the W-SVC codec is similar to the other codecs: find the optimum partitioning for a given frame, balancing the rate and the distortion to achieve a better performance. However, differently from the H.264/AVC, where the rate distortion is performed targeting a specific bitrate and/or quality, the W-SVC produces a scalable bitstream, that can be decoded at multiple spatio-temporal (ST) resolutions and qualities, and the same set of partitioning, modes and motion vectors is used for all decoding points (unless scalable motion [91] is used, but this case is not considered in this thesis). Therefore, the optimization has to be tailored to a wide range of bitrates and spatio-temporal resolutions. This makes the optimization for the W-SVC codec behave differently from that of the H.264/AVC codec, and produce different results.

In the W-SVC codec, the rate distortion (RD) optimization consists of three main elements: (i) selecting an appropriate partitioning of a MB into smaller blocks; (ii) selecting modes for each partition (forward, backward or bi-directional); and (iii) selecting a motion vector for each partition (or two motion vectors if the prediction is bi-directional). For each block, the motion estimation module attempts to find its best match in the reference frame(s), within a search window specified in encoding parameters. The best match for a block B_n^k is the block in the reference frame that minimizes the cost function:

$$J = D_n^k + \lambda \cdot R_{mv_{n \to n-\alpha}^k} \tag{3.1}$$

where the distortion D_n^k is a distortion metric, λ is the Lagrange multiplier and $R_{mv_{n \to n-\alpha}^k}$ is the rate required for encoding the motion vector $mv_{n \to n-\alpha}^k$, similar to the technique discussed in Sec. 2.1.1. The distortion metric is defined as the sum of absolute differences (SAD) between the elements of the block B_n^k and a block in the reference frame $n - \alpha$.

The difference in the optimization between the H.264/AVC and the W-SVC comes from the choice of the Lagrange multiplier λ, which controls the trade-off between rate and distortion. In many H.264/AVC implementations, λ is a function of the quantization parameter QP [106] (as seen in Sec. 3.1.4), and so the trade-off between rate and distortion is optimized for each bitrate. Thus, in the H.264/AVC codec, different QPs lead to different sets of motion vectors, since the optimization changes with the QP. For

Table 3.2.: Profile of the chosen partitions for Soccer and Crew sequences. The number of Temporal Decompositions (TD) used in the W-SVC is shown for each sequence.

	Soccer CIF			Crew CIF		
Partition Size	H.264 $QP = 20$	H.264 $QP = 28$	W-SVC 3 TD	H.264 $QP = 20$	H.264 $QP = 28$	W-SVC 3 TD
Intra	5.28%	4.73%	12.66%	21.16%	18.47%	12.66%
16×16	39.33%	61.44%	76.12%	23.86%	40.96%	77.78%
16×8 or 8×16	15.43%	17.02%	N/A	22.31%	23.58%	N/A
8×8	19.53%	9.54%	10.12%	16.42%	10.64%	9.00%
8×4 or 4×8	17.18%	6.41%	N/A	14.39%	5.87%	N/A
4×4	3.25%	0.86%	1.08%	1.89%	0.48%	0.54%

Table 3.3.: Profile of the chosen partitions for City and Harbour sequences. The number of Temporal Decompositions (TD) used in the W-SVC is shown for each sequence.

	City CIF			Harbour CIF		
Partition Size	H.264 $QP = 20$	H.264 $QP = 28$	W-SVC 5 TD	H.264 $QP = 20$	H.264 $QP = 28$	W-SVC 6 TD
Intra	0.45%	0.52%	3.33%	1.09%	0.79%	1.67%
16×16	46.50%	63.31%	88.98%	29.03%	39.36%	90.71%
16×8 or 8×16	12.92%	17.88%	N/A	16.58%	20.80%	N/A
8×8	21.76%	10.73%	6.86%	27.37%	21.59%	6.96%
8×4 or 4×8	15.18%	6.66%	N/A	22.72%	15.55%	N/A
4×4	3.18%	0.89%	0.83%	3.20%	1.89%	0.66%

instance, using a higher QP (lower quality) increases the frequency of larger blocks, as the rate of motion information has to decrease, compared to using a lower QP. Therefore, even though smaller blocks always leads to a better prediction, the mode decision engine seeks to find the optimal balance between the rate available for motion information and the rate available for the residual coefficients, for each target quality.

For the W-SVC codec, on the other hand, the same set of motion vectors has to be used for all bitrates, since the result of a W-SVC encoding is a single, scalable bitstream, that can be extracted at different qualities. Thus, the choice of λ has to consider all bitrates at a particular spatio-temporal resolution. Due to the scalability, sometimes a small decrease in quality at high bitrates has to be traded for a large increase at low bitrates. The net effect of this optimization is that larger blocks are preferred over smaller ones, since the former usually yields better overall results. This is different to what usually happens in a H.264/AVC codec. To illustrate this difference, Table 3.2 and Table 3.3 shows the percentage of different block sizes chosen for four different sequences at CIF (352×288) resolution, both for the H.264/AVC (using *IPP1* configuration and a quantization parameter $QP = 20$ and $QP = 28$) and the W-SVC codecs. A brief description of these video sequences can be found in Appendix C.

3.3. The HEVC Codec

The ISO-IEC/MPEG and the ITU-T/VCEG have recently formed the Joint Collaborative Team on Video Coding (JCT-VC) with the goal to develop a new video coding standard, called High Efficiency Video Coding (HEVC). In January 2010, a joint Call for Proposals was issued by the JCT-VC [58]. Later that year, the responses to the Call for Proposals were evaluated, and the most promising techniques were put together in a Test Model under Consideration (TMuC). The TMuC was later renamed HEVC Test Model (HM), and further refined several times in order to decide which techniques will be used in the final version of the standard. In the schedule of the standard, the final draft of the HEVC standard should be complete and ready for submission in January 2013.

The main goal of the codec is to offer a substantially higher compression capability than the H.264/AVC, targeting new applications, such as beyond high-definition resolutions, and improved picture quality, incorporating tools to support other colour spaces, samplings formats and pictures with higher dynamic range. Even before it is finalised, in a recent iteration ($HM4.0rc1$), the HEVC outperforms the H.264/AVC by 30% to 50% [75].

The test model used in this thesis, and presented in the following sections, is the $HM4.0rc1$ [24, 86], dated from July 2011. At the time of writing, the latest version is the $HM7.1rc1$, from April 2012. While there have been modifications in the codec, and there will likely be other modifications before it becomes a standard, the main characteristics remain: it adopts the well-known hybrid coding scheme (DPCM/DCT model), with advanced intra and inter prediction, followed by transform and entropy coding. Also, the main tools of the transcoding options presented on Chapter 6 are based on the coding unit and prediction unit structures (which will be explained in the following sections), which remain largely unchanged in $HM7.1rc1$, and thus could be adapted to the newest version of the codec.

3.3.1. Encoding Sequence

A simplified overview of the basic flow of encoding in the $HM4.0rc1$ codec is shown in Fig. 3.20. The codec also follows the DPCM/DCT model seen on Chapter 2 and its

basic building blocks are similar to those of the H.264/AVC. However, the techniques themselves used in each of the similar blocks are different than those in the H.264/AVC.

In the $HM4.0rc1$, each frame is divided in $N \times N$ blocks, called coding units (CUs), where N can be decided in the encoding parameters and it is transmitted in the bitstream. The maximum allowed size for the luma component is 64×64, and the CU at the largest size is called largest coding unit (LCU). The concept of the LCU is analogous to the concept of the macroblock in the H.264/AVC codec. Each coding unit defines: the type of prediction (inter, intra or skip), the transform unit (TU) and the type of the prediction unit (PU). As in the H.264/AVC, the LCUs are grouped in slices, and a collection of slices form a frame. Each slice is independently decodable (except for inter prediction). In the $HM4.0rc1$, the LCUs are encoded in raster scan order within a slice. Similarly to Sec. 3.1, it is assumed in this thesis one slice per frame.

The $HM4.0rc1$ allows a flexible partitioning for the CUs, and many partition sizes can be used for inter and intra prediction within a CU. Typically, the encoder tests several coding modes for a given CU and computes an associated cost to each coding mode. Then, a mode decision module decides which is the best mode to encode the CU, using a rate distortion criteria.

Similar to the H.264/AVC, after the CU mode is selected, the residual $R_n^k = B_n^k - P_n^k$ is computed, where B_n^k is the current block and P_n^k is the prediction for that block, derived from the CU partitioning and motion information. Then, the chosen transforms are applied to the residual, and the transformed coefficients are then quantized, entropy encoded and written in the bitstream. Afterwards, the encoder performs inverse quantization and transform operations, generating the reconstructed residual $R_n'^k$, which is then added to the prediction to generate the reconstructed block $B_n'^k = R_n'^k + P_n^k$. Once all CUs in a slice are encoded, a deblocking filter similar to the H.264/AVC [78] is applied to the frame to reduce the blocking artifacts. However, before finishing the encoding of the frame, two additional filters are applied to the frame, the Sample Adaptive Offset (SAO) [44, 86] and an Adaptive Loop Filter (ALF) [32, 86], in the form of a Wiener filter [30]. These are briefly explained later in this section. The output of these filters is finally added to the frame buffer, and can then be used as reference for inter prediction for the next frames. Again, an important encoder decision is the coding structure, which is discussed in Sec. 3.3.3.

One new tool in the $HM4.0rc1$ is the internal bit depth increase. When this technique is used, even if the original bit depth of the samples for the block B_n^k was 8 bits, they are

up-scaled and processed at an increased bit depth (usually, 2 bits of precision are added to the original bit depth), and thus all operations of residual, prediction, interpolation, transform and filtering are performed at the increased bit depth. It has been shown that this technique leads to superior rate distortion performance [86].

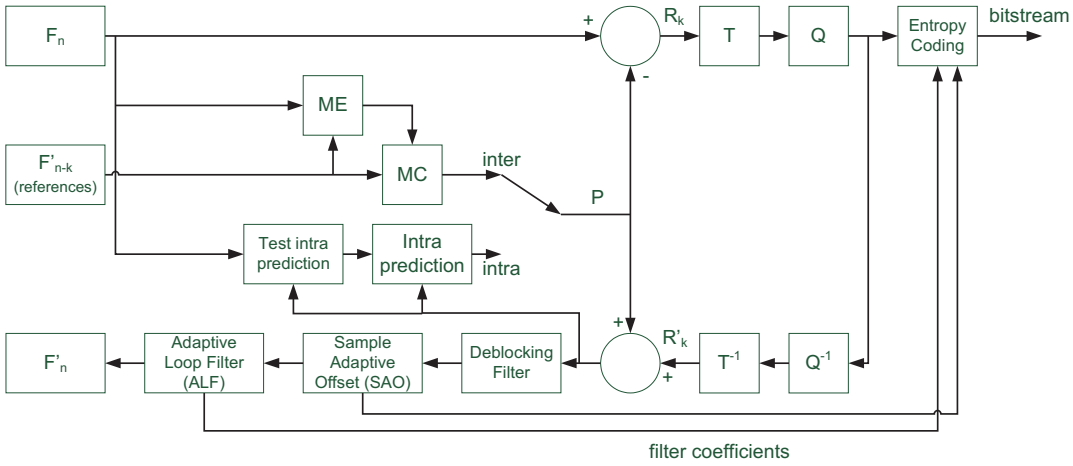


Figure 3.20.: $HM4.0rc1$ encoder diagram.

3.3.2. Decoding Sequence

The basic flow of decoding is shown in Fig. 3.21. The bitstream for each CU is decoded, and the inverse operations of entropy coding, quantizing and transform are applied, producing the reconstructed residual R'^k_n. Then, the prediction for this block, P^k_n, is generated, according to the signals decoded from the bitstream, and the reconstructed block is computed as $B'^k_n = R'^k_n + P^k_n$. Once all CUs are decoded, the remaining filters are applied (deblocking, Sample Adaptive Offset and Adaptive Loop Filter). The output of these filters is then ready to be displayed.

3.3.3. Slice Types and Coding Structures

Similar to the H.264/AVC codec, the $HM4.0rc1$ also defines three types of slices:

- I (intra): all CUs within this slice are encoded using intra prediction.
- P (predictive): all CUs within this slice are encoded using intra prediction or inter prediction using at most *one* motion vector and reference frame index.

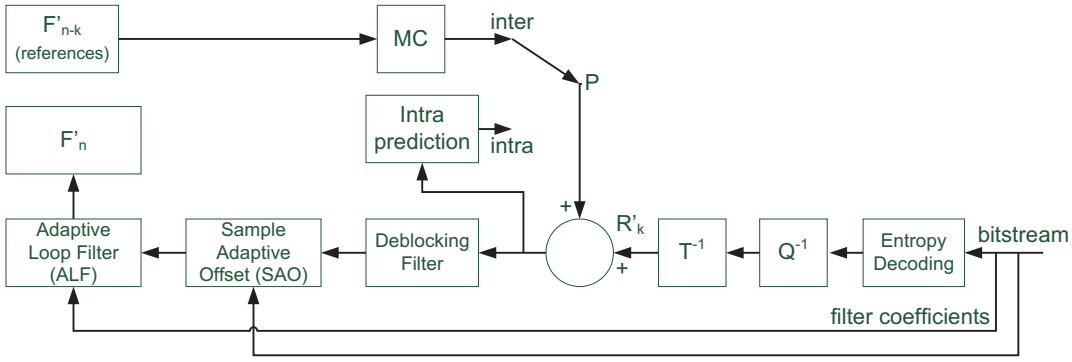


Figure 3.21.: $HM4.0rc1$ decoder diagram.

- B (bi-predictive): all CUs within this slice are encoded using intra prediction or inter prediction using at most *two* motion vectors and reference frame indices.

Again, as in the H.264/AVC, the slice mode is encoded in the slice header, providing full flexibility on the slice type choice. The $HM4.0rc1$ similarly defines two lists, "LIST 0" and "LIST 1", that stores reference frames for motion estimation. P blocks may use one motion vector pointing to a frame in "LIST 0", while B blocks may use one motion vector pointing to a frame in "LIST 0", one pointing to "LIST 1", or two motion vectors, each one pointing to a frame in each list, forming a bi-directional prediction. In the H.264/AVC codec, B slices are typically used when "future" frames in display order, relative to the current picture, are considered for inter prediction. In the HEVC codec, a "generalized P and B picture" (GPB) is defined, in which B-slices are used even if only past pictures are found in the reference lists. In fact, it can be used even if both motion vectors use the same reference frame.

While the same kinds of configurations as in the H.264/AVC can be used, two coding configurations receive more importance in the $HM4.0rc1$ codec: low-delay and random-access configurations.

In the low-delay configuration, the frames are encoded in display order, and thus only past reference frames can be used as reference for inter prediction. Apart from the first frame that is encoded as intra, all others are encoded as GPB, and the two reference lists ("LIST 0" and "LIST 1") are identical, containing only past frames. Also, the quantization parameter (QP) changes for each frame, depending on the temporal layer. This adjustment on the QP may be applied to a group of n pictures, and it is usually applied to a group of four pictures, in a hierarchical way. The base QP is used to encode the intra frame (with $QP = QP_I$), then frames in the first temporal layer are encoded

with $QP_{Lx} = QP_I + x$, where the suffix Lx indicates the temporal layer. When forming the two reference frame lists, the first index of the list is always the previous frame, regardless of the QP used to encode that frame, and the following reference frames in the lists are the closest frames encoded with a lower QP (i.e., higher quality). This configuration is shown in Fig. 3.22, where the usual four reference frames are used.

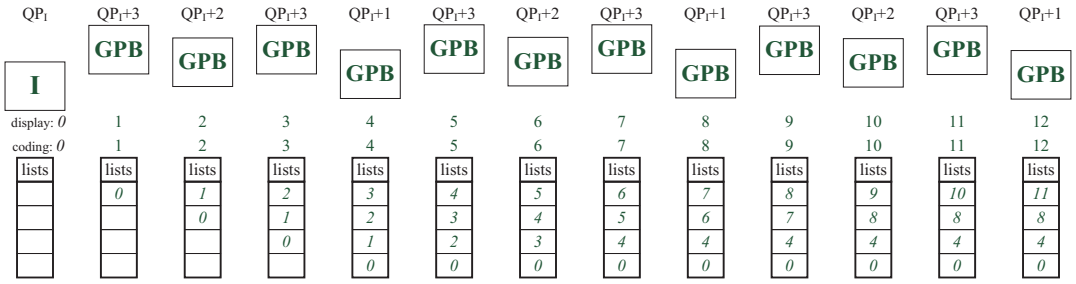


Figure 3.22.: $HM4.0rc1$ low-delay configuration. In this configuration, the two lists, "LIST 0" and "LIST 1", are identical, and are referred as "lists" in the figure. Also, the coding order and the display order are the same.

The second configuration is called random-access configuration. It uses hierarchical B structure, similar to the one shown in Fig. 3.5(c). The main differences for the HEVC is that all frames are encoded as GPB and the QP is also assigned in a hierarchical way (this time, using a group of eight frames). Also, note that frames in the highest temporal layer are never used as reference, while frames in the other temporal layers may be. This configuration is shown in Fig. 3.23.

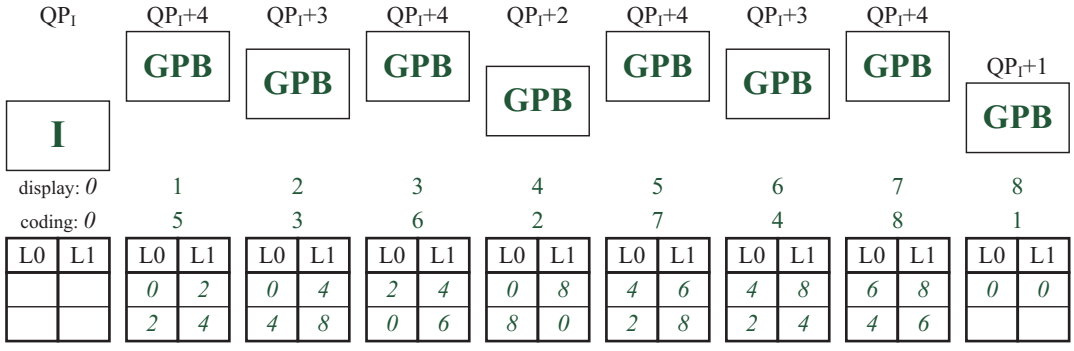


Figure 3.23.: $HM4.0rc1$ random-access configuration. In this example, two reference frames for each direction are used. Note that the codec always attempts to fill the two slots in each list, and that the coding order and display order are different.

3.3.4. Coding Unit Types and Partitioning

One of the main improvements in the HEVC codec is on the prediction module. In the HEVC, there are more options to partition the frame, yielding to a more accurate prediction. At the same time, other advances on the motion vector prediction optimises the coding of the motion information, so that the improved prediction can yield a gain in rate distortion sense.

In the $HM4.0rc1$, the coding unit (CU) is the basic unit used for inter and intra coding. The CU is always square, and its size may range from 8×8 to the LCU size (maximum of 64×64) pixels. The LCU size is an encoding parameter, however, in this thesis, the default value of 64×64 is always used.

A key concept in the CU partitioning is that it allows recursive splitting. In the $HM4.0rc1$ codec, each CU is encoded independently and defines (among other things): (i) the type of prediction; (ii) the prediction unit (PU); and (iii) if the CU is split. If the CU split flag is true, a $n \times n$ CU is always split in four CUs of $\frac{n}{2} \times \frac{n}{2}$, and each of these CUs is then encoded independently. If the CU is not split, then the parameters for the type of prediction and the prediction unit size must be transmitted in the bitstream, otherwise, these parameters do not need to be transmitted for that CU (but they must be transmitted for the children CUs, if they are not split). The LCU is defined as the depth 0 and, if it is split, the four children CUs are defined as depth 1, and so on. This is shown in Fig. 3.24.

The partitioning used for motion compensation is given by the prediction unit (PU), which is transmitted in the CU header. Similarly to the H.264/AVC, the $HM4.0rc1$ defines three types of prediction: inter, intra and skip, and the size of the prediction unit depends on the type of prediction. The following sections explain the prediction types and the PU sizes for an arbitrary CU of size $2N \times 2N$.

SKIP

The $HM4.0rc1$ defines a skip mode where the reconstructed block is equal to the prediction (i.e., no transform coefficient is transmitted and the residual is set to zero). In the skip mode, the prediction size is always equal to the CU size (i.e., the PU is always $2N \times 2N$), as shown in Fig. 3.26(a). The SKIP mode can be used at different depths of

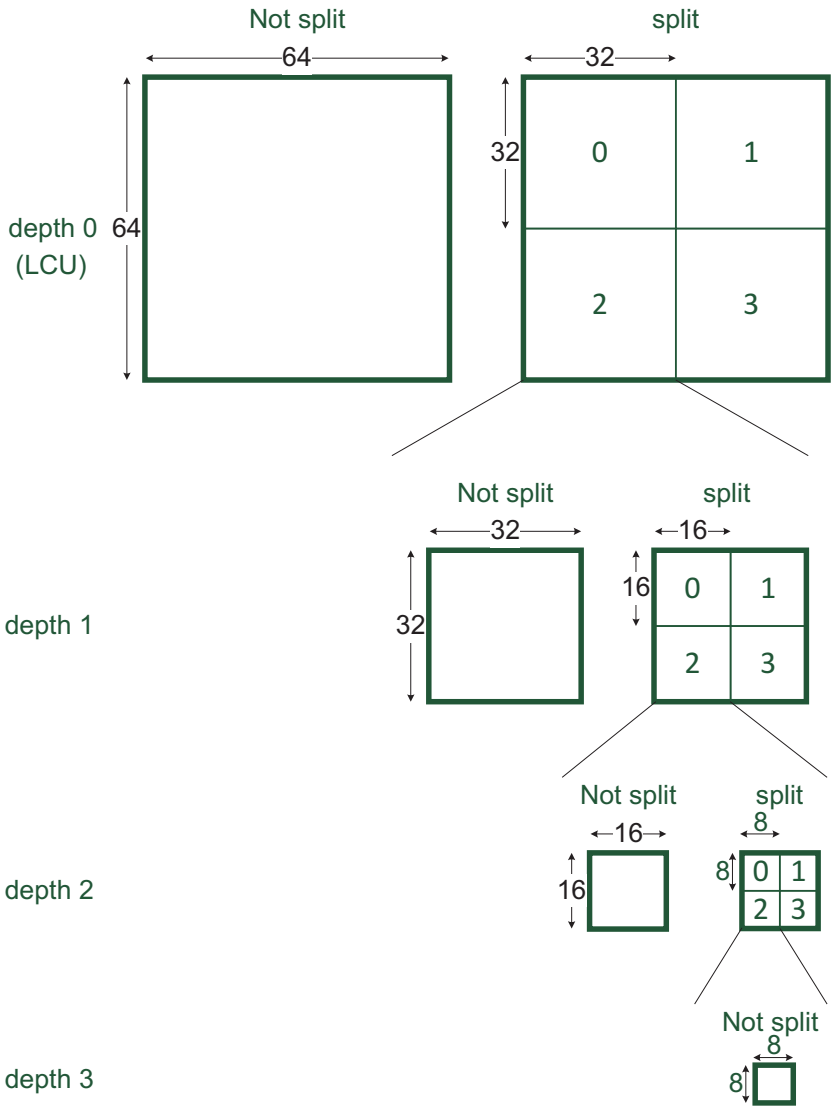


Figure 3.24.: Example of CU splitting on the $HM4.0rc1$. In the figure, the maximum size of a CU, 64×64 pixels, is used. Also, each time a CU is split, the four children CUs can be further split independently.

the tree, and thus effectively the different region sizes can be encoded as skip (64×64, 32×32, 16×16 and 8×8, in the usual CU size configuration).

Similar to the H.264/AVC codec, not all motion vectors can be used for the skip mode. In the H.264/AVC, a motion vector prediction process is used to derive the motion vector used in the skip mode. In the $HM4.0rc1$, a similar motion vector prediction process can also be used, but the usual configuration uses a "merge" process. In this process, the partition may select which motion vector it will use, among a pre-defined set of candidates, and this decision is then transmitted in the bitstream. The neighbouring

partitions that can be considered for merging (i.e., the set of candidates) are shown in Fig. 3.25, labeled from A to E (e.g., partition A means the top-left partition). Note that this is done for each PU, not CU.

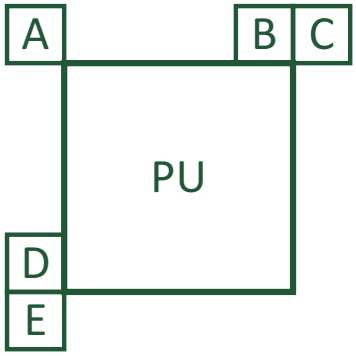


Figure 3.25.: Motion merge candidates in the $HM4.0rc1$. The labels A through E denote the neighbouring partitions that may be considered for merging for the current PU.

Intra modes

In the intra prediction, only two prediction unit sizes can be used, $2N \times 2N$ and $N \times N$, however, the latter can only be used in the last depth. The intra prediction can be thought as a generalization of the intra prediction in the H.264/AVC, with up to 33 directional prediction modes, plus a *DC* and planar mode. The number of possible modes depends on the size of the prediction unit. Prediction units with sizes 32×32, 16×16 and 8×8 may use all 35 intra modes, while PUs with sizes 64×64 and 4×4 may use only 3 and 17 modes, respectively. The available PU sizes for the intra modes are shown in Fig. 3.26(b). In addition to these, the HEVC also considers an intra PCM mode, similar to the H.264/AVC, which is applied to a PU size of $2N \times 2N$.

Inter modes

Finally, if inter prediction is used, several PU sizes (or modes) may be used: $2N \times 2N$, $2N \times N$, $N \times 2N$, $N \times N$, $2N \times nU$, $2N \times nD$, $nL \times 2N$ and $nR \times 2N$. The four partitions $2N \times nU$, $2N \times nD$, $nL \times 2N$ and $nR \times 2N$ are called collectively "asymmetric motion partitions" (AMP). They are called asymmetric because they divide the CU in two PUs of different sizes. The $2N \times nU$ mode divides the CU in a $2N \times \frac{N}{2}$ (top) and a $2N \times \frac{3N}{2}$ (bottom), the $2N \times nD$ divides the CU in a $2N \times \frac{3N}{2}$ (top) and a $2N \times \frac{N}{2}$ (bottom),

the $nL \times 2N$ divides the CU in a $\frac{N}{2} \times 2N$ (left) and a $\frac{3N}{2} \times 2N$ (right), and finally the $nR \times 2N$ divides the CU in a $\frac{3N}{2} \times 2N$ (left) and a $\frac{N}{2} \times 2N$ (right). These partitions are shown in Fig. 3.26(c).

In inter PUs, the encoder is free to either use the motion merge technique and, thus, signaling the complete set of motion vectors plus reference frame index by transmitting only the index to which the motion was merged, or to explicitly transmit the parameters, choosing motion vector and reference frames independently. Even if the parameters are transmitted explicitly, a process to predict the motion vectors is used. In the HEVC, the motion vectors have up to quarter-pixel precision, and the reference frames are interpolated using a 8-tap filter.

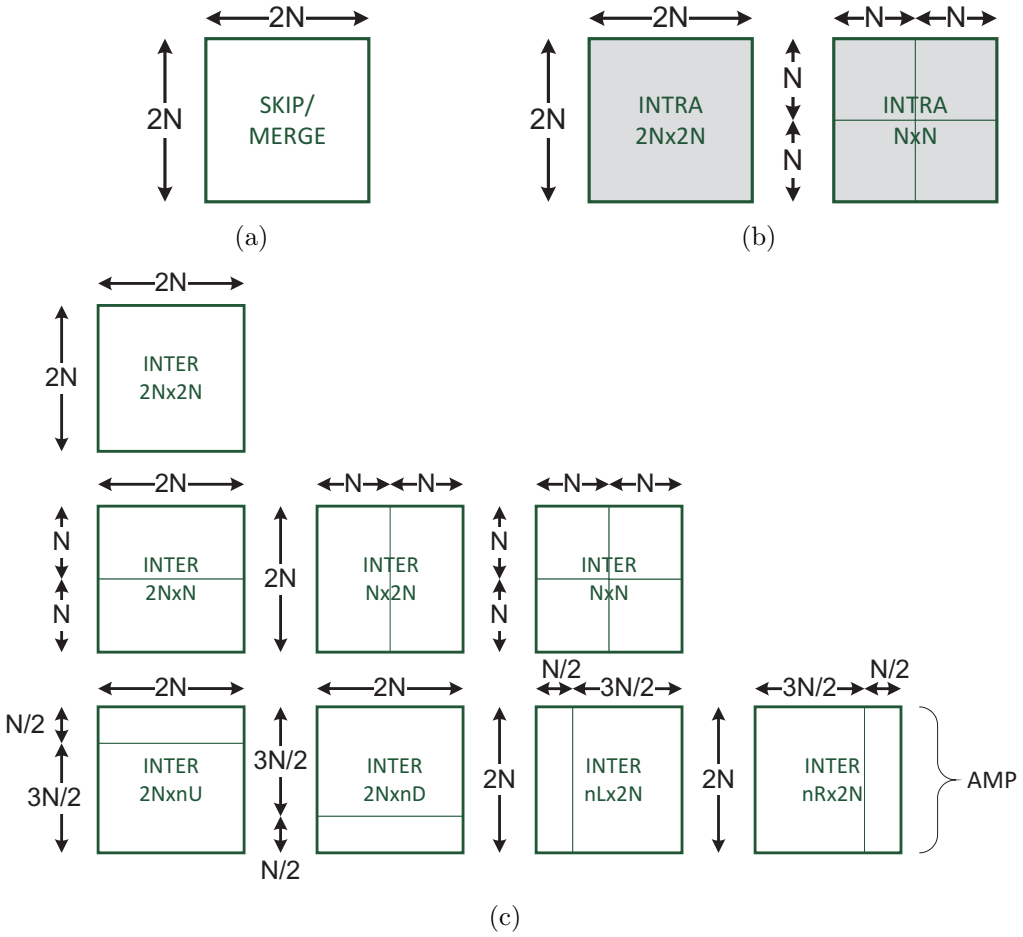


Figure 3.26.: Prediction Unit sizes in the $HM4.0rc1$ for: (a) SKIP mode; (b) Intra modes; and (c) Inter modes. The size of the CU is considered to be $2N \times 2N$.

3.3.5. Residual Coding and Loop Filtering

The transform and quantization process are applied to a block called transform unit (TU), which is considered at the CU level. The codec uses transform sizes ranging from 4×4 to 32×32, using a tree structure to transmit which transform is used for a given CU. The HEVC uses modified integer DCT transforms [86]. To quantize the transform coefficients, the HEVC uses a quantization parameter similar to the H.264/AVC, ranging from 0 (highest quality) to 51 (worst quality). Finally, the entropy coding is also very similar to the H.264/AVC, even using the same entropy coder, the Context-Adaptive Binary Arithmetic Coding (CABAC) [84].

The H.264/AVC standard introduced a mandatory in-loop deblocking filtering to reduce the blocking artifacts common to motion compensated prediction and transform coding (seen in Sec. 3.1.5). In the $HM4.0rc1$ codec, a similar filter is also applied, and, in addition, two other filtering strategies are applied in tandem: Sample Adaptive Offset (SAO) and Adaptive Loop Filter (ALF).

After the deblocking filter is applied, a sample adaptive offset (SAO) [44] is applied to the signal. The goal of the SAO is to reduce the distortion between the current original and reconstructed frames (after de-quantization and deblocking filter). The SAO computes optimal offsets for different regions of the image, using the information available to the encoder after the current frame is encoded. The offset for each category is then just added to the output of the deblocking filter, and these offsets and related information are transmitted in the bitstream so that the process can be repeated at the decoder.

After the SAO filter step, an adaptive loop filter (ALF) is applied to the signal [32]. The loop filter used is a Wiener filter, which is a well known data dependent linear filter that minimises the distortion (computed as the mean squared error) between the filter output and a target signal [30]. The encoder makes a decision of whether or not the filter is applied for each CU in a slice, and the filter coefficients are transmitted in the slice header (thus, the same filter is applied to all CUs to which the filter is applied). The output of the ALF is then ready to be displayed (at the decoder), or to be used as reference for the next frame (at the encoder).

3.4. Conclusions

This section introduces the three codecs that are used in the thesis: the H.264/AVC, the HEVC and the W-SVC. For each codec, the most important modules used in the transcoders are presented, as well as an overview of each codec encoding and decoding processes. Also, the main rate distortion optimization techniques of each codec is discussed, as well as its implications on transcoding.

The next chapter discusses motion vector approximation techniques, in particular in the scope of the H.264/AVC to W-SVC transcoder. As seen in this chapter, the coding configuration between these two codecs is very different, and this is a major issue in this transcoder.

Chapter 4.

Motion Vector Approximation Techniques

This chapter takes a closer look at motion vector reuse techniques in the scope of a cascaded pixel-domain transcoder. The main goal of these techniques is to reuse motion vectors from the source bitstream in the target codec. For several reasons, the incoming motion vectors may not be directly reused, and thus other techniques, such as Motion Vector Scaling or Motion Vector Composition, are used to allow the reuse of motion vectors. It is common for a second step of refinement, usually using a fast motion estimation algorithm, to be applied to these motion vectors - for this reason, the motion vectors generated by these methods are often called a *motion vector approximation*.

4.1. Motion Vector Reuse

Motion vector reuse is ubiquitous in transcoding [9, 135, 141]. The goal of this technique is to avoid a complex motion estimation operation in the transcoder, since motion estimation has already been performed in the source encoder. Motion estimation is the most time consuming module in most encoders [54], therefore, by avoiding it, the transcoding operation can be significantly sped up. The technique consists in reusing the motion vectors computed in the source encoder in the transcoder [21]. A simplified diagram of the technique is shown in Fig. 4.1.

There are several variants of this technique, depending on the characteristics of the source and target codecs, and the parameters used in both codecs. Consider that, for a particular block B_n^k, the source encoder used the motion vector $mv_{n\rightarrow n-\alpha}^k$. When this

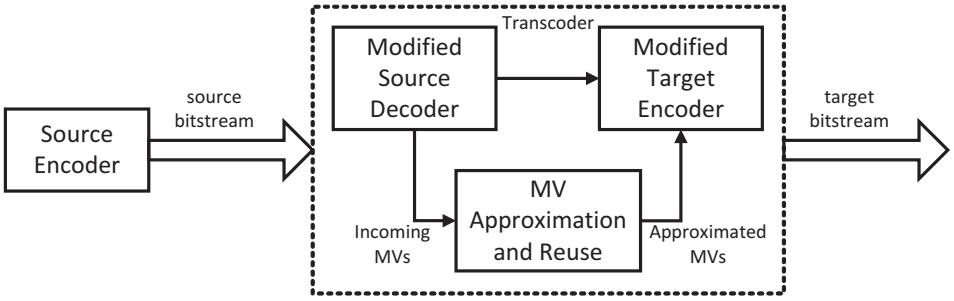


Figure 4.1.: Example of MV Reuse and Approximation Techniques.

block is being transcoded, the transcoder may use the same motion vector $mv^k_{n\to n-\alpha}$, provided that the spatial resolution is kept (i.e., that particular block has the same dimensions in both source and target encoders) and that the frame $n-\alpha$ is available to be used as a reference frame in the target codec. In this case, the motion vector $mv^k_{n\to n-\alpha}$ may be *directly reused* [21, 148]. However, since transcoding may have altered the reference frame $n-\alpha$, or the target encoder may have different partitioning options than the source encoder, the motion vector $mv^k_{n\to n-\alpha}$ may no longer be the optimal choice. Thus, it is very common to use a motion vector refinement, performing motion estimation centered on the motion vector $mv^k_{n\to n-\alpha}$. It is common for this motion estimation to use a fast algorithm [81, 149] or the exhaustive search, with a small search window [100, 117].

If the block B^k_n has a different dimension in the target encoder (for instance, it may have been reduced to quarter its size), the motion vector $mv^k_{n\to n-\alpha}$ cannot be directly reused, as it would refer to a different area of the image. However, it can still be reused by mapping the motion vectors to a different area. This motion vector mapping technique, along with motion vector reuse, is used in several transcoders that use spatial resolution reduction [21, 115].

Finally, it is also possible that the motion vector $mv^k_{n\to n-\alpha}$ cannot be directly reused because the frame $n-\alpha$ is no longer available as a reference (either because it has been dropped, in temporal resolution reduction transcoding, or because the coding configuration has changed in the transcoder). In this case, the motion vectors cannot be directly reused, but they can be exploited to derive new motion vectors to be used in the transcoder. Techniques such as motion vector composition [112, 149] and motion vector scaling [142] have been successfully used in transcoders where the reference frames do not match [10, 71, 112, 117, 149].

It is also possible to make use of motion vector prediction techniques, such as those used by some fast motion estimation algorithms [132]. In this case, the motion vectors

for the current partition are derived using the motion vectors from the neighbouring blocks that have already been coded.

While MV Approximation techniques are important for most transcoders, they are mainly used in this thesis in the scope of the H.264/AVC to W-SVC transcoder. For this transcoder, the main issue that prevents the reuse of motion vectors is that the reference frames between the incoming and target bitstreams might not match. For this reason, the techniques to tackle this reference frame mismatch are further discussed in the next sections.

4.2. The state of the art

The main technique to cope with the reference frame mismatch is the motion vector composition [112, 149]. Usually, this technique is applied in temporal resolution reduction transcoding [71, 112, 117], where some frames are dropped in the transcoder. Even though the reason for mismatch is different from the one in the H.264/AVC to W-SVC transcoder, the goal of the technique is the same, and they can be used for both purposes. Another simpler technique that is also able to cope with the reference frame mismatch is the MV Scaling [142], which is explained in the following section.

4.2.1. Motion Vector Scaling

In this technique, the movement between the frames is modeled as linear, and the motion vector is just scaled to the new reference frame. Since this assumption does not always hold, this is not a particularly accurate technique, but its advantage is its low computational complexity and the possibility to always produce a candidate, given a starting motion vector. The scale factor is directly proportional to the distance between the original and target reference frames and the current frame [142]. Let n be the current frame, $n-\alpha$ be the original reference frame used and $n-\beta$ be the target reference frame. The technique is shown in Fig. 4.2 and the equation for scaling is:

$$mv^k_{n\to n-\beta} = \left(\frac{\beta}{\alpha}\right) \cdot mv^k_{n\to n-\alpha} \tag{4.1}$$

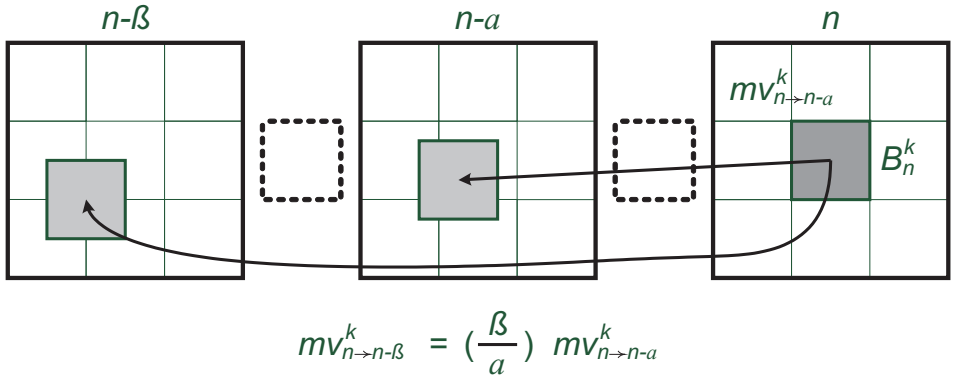


Figure 4.2.: Example of Motion Vector Scaling.

4.2.2. Motion Vector Composition

In this technique, the idea is to follow the movement between the current frame and the target reference frame, taking into account the motion vector path through the dropped frames [112, 149].

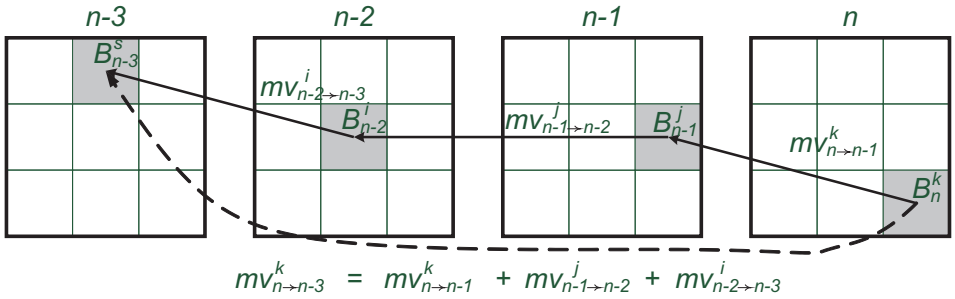


Figure 4.3.: Example of Motion Vector Composition.

An example of MV Composition is shown in Fig. 4.3. In this example, the goal is to generate the motion vector $mv^k_{n\to n-3}$ for the block B^k_n. However, the incoming motion vector for this block is $mv^k_{n\to n-1}$. Starting from the given block in frame n, the MV Composition checks where its motion vector points to in frame $n-1$. Then, it gets a new motion vector from the source codec in the partition corresponding to that position in frame $n-1$ ($mv^j_{n-1\to n-2}$ in the figure), and then checks where this motion vector points to in frame $n-2$. The process iterates until the desired reference frame is reached. The final motion vector is then composed as:

$$mv^k_{n\to n-3} = mv^k_{n\to n-1} + mv^j_{n-1\to n-2} + mv^i_{n-2\to n-3} \tag{4.2}$$

The motivation for this algorithm is that the current block B_n^k can be accurately predicted using block B_{n-1}^j, which can be accurately predicted by the block B_{n-2}^i, which finally can be accurately predicted by block B_{n-3}^s. Thus, the block B_{n-3}^s may be a good prediction for the current block B_n^k, and this motion vector is given by Eq. 4.2.

There are three main issues that may arise in the MV Composition:

1. The motion vectors generally do not point to a position in the grid. Thus, the partition pointed to by the motion vector may overlap with other partitions.
2. Even if the new position matches the grid, there may be more than one motion vector for the matching partition. As an example, the original starting partition in frame n may have been 16×16, while the partition in frame $n-1$ may have been coded as two 16×8 partitions.
3. A macroblock in the composition process may have been coded in intra mode and, therefore, may have no motion vector, or its motion vectors may use invalid reference frames (out of the range from current frame to the target reference frame, or in an invalid direction).

The main difference between the two most commonly used MV Composition algorithms, Forward Dominant Vector Selection (FDVS) [149] and Telescopic Vector Composition (TVC) [112], is the way they tackle the grid problem. The second issue, of having multiple motion vectors in the pointed region, is not discussed in the original papers that describe these algorithms as this was not possible to the particular transcoders that they were developed for, and therefore were not an issue at the time. However, a common solution developed in other works [117] is to use a weighted average on the motion vectors on the region. The equation is:

$$mv_{n \to n-\alpha}^k = \frac{\sum_{i \in K} w_i \cdot mv_{n \to n-\alpha}^i}{\sum_{i \in K} w_i} \tag{4.3}$$

where i is the index of a subpartition within the partition B_n^k, K represents the set of subpartitions within the current partition B_n^k and w_i represents a weight, which is the area occupied by the partition with index i.

Forward Dominant Vector Selection

This technique was first proposed in a H.263 transcoder [149, 147] and used in a scenario where the frame rate of the sequence would be reduced during transcoding, which would cause the reference frame mismatch. The Forward Dominant Vector Selection (FDVS) will select the position on the grid that has the largest overlapping area with the position estimated in an intermediate step of MV Composition. An example is given in Fig. 4.4.

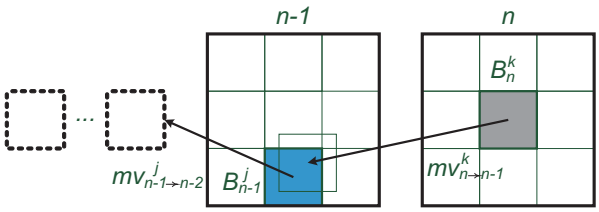


Figure 4.4.: Example of Forward Dominant Vector Selection. In this case, the motion vector considered in the second step is the motion vector $mv^j_{n-1\to n-2}$.

Telescopic Vector Composition

This technique was first proposed in a MPEG-1, MPEG-2 to H.261 and H.263 transcoder [112]. It is also used in a scenario where the frame rate of the sequence is reduced during transcoding, causing the reference frame mismatch. The Telescopic Vector Composition (TVC) always look for motion vectors in the same position as the starting partition, therefore avoiding the grid-matching problem. As such, the final motion vector is just the sum of the motion vectors of that particular partition in the dropped frames. An example is given in Fig. 4.5.

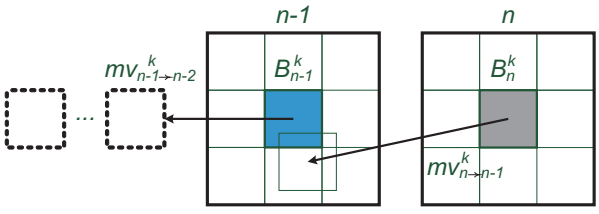


Figure 4.5.: Example of Telescopic Vector Composition. In this case, the motion vector considered in the second step is the motion vector $mv^k_{n-1\to n-2}$.

Other MV Composition Algorithms

The two techniques mentioned, FDVS and TVC, are the main algorithms existent in the literature to perform MV Composition. Since then, other techniques were proposed to improve MV Composition, building upon FDVS and TVC, but the improvements are either towards the way a block is aligned to the grid at each step of composition [71, 144] or to allow the techniques to be used with different block sizes [80, 117].

A different way to tackle the grid matching problem was proposed [144], used in the context of a frame rate reduction transcoder, called Efficient-Forward Dominant Vector Selection (E-FDVS). In E-FDVS, when deciding the partition for the next iteration, both the current composed motion vector and a compensation vector are used. The compensation vector is defined as the difference vector between the dominant block in the current iteration and the motion vector at a previous iteration. Another algorithm to tackle the grid matching problem was proposed [71], also in the context of a frame rate reduction in H.264/AVC transcoding. Instead of using the whole block pointed to the motion vector $mv^k_{n\rightarrow n-1}$ when deciding the dominant block for frame $n-1$, this algorithm attempts to use just the area relevant to the block B^k_n to contribute to the selection of the dominant block.

In order to cope with different block sizes, an algorithm called Block-Adaptive Motion Vector Re-sampling (BAMVR) was proposed [117], also in the context of frame rate reduction in H.264/AVC transcoding. It tackles multiple block sizes in the following manner: consider a block B^k_n, of size 8×8, consisted of the 4×4 partitions $\{k0, k1, k2, k3\}$ (the algorithm works in a similar way if other block sizes are used, but more 4×4 partitions would be needed - also, it uses 4×4 partitions because that is the minimum size for a motion compensated block in the H.264/AVC). Then, in order to compose the motion vector for B^k_n, it applies the FDVS algorithm for each of the four sub-blocks $k0$ through $k3$, resulting in the composed motion vectors $\left\{mv^{k0}_{n\rightarrow n-\beta}, mv^{k1}_{n\rightarrow n-\beta}, mv^{k2}_{n\rightarrow n-\beta}, mv^{k3}_{n\rightarrow n-\beta}\right\}$. Finally, the final composed motion vector $mv^k_{n\rightarrow n-\beta}$ is computed as the average of these motion vectors. Another solution to cope with different block sizes, called Multi-Level Bilinear Scheme (MLBS), was proposed [80]. In this algorithm, each time a block is found to contain multiple motion vectors, these motion vectors are combined using bilinear interpolation.

The aforementioned works are all in the context of frame rate reduction transcoding, and all consider the case of transcoding a *IPP1* incoming bitstream to another *IPP1* target bitstream, at a lower frame rate. Therefore, they are only interested in producing

a forward motion vector, and do not provide any solution to produce a backward motion vector. Recently, other algorithms were proposed to transcode H.264/AVC bitstreams to temporal scalable H.264/AVC SVC bitstreams [10, 47]. These works target transcoding to a *hierarchical* coding structure, in which B frames are used. Thus, both forward and backward motion vectors need to be approximated.

In order to achieve temporal scalability, the H.264/SVC codec [108, 66] defines two main types of coding structures: hierarchical B-frame, in which the frames in the enhancement layers are encoded as B-frames (using bi-directional prediction, using as reference both previous and future frames, in display order), and hierarchical zero-delay structure, in which the enhancement layers are encoded as P-frames (using only one past reference frame). A transcoder from H.264/AVC bitstreams to temporal scalable H.264/SVC bitstreams was proposed [10], in which the macroblock mode of the H.264/AVC is reused in the H.264/SVC. In order to approximate motion vectors for the cases in which the reference frame between source and target codecs do not match, it uses a modified FDVS algorithm, the BAMVR [117], to produce forward motion vectors, but does not attempt to approximate the backward motion vectors. After MV Composition is performed, a MV refinement is considered starting at the composed MV, for the forward direction, and at the $(0, 0)$ MV, for the backward direction. Furthermore, the work only considers transcoding from *IPP* coding structures using 1 reference frame, and limits the GOP size of the target hierarchical structure to the GOP sizes of 2, 4 and 8.

Another transcoder targeting temporal scalable H.264/SVC bitstreams was proposed [47]. This simple transcoder is mainly based on adjusting the search window of the H.264/SVC codec, using both the length of the H.264/AVC motion vectors and the temporal layer of the current frame being encoded. All possible modes for the H.264/SVC codec are still tested in the transcoder, and no attempt to approximate the incoming motion vectors is made. Furthermore, the transcoding algorithm is only applied to the higher temporal layers, and it only considers transcoding from *IPP* coding structures using 1 reference frame to a hierarchical zero-delay structure in the H.264/SVC.

4.3. Developed Techniques

The techniques described here were developed and adapted to be used in a H.264/AVC to W-SVC transcoder [4, 6]. The main reason that prevents the transcoder from reusing

motion vectors is the reference frame mismatch. In the transcoder, the frames are not dropped, but a different coding configuration is used in the W-SVC codec. As seen in Chapter 3, the W-SVC codec needs to use a hierarchical structure to select the reference frames, where in the H.264/AVC codec other structures are more common, such as *IPP1*, *IPP5* or *IBBP* (although a hierarchical structure can be used in the H.264/AVC - in this case, the H.264/AVC motion vectors can be directly reused, but this may cause other issues that are discussed in Chapter 5).

One peculiarity of the H.264/AVC to W-SVC transcoder is that, depending on the wavelet filters the W-SVC codecs uses, it needs both a forward and a backward motion vector to perform motion compensated temporal filtering (as seen in Sec. 3.2.6). For this reason, some techniques were developed so that the transcoder may derive backward motion vectors even if only forward motion vectors are found in the H.264/AVC bitstream [6].

Finally, while the techniques described here are developed for the W-SVC transcoder, they could be used in any other transcoder where the reference frame mismatch is an issue.

4.3.1. Motion Vector Composition

As seen in Sec. 4.2.2, the typical motion vector composition algorithms have some limitations: at each iteration, a motion vector may not point to a location in the grid; there may be more than one motion vector in a given partition, possibly with different reference frames; and a macroblock in the process may have been coded in intra mode or it may have motion vectors that use invalid reference frames. Although some solutions to tackle with the grid matching problem were proposed [71, 144], to cope with variable block sizes [80, 117] and to cope with intra frames [112, 149], none of these algorithms are suited to tackle the multiple reference frames problem, nor are they able to produce backward motion vectors. Moreover, all of these algorithms cope with intra frames using the possibly wrong assumption that the motion for that block is zero, and use the $(0, 0)$ motion vector for that block.

All of the mentioned MV Composition algorithms are applied to bitstreams encoded with an *IPP* structure using one reference frame, either targeting another *IPP1* structure [71, 80, 112, 117, 144, 149], or even an hierarchical structure, but not producing any backward motion vector [10, 47]. Also, most of the works on other transcoders where

reference frame mismatch could be an issue, such as H.264/AVC to MPEG-2 transcoders [70, 114, 118] or H.264/AVC to MPEG-4 transcoders [55], are also applied to bitstreams encoded with an *IPP1* coding structure (and targeting the same coding structure in the target codec).

To deal with more complex motion structures, a new MV Composition method was developed that is similar in spirit to Forward Dominant Vector Selection (FDVS) [149] and Telescopic Vector Composition (TVC) [112], extended to work with different coding configurations, multiple reference frames and variable block sizes occurring in H.264/AVC motion information [3, 6]. It combines both FDVS and TVC to overcome the issues listed before, and it is explained in the following sections.

MV Composition based on FDVS supporting multiple reference frames and variable block sizes (MRVB-FDVS)

The aim here is to compose a MV from the current frame n to the W-SVC reference frame $n - \beta$ ($\beta > 0$). The example here is constructed based on the test of a 16×16 partition for B_n^k. The algorithm works in the same way for other partition sizes.

The algorithm is based on an ordered motion vector list, which starts empty ($list = \{\cdot\}$). At each step of composition, the list is refreshed. In the first step, all motion vectors from the source bitstream within the current partition $P_0 = B_n^k$ are considered. These motion vectors are grouped according to their reference frames, each group containing motion vectors that point to the same reference frame. The algorithm then uses a grouping algorithm to select the motion vector for that partition. The input of this grouping algorithm are all motion vectors within that block that point to a single reference frame, and the output is a single motion vector that represents that group. Different grouping algorithms are discussed in Sec. 4.3.2.

The motion vectors output by the grouping algorithm (one for each reference frame found within the block) are added into the list only if their reference frames are between the frames n and $n - \beta$, ordered such that the elements towards the end of the list have reference frames that are closer to $n - \beta$. This first step is shown in Fig. 4.6. In the figure, $k0$ and $k1$ denote the two 16×8 partitions within B_n^k. In the example, the partition B_n^k has two motion vectors, which point to two different reference frames, $n-1$ and $n - 2$. They are classified in two groups, and there is no need to group the motion vectors because each group has only one motion vector, and thus $mv_{n \to n-1}^{k} = mv_{n \to n-1}^{k0}$

and $mv^{k}_{n\to n-2} = mv^{k1}_{n\to n-2}$. The two motion vectors are added to the ordered list, which is now:

$$list = \left\{mv^{k}_{n\to n-1}, mv^{k}_{n\to n-2}\right\} \tag{4.4}$$

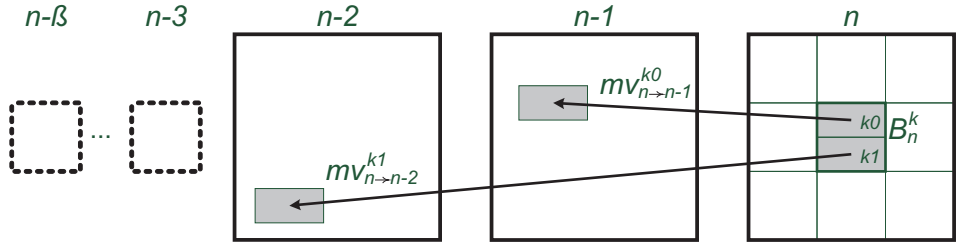


Figure 4.6.: Step 1 of MV Composition using MRVB-FDVS.

In the next step, the algorithm pops the last motion vector on the list (i.e., the one closer to the target reference frame, $mv^{k}_{n\to n-2}$, in the example) and adds it into the current composition group CG. The composition group is the set of motion vectors that is effectively used to compose the motion vector from frame n to the target frame $n-\beta$. It also starts empty and, at this point in the example, it is:

$$CG = \left\{mv^{k}_{n\to n-2}\right\} \tag{4.5}$$

The new position of the partition in the frame $n-2$ (in this example) is calculated as $P_i = P_0 + \sum_{j\in CG} MV_j$. In the example, we have $P_1 = P_0 + mv^{k}_{n\to n-2}$. To continue the composition, the partition P_1 is aligned to the grid, using the position that has the highest number of overlapping pixels with P_1. Here, it is important to notice that, since the source codec studied in this thesis is the H.264/AVC, and the minimum block size used for inter prediction in this codec is 4×4 pixels, the partition can be aligned to the 4×4 grid, instead of aligning it using the same partition size. Both cases are considered in the evaluation, later in this chapter. This step is shown in Fig 4.7, where P_1 is aligned to the block B^{j}_{n-2}. In the example, this macroblock is also partitioned in two 16×8 blocks (the partitions B^{j0}_{n-2} and B^{j1}_{n-2}), and both of its motion vectors point to frame $n-3$. In this case, in order to have only one motion vector for this step, the motion vectors are grouped, using one of the algorithms seen in Sec. 4.3.2.

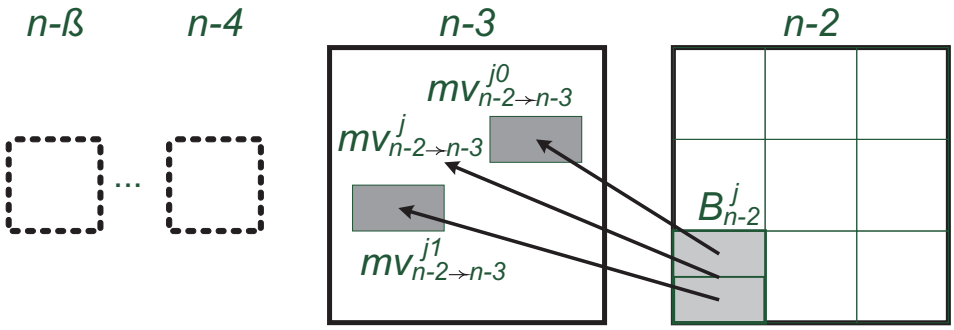


Figure 4.7.: Step 2 of MV Composition using MRVB-FDVS. In this case, the used motion vector for the partition B^j_{n-2}, $mv^j_{n-2\to n-3}$, depends on the grouping algorithm used (see Sec. 4.3.2).

The resulting motion vector, coming from the grouping algorithm, is then added to the ordered list, which is now:

$$list = \left\{mv^k_{n\to n-1}, mv^j_{n-2\to n-3}\right\} \tag{4.6}$$

In the next step, the element at the back of the list, $mv^j_{n-2\to n-3}$, is popped and the algorithm repeats again until the target reference frame, $n-\beta$, is reached. The composed motion vector is:

$$mv^k_{n\to n-\beta} = \sum_{i\in CG} MV_i \tag{4.7}$$

If, at some intermediate step, the motion vectors cannot be used (either because they point outside the desired range, or the block had been coded in intra mode), the algorithm will remove all motion vectors from CG that were added in any subsequent step to the one in which the currently last element of the ordered list has been added. Then it will pop another MV from the ordered list, using it in the same way as before. Following the example, if the partition pointed by the motion vector $mv^j_{n-2\to n-3}$ was encoded in intra mode, then the algorithm will refer to the ordered list, given in Eq. 4.6, and pop the currently last element, the motion vector $mv^k_{n\to n-1}$ (recall that the first action when considering a motion vector is to pop it from the list, and thus $mv^j_{n-2\to n-3}$ was already removed from the list). At this point, all motion vectors from CG that were

added after the algorithm pushed the motion vector $mv^k_{n\to n-1}$ into the ordered list will be removed. In this case, this means the motion vectors $mv^k_{n\to n-2}$ and $mv^j_{n-2\to n-3}$, and, thus, the list will be empty again.

The algorithm continues in such a manner until the reference frame $n-\beta$ has been reached or until the list is empty. Once the target reference frame has been reached, the algorithm stops, and the remaining elements in the lists are cleared, otherwise, if the list is empty, the algorithm does not return a motion vector.

MV Composition based on TVC supporting multiple reference frames and variable block sizes (MRVB-TVC)

This method is very similar to the method described in the previous section. The only difference is that, at each composition step, the new position is estimated as $P_i = P_0$. In other words, the position of the block is considered to be the same throughout the frames. The final motion vector is composed in the same way, using Eq. 4.7.

Composing MVs in *IPP* coding configuration with 1 reference frame

For the simple case of *IPP1* coding configuration, only the forward motion vector is available from the source bitstream. However, in order to produce a backward motion vector for the target codec, the following algorithm is applied. Let frame n be the current frame, frame $n-\beta$ be the target frame for the forward motion vector, frame $n+\beta$ be the target frame for the backward motion vector (since the codec uses a hierarchical coding configuration, the distance between the current frame and both forward and backward reference frames is the same), and B^k_n be the current partition, for which it is desired to compose the motion vectors. Then, we have:

- For the forward motion vector, MRVB-FDVS is directly applied for motion vector composition, starting at block B^k_n and generating the approximated MV $mv^k_{n\to n-\beta}$.
- For the backward motion vector, the approach is changed slightly. Since there is no backward motion vector, the transcoder applies MRVB-TVC starting at block $B^k_{n+\beta}$, generating a motion vector $mv^k_{n+\beta\to n}$. Then, it uses inversion on this motion vector to get the approximated motion vector $mv^k_{n\to n+\beta}$. TVC is used instead of FDVS because the latter cannot guarantee that the composition will arrive at the same block, which is not the case with the former.

Composing MVs in *IPP* coding configuration with multiple reference frames

In the case of multiple reference frames, the composition method is performed in two phases. The second phase is executed only if the composition is unsuccessful in the first phase. The first phase is similar to the composition for *IPP* coding configuration, with the main difference that the algorithm may record composed MVs for frames beyond the target reference frame.

First, the algorithm tries to compose a motion vector starting at the block B_n^k to the desired reference frame $n-\beta$, using MRVB-FDVS. If it cannot compose such motion vector, it allows the last iteration to go beyond the target reference frame, generating the motion vector $mv^k_{n\to n-\beta-\gamma}$ ($\gamma > 0$). Then, it will try to compose a motion vector starting at $B^k_{n-\beta}$ to the reference frame $n-\beta-\gamma$, using MRVB-TVC. This is shown in Fig. 4.8. Finally, the final composed motion vector is:

$$mv^k_{n\to n-\beta} = mv^k_{n\to n-\beta-\gamma} - mv^k_{n-\beta\to n-\beta-\gamma}. \tag{4.8}$$

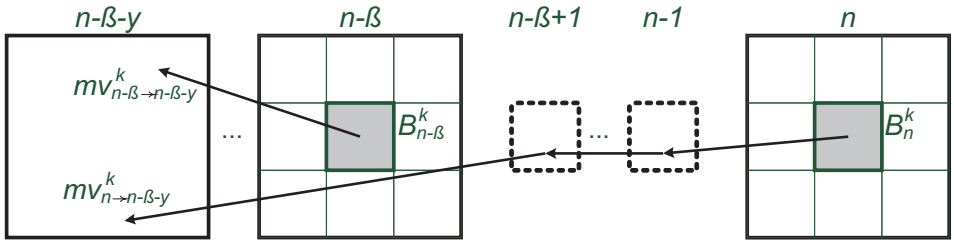


Figure 4.8.: Example of MV Composition in two phases. The goal is to compose the MV $mv^k_{n\to n-\beta}$ for block B^k_n. However, for some reason, the composition can only compose the MV $mv^k_{n\to n-\beta-\gamma}$ in the first phase (using MRVB-FDVS). The second phase attempts to compose the MV $mv^k_{n-\beta\to n-\beta-\gamma}$ for block $B^k_{n-\beta}$, using MRVB-TVC. The final composed MV is then: $mv^k_{n\to n-\beta} = mv^k_{n\to n-\beta-\gamma} - mv^k_{n-\beta\to n-\beta-\gamma}$.

The same strategy is applied to compose the backward motion vector. For this case, since there is no backward motion vector in the block B^k_n to start the composition, the algorithm applies MRVB-TVC starting at block $B^k_{n+\beta}$. If the desired reference frame n cannot be reached, it allows the last iteration to go beyond this frame, generating the motion vector $mv^k_{n+\beta\to n-\gamma}$. Then, it will try to compose a motion vector starting at B^k_n

to the reference frame $n-\gamma$, also using MRVB-TVC. Finally, the final composed motion vector is:

$$mv^k_{n\to n+\beta} = (-1)\cdot\left(mv^k_{n+\beta\to n-\gamma} - mv^k_{n\to n-\gamma}\right) \tag{4.9}$$

Composing MVs for other coding configurations

In the more general case, when B-frames are present in the stream, the composition for a given block B^k_n to a target reference frame $n-\beta$, for the forward motion vector, and $n+\beta$, for the backward motion vector, works as follows:

- If the block B^k_n has a forward motion vector, then MRVB-FDVS is used starting at this block. If a direct motion vector $mv^k_{n\to n-\beta}$ cannot be composed in the first phase, the algorithm tries to compose the motion vector using the second phase.
- If the block B^k_n does not have a forward motion vector, then the algorithm checks if the block $B^k_{n-\beta}$ has a backward motion vector. If this is the case, then MRVB-TVC is used to compose the motion vector $mv^k_{n-\beta\to n}$, which will then be inverted to get the desired motion vector. Similarly, a second phase may be performed if the composition is not successful in the first phase.
- If the block B^k_n has a backward motion vector, MRVB-FDVS is performed starting at this block, also allowing two phases.
- If the block B^k_n does not have a backward motion vector, then the algorithm checks if the block $B^k_{n+\beta}$ has a forward motion vector. It will then use MRVB-TVC to compose the motion vector $mv^k_{n+\beta\to n}$, which will be inverted to get the desired motion vector. Similarly, a second phase may be performed if the first phase is unsuccessful.

Note that the second phase may be particularly useful in the case of *IBBP* configuration, as shown in Fig. 4.9. However, even with the second phase, the MV Composition method proposed here cannot always succeed in generating a candidate. A number of reasons may cause this, like a high number of intra MBs, or very different reference frames.

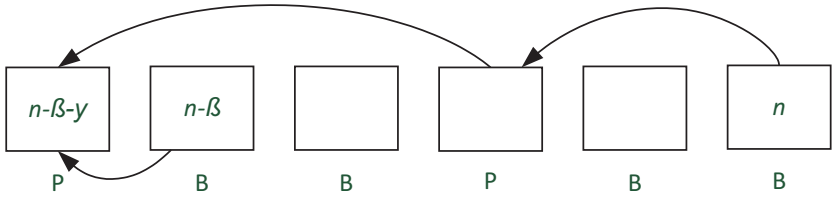


Figure 4.9.: Example of MV Composition for *IBBP* configuration. In this figure, the arrows point to the reference frame used by each frame. The goal is to compose the MV $mv^k_{n\to n-\beta}$. In this example, the first phase of composition will be unsuccessful for every block in frame n, being able only to generate a MV $mv^k_{n\to n-\beta-\gamma}$. Thus, the second phase is applied to adjust the composition.

4.3.2. Grouping Algorithms

When selecting the motion vector for a given block B^k_n, there may be more than one motion vector within that block. The input of the grouping algorithms are all motion vectors within B^k_n that point to a given reference frame $n-\beta$ (i.e., the set: $\{mv^{i0}_{n\to n-\beta}\ ,\ mv^{i1}_{n\to n-\beta}\ ,\ \ldots\ ,\ mv^{iN}_{n\to n-\beta}\}$), and the output is a single motion vector that represents that group ($mv^k_{n\to n-\beta}$). Here, two solutions to this problem are considered: using the average motion vector and the mode.

Using the average motion vector is a common solution found on the literature [117]. Usually, it consists on calculating a weighted average of motion vectors within the block, with weights that are proportional to the size of their corresponding partition area. The equation for the average is:

$$\overline{mv^k_{n\to n-\beta}} = \frac{\sum_{i\in K} w_i \cdot mv^i_{n\to n-\beta}}{\sum_{i\in K} w_i} \tag{4.10}$$

where i is the index of a subpartition within the partition B^k_n, K represents the set of subpartitions within the current block B^k_n and w_i represents a weight, which is the area occupied by the partition with index i.

However, the problem with the average motion vector is that it may not represent any of the motion vectors in the block [103, 102], and therefore take the composition to an incorrect path. This could be solved by using a convex solution, in the form of:

$$mv^{k}_{n\to n-\beta} = \left\{ mv^{i}_{n\to n-\beta} \;\mid\; i = \underset{i\in K}{\arg\,min} \left(\left\| mv^{av}_{n\to n-\beta} - mv^{i}_{n\to n-\beta} \right\|_2 \right) \right\} \tag{4.11}$$

where $mv^{av}_{n\to n-\beta}$ is the average motion vector for the partition B^{k}_{n} and $mv^{k}_{n\to n-\beta}$ is the convex solution. In other words, $mv^{k}_{n\to n-\beta}$ is the motion vector within the partition that is closest to the average motion vector.

In this thesis, another simpler solution is proposed: using the mode motion vector, i.e., the motion vector that covers the largest area in that partition [5]. The rationale is that this is the optimal motion vector for the most part of the block. If there exists more than one mode for a given partition, then the convex solution is used.

4.3.3. Motion Vector Scaling

Since this method has a very low complexity, and it is capable of producing an approximation for the current block if the incoming bitstream has a motion vector for that block, it is considered in the transcoder. However, the implementation in this work differs slightly from the implementation in Sec. 4.2.1. When it needs to group the motion vectors for a block, it uses a similar mode algorithm as the one described in Sec. 4.3.2. The only difference is that, for MV Scaling, if there is more than one mode, than all mode motion vectors are scaled. Thus, this method may produce more than one candidate motion vector for a given block.

4.4. Other Techniques Used

In addition to the techniques discussed in the previous chapters, others techniques may also be used in order to approximate motion vectors in the transcoder. Some of these techniques, such as the spatial prediction techniques, do not use the motion vectors found in the incoming bitstream, instead using the motion vectors from spatial neighbours of the current partition. These neighbouring partitions were already encoded in the transcoder, and these techniques behave similarly to techniques used in fast motion estimation methods that are usually found in video encoders, such as the EPZS [132].

The other technique used in the transcoder is the inversion technique, that is particularly useful in order to generate both forward and backward motion vectors.

4.4.1. Spatial Prediction

There are many spatial prediction techniques in the literature [132]. In this thesis, two popular methods are considered: the median motion vector and a weighted average. In fact, these are simple methods, that are common in many video encoder implementations, not only on transcoding. They have very low computational complexity, and they do not use motion vectors from the incoming bitstream - instead, they use the set of motion vectors that were already computed in the transcoding loop. As such, they can only use the motion vectors for the blocks in the causal region of coding, which is usually all blocks above and to the left of the current block.

The first method is the median motion vector. This method is used in the H.264/AVC to generate the predicted motion vector [106], and it is also used in some popular fast motion estimation algorithms, such as the EPZS [132]. To compute the median motion vector, three neighbours of the current block are considered: the "block to the left", "block to the top" and "block to the top-right" (namely, A, B and C in Fig. 4.10). If block C is not available, a "block to the top-left" (namely D in the figure) is used instead. Note that, despite the size of the neighbouring partitions, only the motion vector at these positions are used to compute the median (for example, only the motion vector for the B neighbour is used for the "top" block, the remaining motion vectors are ignored). If any of the others are not available, the algorithm uses only the remaining blocks. The motion vector generated by this method is the median between the three blocks.

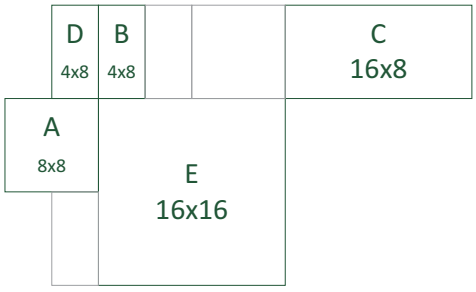


Figure 4.10.: Example of the neighbouring partitions used for computing the Median approximation.

The second method is a weighted average of the motion vectors belonging to blocks above and on the left of the current block. The equation used to generate the candidate is:

$$mv^{k}_{n\to n-\alpha} = \frac{\sum_{i\in\Gamma} w_i \cdot mv^{i}_{n\to n-\alpha}}{\sum_{i\in\Gamma} w_i} \tag{4.12}$$

where Γ denotes the set of partitions directly above and to the left of the current partition, and w_i is the weight, which is equal to the number of pixels in the current partition B^k_n that are neighbouring the partition B^i_n. This second method is exactly the method that the W-SVC codec uses to predict the motion vectors. Note that, differently from the median motion vector, all motion vectors in the neighbouring partition are considered. An example of this method is shown in Fig. 4.11, and the approximated motion vector for this example is:

$$\begin{aligned} mv^{k}_{n\to n-\alpha} =& \frac{1}{4}\cdot mv^{A}_{n\to n-\alpha} + \frac{1}{4}\cdot mv^{B}_{n\to n-\alpha} \\ &+\frac{1}{8}\cdot mv^{C}_{n\to n-\alpha} + \frac{1}{8}\cdot mv^{D}_{n\to n-\alpha} + \frac{1}{4}\cdot mv^{E}_{n\to n-\alpha} \end{aligned} \tag{4.13}$$

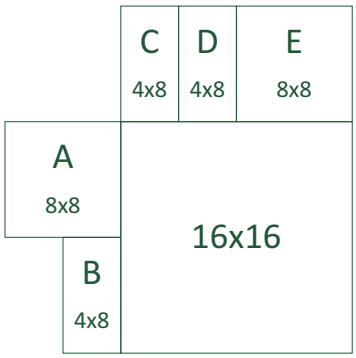


Figure 4.11.: Example of the neighbouring partitions used for computing the Spatial approximation. In this example, the final motion vector for the current partition k is given by Eq. 4.13.

4.4.2. Motion Vector Inversion

This method is used to approximate backward motion vectors from forward motion vector candidates. It achieves this by just inverting the forward motion vector. The equation for inversion is:

$$mv^k_{n\to n+\alpha} = (-1) \cdot mv^k_{n\to n-\alpha} \tag{4.14}$$

4.5. Experimental Results

This section analyses the motion vector approximation techniques seen on the previous sections. While the goal of these techniques is to enable the transcoder to achieve a better rate distortion performance at a lower complexity cost, this is dependent on the type of transcoder they are applied to. Thus, two experiments were devised to evaluate these techniques. The first experiment attempts to evaluate these techniques in an unbiased manner, independently from the transcoder target codec, analysing the techniques in terms of reliability and accuracy. The second experiment analyses these techniques within the H.264/AVC transcoder.

4.5.1. Reliability and Accuracy Analysis

The first experiment was implemented in the W-SVC transcoder, operating on motion vectors generated by the H.264/AVC codec. Here, and throughout this thesis, the H.264/AVC implementation used is the JM 14.2 [64], while the W-SVC software used is the version 13.6 [124]. First, all partitions in the W-SVC codec are tested (64×64, 32×32, 16×16, 8×8 and 4×4). Then, when testing each partition, the encoding loop was modified to perform full motion estimation and apply each MV Approximation technique for that partition in that particular initial condition. This way, since both actions are being performed in the same loop, the motion vectors generated by the MV Approximation techniques can be compared to those obtained by the full motion estimation algorithm. It is very important for the full motion estimation to be performed inside the transcoder loop, rather than in a different execution, so that it uses the same initial values as the MV approximation techniques (such as the same predicted motion

vectors, the same reference frames, etc..). Which motion vectors and modes are actually used to encode the sequence are not particularly important for the rest of the experiment. In the experiments, the first 100 frames of each sequence in 4CIF resolution were used, and the H.264/AVC bitstreams were encoded using $QP = 20$. A brief description of the sequences used can be found in Appendix C. The techniques considered, and its parameters, are described next.

(i) H.264/AVC MVs: These are the incoming H.264/AVC motion vectors. They can only be reused if their reference frames matches.

(ii) Spatial: This is the weighted average of MVs of the neighbouring blocks, as seen in Sec. 4.4.1.

(iii) Median: This is the median of the MVs of the neighbouring blocks, as seen in Sec. 4.4.1.

(iv) MV Scaling - average: This is the MV Scaling algorithm (Sec. 4.2.1) using a weighted average as the grouping algorithm (Sec 4.3.2).

(v) MV Scaling - mode: This is the MV Scaling algorithm (Sec. 4.2.1) using mode as the grouping algorithm (Sec. 4.3.3).

(vi) MV Composition - FDVS: This is the popular FDVS algorithm (Sec. 4.2.2) using the weighted average as the grouping algorithm (Sec. 4.3.2).

(vii) MV Composition - TVC: This is the popular TVC algorithm (Sec. 4.2.2) using the weighted average as the grouping algorithm (Sec. 4.3.2).

(viii) Proposed MV Composition - average, alignment at the same size as the partition: This is the proposed MV Composition algorithm (Sec. 4.3.1) using the weighted average as the grouping algorithm (Sec. 4.3.2), and aligning the partition at the same size at each step.

(ix) Proposed MV Composition - average: This is the proposed MV Composition algorithm (Sec. 4.3.1) using the weighted average as the grouping algorithm (Sec. 4.3.2), but aligning the partition with the 4×4 grid.

(x) Proposed MV Composition - mode: This is the proposed MV Composition algorithm (Sec. 4.3.1) using the mode as the grouping algorithm (Sec. 4.3.2), and aligning the partition with the 4×4 grid.

Therefore, the MV Approximation techniques can be grouped in four classes: the reuse of H.264/AVC MVs (method (i)); spatial MV approximations, that use motion vectors from the neighbouring blocks rather than incoming motion vectors from the H.264/AVC bitstream (methods (ii) and (iii)); MV Scaling algorithms, that use motion vectors from the incoming bitstream (methods (iv) and (v)); and MV Composition algorithms, that also use motion vectors from the incoming bitstream (methods (vi) through (x)). For all cases of MV Composition, no motion vectors are assumed for intra frames - if an intra frame is found, the composition stops (or pops the next motion vector in the composition list, in the case of the proposed techniques). For the remainder of this section, the methods will be referred to with the numbers on this list.

In order to evaluate the MV approximation techniques, two criteria are used: reliability and accuracy. The first criterion refers to the frequency that a particular MV Approximation technique is able to produce a motion vector candidate. Some of the techniques used, such as the Median or Spatial approximation, can always produce a candidate. The others, such as those based on MV Scaling or MV Composition, may fail to produce a candidate in certain situations. Also, note that the reuse of the incoming motion vectors is always the first choice - the other techniques are used only if the incoming motion vectors cannot be directly reused. Here, the reliability rate is defined as the number of times a particular technique was able to produce a candidate divided by the number of times the technique was required to do so.

The accuracy is analysed by comparing the distance of the approximated MV to the best motion vector, given by full motion estimation, which is used as the ground truth. To better visualize the results, this distance is separated in three classes: *equal*, *close* and *far*. The first class is self defined - approximated motion vectors that are equal to the best motion vector, at quarter-pixel level, fall in this category. The separation between the other two categories, *close* and *far*, is made using a threshold T_{dMV}. This threshold is selected as $T_{dMV} = 9$, in quarter-pixel scale. The threshold is chosen because it is common that a refinement of the type of Hexagon search (see Appendix A for details), or full search with a search window of 2 [100, 117], are used on top of the approximated motion vector, and this threshold would mean that it is possible that the best motion vector can be found in the first step of the search (in the case of the Hexagon search).

Therefore, by using these two metrics, it is possible to compare if a particular technique is robust, in the sense that it is able to generate a motion vector candidate in different scenarios, and accurate, in the sense that the motion vectors produced are actually close to the best motion vector.

Table 4.1.: Reliability of MV Approximation Techniques for the *forward* direction, shown as a percentage.

		IPP1				*IBBP*			
	Technique	City	Crew	Soccer	Average	City	Crew	Soccer	Average
MB 16×16	(i)	53	34	52	**46**	28	19	27	**25**
	(ii),(iii)	100	100	100	**100**	100	100	100	**100**
	(iv),(v)	95	45	91	**77**	96	61	87	**81**
	(vi)	93	26	86	**68**	7	3	6	**5**
	(vii)	93	27	87	**69**	7	3	6	**5**
	(viii)	93	26	86	**68**	91	38	76	**68**
	(ix),(x)	94	29	89	**71**	94	41	80	**72**
MB 32×32	(i)	54	44	54	**51**	28	22	28	**26**
	(ii),(iii)	100	100	100	**100**	100	100	100	**100**
	(iv),(v)	100	69	99	**89**	100	80	97	**92**
	(vi)	99	53	98	**83**	7	5	6	**6**
	(vii)	99	53	98	**83**	7	5	6	**6**
	(viii)	99	53	98	**83**	99	64	95	**86**
	(ix),(x)	99	54	99	**84**	100	66	96	**87**
MB 64×64	(i)	54	49	54	**52**	28	25	28	**27**
	(ii),(iii)	100	100	100	**100**	100	100	100	**100**
	(iv),(v)	100	82	100	**94**	100	92	100	**97**
	(vi)	100	72	100	**91**	6	6	6	**6**
	(vii)	100	71	100	**90**	6	6	6	**6**
	(viii)	100	72	100	**91**	100	85	99	**95**
	(ix),(x)	100	72	100	**91**	100	86	100	**95**

Analysing the Reliability

The results to evaluate the reliability are shown in Tables 4.1 and 4.2, for the forward and backward directions, respectively.

First, it can be seen from the tables that, on average for the *IPP1* configuration, only 46% (for a MB Size of 16×16) to 52% (for a MB size of 64×64) of the H.264/AVC motion vectors can be directly reused. Naturally, no motion vector can be reused for the backward direction. For the *IBBP* configuration, around 25% of the motion vectors can be reused, on average for all sequences, directions and block sizes. Therefore, the MV Approximation methods are highly needed, for both forward and backward directions, regardless of the configuration used. It can also be seen that the spatial approximation

Table 4.2.: Reliability of MV Approximation Techniques for the *backward* direction, shown as a percentage.

		IPP1				*IBBP*			
	Technique	City	Crew	Soccer	Average	City	Crew	Soccer	Average
MB 16×16	(i)	0	0	0	**0**	26	20	25	**24**
	(ii),(iii)	100	100	100	**100**	100	100	100	**100**
	(iv),(v)	0	0	0	**0**	54	39	52	**48**
	(vi)	0	0	0	**0**	0	0	0	**0**
	(vii)	0	0	0	**0**	0	0	0	**0**
	(viii)	96	53	94	**81**	94	53	85	**77**
	(ix),(x)	96	53	94	**81**	95	54	87	**79**
MB 32×32	(i)	0	0	0	**0**	26	22	26	**25**
	(ii),(iii)	100	100	100	**100**	100	100	100	**100**
	(iv),(v)	0	0	0	**0**	56	46	56	**53**
	(vi)	0	0	0	**0**	0	0	0	**0**
	(vii)	0	0	0	**0**	0	0	0	**0**
	(viii)	100	74	99	**91**	99	72	98	**90**
	(ix),(x)	100	74	99	**91**	100	73	98	**90**
MB 64×64	(i)	0	0	0	**0**	26	24	26	**25**
	(ii),(iii)	100	100	100	**100**	100	100	100	**100**
	(iv),(v)	0	0	0	**0**	56	51	56	**54**
	(vi)	0	0	0	**0**	0	0	0	**0**
	(vii)	0	0	0	**0**	0	0	0	**0**
	(viii)	100	86	100	**95**	100	87	100	**96**
	(ix),(x)	100	86	100	**95**	100	87	100	**96**

methods ((ii) and (iii)) can always produce a candidate, as expected from the definition of these methods.

The reliability for the MV Scaling based methods is also high for all cases, except for the backward direction using *IPP1* configuration. The reason is that this method needs a H.264/AVC motion vector in order to produce a candidate, and, for this configuration, there are no backward motion vectors.

The reliability for all MV Composition based methods is very similar for the forward direction using *IPP1* configuration. This is expected, as all these methods perform similar operations for this case. However, for the forward motion vector using *IBBP* configuration, the methods (vi) and (vii) shows a reliability rate of less than 7% (in the best case, for City sequence using a MB size of 16×16), and for both forward and backward directions for *IPP1* and *IBBP* configurations they are unable to produce any motion vector. At the same time, the proposed method (x) show a reliability rate in the range of 71% (on average for all sequences, for the *IPP1* configuration using a MB size of 16×16 for the forward direction) to 96% (on average for all sequences, for the *IBBP* configuration using a MB size of 64×64 for the backward direction).

For all methods that are based on the H.264/AVC motion vectors, it is important to notice that the performance for Crew sequence is substantially lower than for City and Soccer sequences. The reason is that the bitstream for Crew sequence contains a significant higher number of intra macroblocks (31% and 21%, for *IPP1* and *IBBP* configurations, respectively), compared to the other sequences (ranging from 1.9% to 4.7%), which naturally impacts the techniques that rely on approximating the incoming motion vectors.

Analysing the Accuracy

The accuracy results for all sequences tested are shown in Figs. 4.12 and 4.13, also for *IPP1* and *IBBP* configuration, respectively. In these figures, the reliability can also be seen as the height of each particular bar.

The spatial approximation methods ((ii) and (iii)) both proved very accurate, with over 80% of the motion vectors being close to the best motion vector (85% for method (ii) and 84% for method (iii), on average for all sequences, MB sizes, coding configuration and directions). However, it is important to notice that these methods operate on motion vectors already refined for the W-SVC codec, located on the neighbouring blocks, not

on the H.264/AVC motion vectors. Furthermore, the method (ii), in particular, is used in the W-SVC as the predicted motion vector, meaning that the rate R in the equation $J = D + \lambda R$, used to calculate the cost (as seen in Sec. 3.2.8), is computed using the method (ii) as prediction.

As for the reuse of the H.264/AVC motion vectors (method (i)), the results show that, while only a small percentage of the incoming motion vectors can be directly reused, 95% of the reused motion vectors are close to the best motion vector (on average for all sequences, MB sizes, coding configuration and directions). At the same time, while the MV Scaling based algorithms (methods (iv) and (v)) show a high reliability rate, these methods are not so accurate. However, even though these methods present the lowest accuracy among all methods tested, using the mode as the grouping algorithm (method (v)) provides slightly better overall results, with 72% of the candidates produced by method (iv) being close to the best motion vector, and 75% of the motion vectors for method (v) (again, on average for all sequences, MB sizes, coding configuration and directions).

Among the MV Composition based methods, although methods (vi) and (vii) fail to produce candidates in many situations, showing a poor reliability rate, the motion vectors produced by these methods are very accurate, with 80% of the motion vectors produced by each method being close to the best motion vector. Analysing the methods (viii), (ix) and (x) (the proposed methods), it can be seen that aligning the partition at the 4×4 grid or at the same partition size provides a very small gain for the former, but using the mode as the grouping method (method (x)) improves the performance significantly, especially for larger block sizes. On average for all sequences, MB sizes, coding configuration and directions, 80% of the motion vectors produced by methods (viii) and (ix) and 86% of the candidates produced by method (x) are close to the best motion vector.

Therefore, among the methods presented to approximate H.264/AVC motion vectors, it can be verified that methods based on MV Composition produce more accurate results than methods based on MV Scaling, on average. Among the methods based on MV Composition, the proposed method (x) is both more reliable, being able to produce MV candidates on different scenarios, and more accurate, being able to produce candidates that are closer to the best motion vector.

Table 4.3.: Average PSNR Loss using different MV Approximation methods within the H.264/AVC to W-SVC transcoder. The RT-FS, using the appropriate MB size, is used as anchor to compare the PSNR of the different methods.

			MV Approximation Methods									
	MB	Sequence	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)
IPP1 Configuration	16 × 16	City	−2.84	−0.54	−0.31	−1.44	−1.29	−0.72	−0.78	−0.32	−0.34	**−0.14**
		Crew	−0.53	−0.31	−0.17	−0.24	−0.19	−0.28	−0.30	−0.17	−0.15	**−0.10**
		Soccer	−1.58	−0.94	−0.53	−0.09	+0.00	−0.35	−0.41	+0.18	+0.21	**+0.27**
	32 × 32	City	−1.74	−0.17	−0.12	−0.86	−0.70	−0.49	−0.52	−0.20	−0.21	**−0.06**
		Crew	−0.34	−0.19	−0.10	−0.17	−0.11	−0.20	−0.21	−0.12	−0.11	**−0.06**
		Soccer	−1.33	−0.62	−0.32	−0.11	−0.04	−0.34	−0.37	+0.09	+0.10	**+0.14**
	64 × 64	City	−1.17	−0.07	−0.02	−0.62	−0.47	−0.35	−0.38	−0.12	−0.12	**+0.00**
		Crew	−0.24	−0.11	−0.06	−0.11	−0.06	−0.13	−0.14	−0.06	−0.06	**−0.02**
		Soccer	−1.03	−0.38	−0.18	−0.03	+0.03	−0.26	−0.28	+0.13	+0.14	**+0.18**
IBBP Configuration	16 × 16	City	−2.75	−0.52	−0.30	−0.93	−0.91	−2.26	−2.26	−0.32	−0.27	**−0.19**
		Crew	−0.49	−0.25	−0.13	−0.10	**−0.09**	−0.43	−0.43	−0.22	−0.18	−0.17
		Soccer	−1.44	−0.80	−0.45	+0.07	**+0.09**	−1.26	−1.27	−0.21	−0.14	−0.11
	32 × 32	City	−1.70	−0.18	−0.10	−0.52	−0.51	−1.35	−1.35	−0.13	−0.13	**−0.06**
		Crew	−0.33	−0.17	−0.09	**−0.07**	**−0.07**	−0.29	−0.30	−0.11	−0.10	−0.08
		Soccer	−1.22	−0.54	−0.32	−0.01	**+0.00**	−1.05	−1.06	−0.09	−0.07	−0.04
	64 × 64	City	−1.15	−0.06	−0.02	−0.38	−0.37	−0.92	−0.92	−0.07	−0.06	**−0.01**
		Crew	−0.23	−0.10	−0.06	**−0.04**	**−0.04**	−0.20	−0.21	−0.06	−0.05	**−0.04**
		Soccer	−0.95	−0.35	−0.17	**+0.06**	**+0.06**	−0.80	−0.81	+0.01	+0.02	+0.04
Average		MB 16 × 16	−1.60	−0.56	−0.31	−0.45	−0.40	−0.88	−0.91	−0.17	−0.15	**−0.07**
		MB 32 × 32	−1.11	−0.31	−0.17	−0.29	−0.24	−0.62	−0.63	−0.09	−0.09	**−0.03**
		MB 64 × 64	−0.80	−0.18	−0.08	−0.19	−0.14	−0.44	−0.46	−0.03	−0.02	**+0.03**

4.5.2. Evaluation within the H.264/AVC to W-SVC transcoder

In this experiment, the motion vector approximation methods are applied within a H.264/AVC to the W-SVC transcoder. The complete details of this transcoder are given in Chapter 5, however, to evaluate this experiment, it is sufficient to know that all other transcoding parameters are kept unchanged, only the MV Approximation method is varied. In this experiment, the full length of the sequences is used, and all sequences are in 4CIF resolution. For all cases, the MV Refinement is turned on (using the default Hexagon search), thus, the experiment shows the impact that the start of the search has on the final outcome of the Hexagon search algorithm. Also, the H.264/AVC motion vectors are reused, and the MV Approximation methods are only used when the H.264/AVC motion vectors cannot be directly reused.

In the experiment, first the original sequence is encoded with the H.264/AVC, using $QP = 26$ and a given coding configuration (two coding configurations are used in this experiment: *IPP1* and *IBBP*). The output bitstream is then transcoded to the W-

SVC using the H.264/AVC to W-SVC transcoder, generating a scalable bitstream. This scalable bitstream is then extracted and decoded at a specific bitrate i (with $i \in B$, and $B = \{1280, 1536, 1792, 2048, 2304, 2688, 3072\}$ kbps) and, for each bitrate, the PSNR P_i^m (where i denotes the bitrate and m denotes the transcoding method) is computed (the PSNR is always computed using the original, uncompressed, sequence). In addition to P_i^m, the PSNR P_i^{FS} for the trivial transcoder (using full motion estimation and a search window of 32, RT-FS) is also computed, and it is used for comparison. The average PSNR loss (see Appendix D for more details on the PSNR and the Average PSNR Loss) for the method m is then defined as:

$$APL\,(m) = \frac{1}{N_B} \cdot \sum_{i \in B} \left(P_i^m - P_i^{FS} \right) \tag{4.15}$$

where $APL\,(m)$ is the average PSNR loss for the method m, and N_B is the number of elements in B (in this case, $N_B = 7$). The remaining settings are the same for both the transcoder and the anchor RT-FS (i.e., the coding configuration and the MB size). The results when using the MV Approximation methods (i) to (x) are shown in Table 4.3. Note that, by definition, $APL\,(FS) = 0$, and therefore the results for RT-FS are omitted from the table. Finally, note also that the W-SVC codec produces bitstreams of constant bitrate and, for this reason, this sections analyses the results using just the average PSNR loss.

It can be seen in Table 4.3 that using MV Approximation techniques is crucial for the transcoder, as only reusing the H.264/AVC motion vectors, and using the null motion vector where these motion vectors cannot be reused (method (i)), yields to a large loss, especially for City and Soccer sequences (up to -2.84 dB for City *IPP1* 16×16). The use of the spatial approximation methods (methods (ii), spatial, and (iii), median) yields much better results, but there is still a large loss for Soccer sequence (up to -0.94 dB for method (ii) and -0.53 dB for method (iii), for Soccer *IPP1* 16×16).

The MV Scaling based algorithms (methods (iv) and (v)) show an erratic behaviour, yielding large losses for some sequences (up to -1.44 dB for method (iv) and -1.29 dB for method (v), for City *IPP1* 16×16 sequence), but showing some gains for other sequences (up to $+0.07$ dB for method (iv) and $+0.09$ dB for method (v), for Soccer *IBBP* 16×16). For some sequences, the MV Scaling algorithm is the best out of all MV Approximation techniques (as shown highlighted in the table). Also, using the

mode as the grouping algorithm (method (v)) is always better than using a weighted average (method (iv)), by up to +0.16 dB, for City *IPP1* 32×32. Note that there are a number of reasons why the proposed transcoder may outperform the full search technique: (i) the full search is only carried out at integer pixel level, not at sub-pixel level - therefore, a better MV may be found if the result of the integer-pixel search is different; (ii) since the transcoder uses fast motion estimation, a search window of 60 is used, instead of the search window of 32 used by the full search algorithm; and (iii) the proposed transcoder reuses the H.264/AVC partitions, and may test different partitions from the trivial transcoder.

The traditional MV Composition algorithms, FDVS (method (vi)) and TVC (method (vii)), show a much better performance for *IPP1* configuration than for *IBBP* configuration, as it would be expected, following the discussion on the reliability and accuracy from the previous section. Also, since they are not able to produce backward MV candidates, the performance is not so good for any of the sequences tested.

The proposed MV Composition algorithms (methods (viii), (ix) and (x)) yields the best performance for most of the sequences, and the best overall performance (as shown highlighted in the table). Changing the alignment method yields a small difference (methods (viii) and (ix)), but using the mode as the grouping algorithm (method (x)) provides a larger gain (up to +0.18 dB, for City *IPP1* 16×16), and the gain is consistent, as method (x) always outperforms the methods (viii) and (ix), for all cases tested.

4.5.3. Subjective Evaluation of the MV Composition Algorithms

In order to evaluate the proposed MV Composition algorithm, a double stimulus subjective test was conducted [65]. In this test, the subjects are exposed to two versions of the video: one of them (in random order) is always the original, uncompressed video, and the other is the video to be tested. After seeing these two videos, the user is asked to rate both videos, from 0 (worst quality) to 100 (best quality). Each of these ratings is called an *opinion score*. The test was conducted with 14 subjects, and the average score of all users, called *mean opinion score* (MOS) is computed. The videos used in the test are: the original sequence; the H.264/AVC to the W-SVC transcoder (which is fully explained in Chapter 5) using only the method (vi) as MV Approximation (FDVS); the H.264/AVC to W-SVC transcoder using only method (vii) (TVC); and the H.264/AVC

Table 4.4.: Comparison among MV Composition methods within the H.264/AVC to W-SVC transcoder. In all cases, the MB size used is 16×16 and the bitrate is 1280 kbps.

		IPP1 Configuration			*IBBP* Configuration		
	Method	PSNR	MSSIM	MOS	PSNR	MSSIM	MOS
City 4CIF	Original	∞	1.00	77	∞	1.00	75
	(vi)	32.6	0.89	49	30.6	0.83	33
	(vii)	32.5	0.89	44	30.6	0.83	32
	(x)	**33.4**	**0.91**	**51**	**33.4**	**0.91**	**53**
Crew 4CIF	Original	∞	1.00	68	∞	1.00	73
	(vi)	33.2	0.85	25	33.0	0.85	24
	(vii)	33.1	0.85	27	33.0	0.85	25
	(x)	**33.4**	**0.86**	**32**	**33.4**	**0.86**	**32**
Soccer 4CIF	Original	∞	1.00	79	∞	1.00	78
	(vi)	31.6	0.83	36	30.7	0.79	28
	(vii)	31.5	0.82	29	30.7	0.79	28
	(x)	**32.2**	**0.85**	**40**	**31.9**	**0.83**	**39**

to W-SVC transcoder using only method (x) (the proposed method). The videos are decoded at 1280 kbps, with 4CIF resolution and full frame rate (60 frames per second). The MOS results, along with the PSNR and the Mean Structure Similarity index (MSSIM) [136], are shown in Table 4.4. More details on how to compute the MSSIM metric and on how the subjective experiments are carried out are given in Appendix D.

It can be seen from the table that the proposed method (x) always yields a better PSNR than the other methods (vi) and (vii). However, when the difference in PSNR between the methods (vi), (vii) and (x) is higher, it is reflected in the results of both the MSSIM and the MOS. An example is the City 4CIF using *IBBP* configuration, where the different in PSNR between methods (x) and the others is about +2.8 dB, and the MSSIM difference is +0.08 and the MOS difference is +20. On the other hand, when the difference in PSNR is lower, its impact cannot be seen in the MSSIM and the MOS. An example is the Crew 4CIF using *IPP1* configuration, where the difference in PSNR is +0.3 dB, and the MSSIM difference is +0.01 and the difference in MOS is +5. It can also be seen that the MOS for the proposed method (x) is always better than, or the same as, the other methods, for all sequences and coding configurations tested. Finally, the difference in MOS is more significant for the *IBBP* coding configuration, as expected, since both methods (vi) and (vii) are not able to reliably produce MV candidates for this case.

4.6. Conclusions

This chapter discusses several motion vector approximation techniques, in particular those based on MV Scaling and MV Composition. The state-of-the-art on these techniques is presented, followed by a discussion of strengths and weaknesses of these techniques. Of special interest, it is noted that the state-of-the-art algorithms based on MV Composition are not able to cope with flexible coding configurations that may be found in H.264/AVC bitstreams.

Then, a novel MV Composition technique is proposed. This technique offers a substantial improvement on existing state of the art techniques, with the unique feature of being able to work with different coding configurations in the source codec, with multiple reference frames and variable block sizes, producing accurate forward and backward motion vectors. Experiments have shown that the proposed method is able to produce accurate motion vector candidates in a wider variety of scenarios, and that using this method within a H.264/AVC to W-SVC transcoder produces the best overall results. Also, subjetive tests concluded that the users can perceive the difference between the MV Composition methods, consistently rating the proposed method better than the existing techniques, especially for more complex coding structures such as *IBBP*.

Finally, since MV Approximation is a crucial module for a transcoder, as evidenced by the performance of not using MV approximation methods at all given in Table 4.3, it is suggested to use a collection of these methods, generating several candidates in order to improve the transcoder overall performance. By using different methods, the strengths of each method can be combined, while minimising the particular weaknesses of the methods. As an example, when one method fails to produce a MV candidate, or produces an inaccurate output, the transcoder can rely on other methods to produce a better approximation. The next chapter presents transcoder from H264/AVC bitstreams to the W-SVC codec, using the MV Approximation methods given in this chapter.

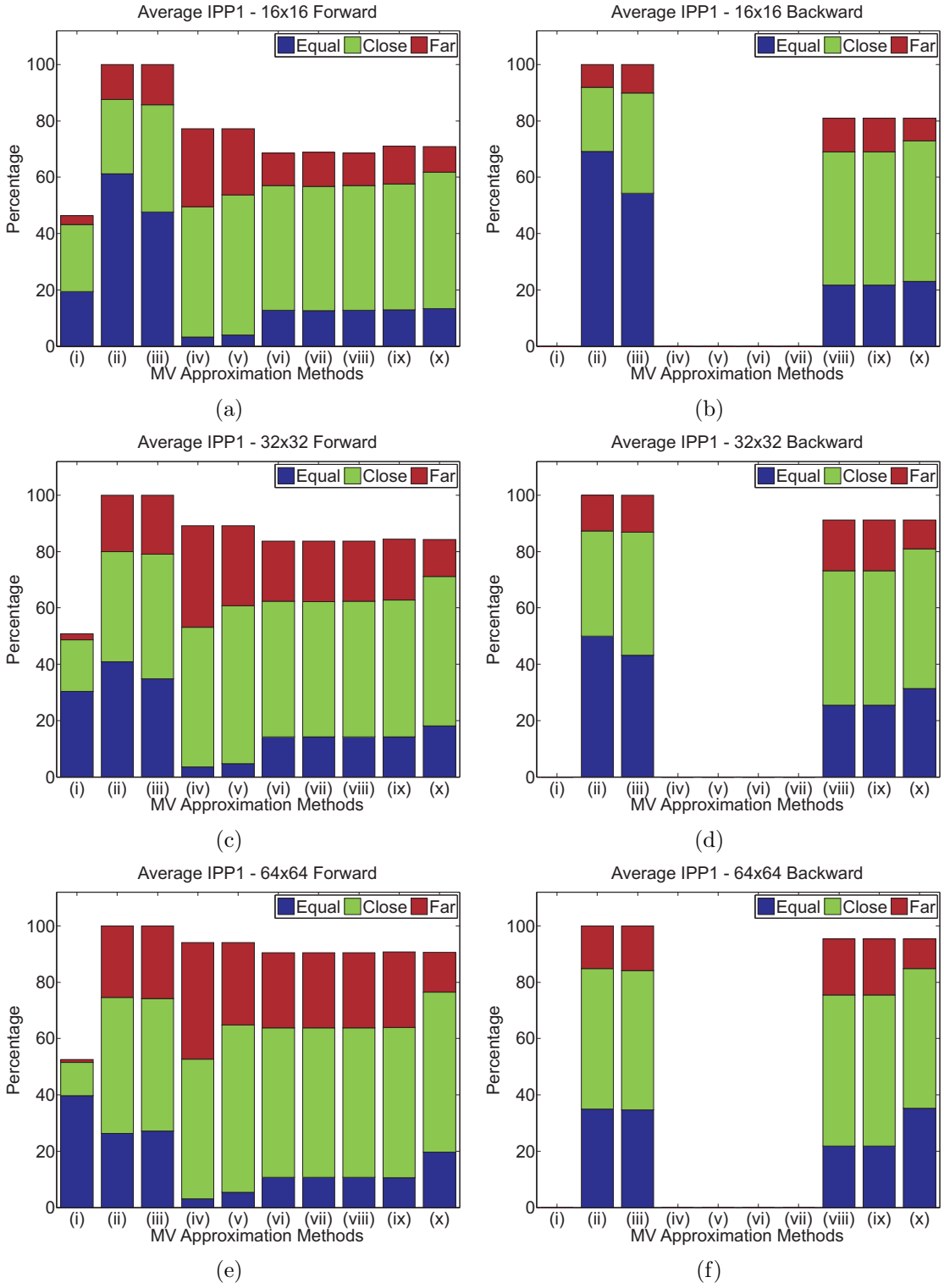


Figure 4.12.: MV Approximation accuracy for *IPP1* configuration, for: (a) 16×16; (c) 32×32 and (e) 64×64, for the forward direction; and (b) 16×16; (d) 32×32; and (f) 64×64, for the backward direction.

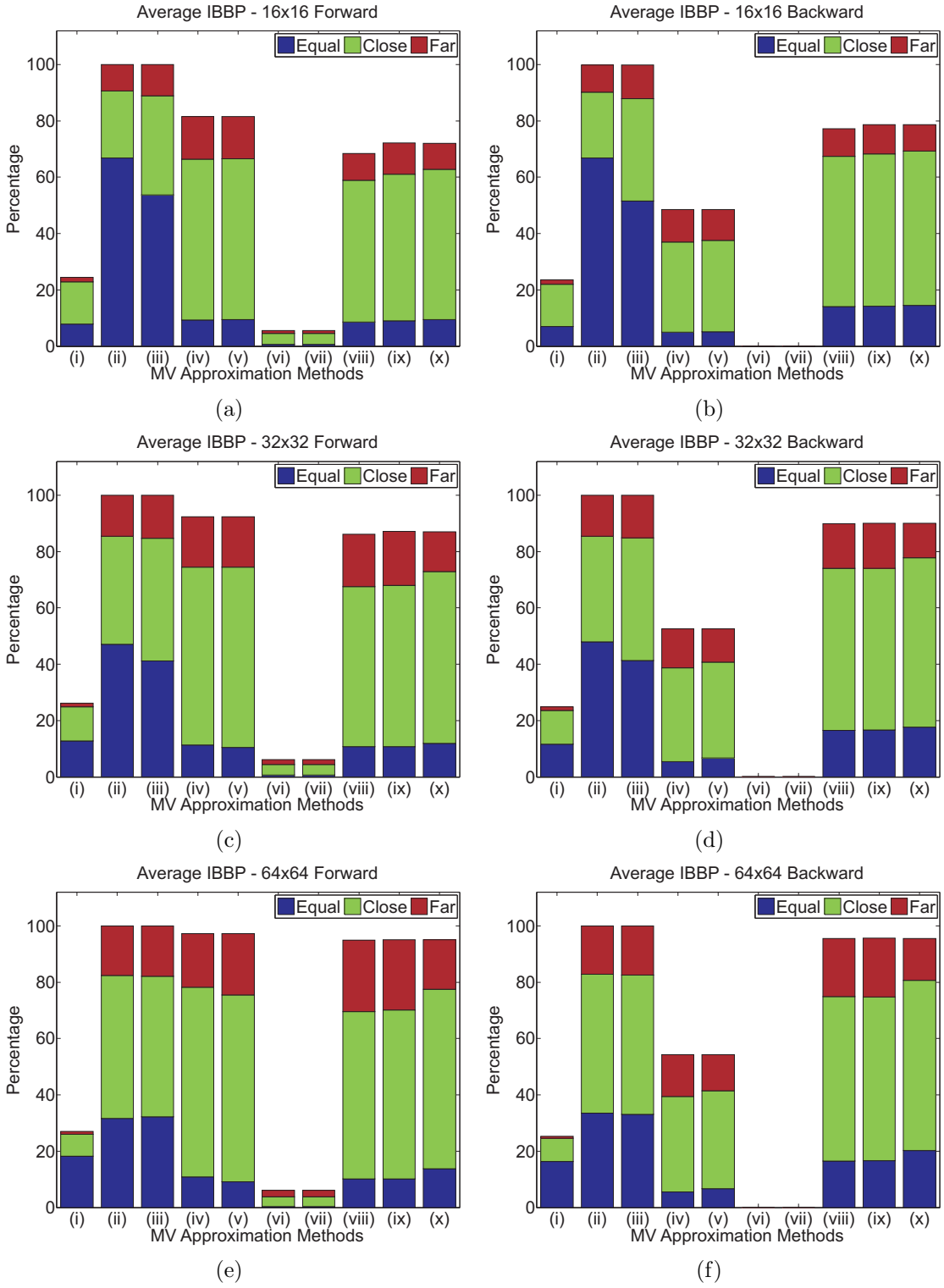


Figure 4.13.: MV Approximation accuracy for *IBBP* configuration, for: (a) 16×16; (c) 32×32 and (e) 64×64, for the forward direction; and (b) 16×16; (d) 32×32; and (f) 64×64, for the backward direction.

Chapter 5.

Transcoding from H.264/AVC to W-SVC

This chapter presents the proposed transcoder from the H.264/AVC [62] to the W-SVC codec [124]. Such a transcoder can potentially enlarge the range of applications for the W-SVC codec, and some of the techniques developed here can be applied in other types of transcoding, such as from H.263 [61] or MPEG-2 [59], to W-SVC, or even between DPCM/DCT codecs.

Since the two codecs are fundamentally different, the transcoder architecture follows the approach of an heterogeneous transcoder [9, 135], seen in Sec. 2.2.2. It consists in decoding the original signal, performing an intermediate processing and re-encoding the signal using the W-SVC codec. In this case, the idea is to decode the H.264/AVC sequence, extract the motion information and re-encode the signal with the W-SVC codec using the provided motion information. This way, complexity is largely reduced, since motion estimation (ME) is the most time consuming task in a video encoder [106].

5.1. Transcoder Issues

The two main issues that need to be addressed by the transcoder are the reference frame mismatch and the optimization issue. As it is common in heterogeneous transcoding, the transcoder tries to reuse the incoming H.264/AVC motion vectors as much as possible [21, 148]. However, the different reference frame structure in the two codecs restricts direct reusing of H.264/AVC motion information in the W-SVC codec. Only those motion vectors that point to the same reference frame in the W-SVC and H.264/AVC codecs

can be directly reused; those whose reference frames do not match have to be approximated and refined. The reference frame mismatch is a common problem in transcoding, especially from H.264/AVC to other codecs, such as MPEG-2 [114, 118] and MPEG-4 [55], and also in frame-rate reduction transcoding [71, 80, 117] or H.264/SVC targeting temporal scalability [10, 47]. The state-of-the-art techniques to tackle the reference frame mismatch, along with a new proposed technique, are presented on Chapter 4.

The second issue regards the optimisation, and it is specific to the H.264/AVC to W-SVC transcoder. As discussed on Chapter 3, the optimization on the W-SVC codec is different than that on the H.264/AVC codec. Even when the H.264/AVC motion vector can be directly reused in the W-SVC codec, it may not be optimal for this codec. The H.264/AVC motion vector was optimized given a target bit rate or distortion parameter, while the motion field for the W-SVC needs to be optimized considering a wider range of bitrates and spatio-temporal resolutions. The effect of this optimisation is shown in Sec. 3.2.8. To further illustrate this issue in the scope of the transcoder, two sequences were encoded with the H.264/AVC codec using the same hierarchical reference frame structure used in the W-SVC codec, with 3 levels. Additionally, the intra mode was disabled for inter macroblocks, so that the motion vectors could be directly reused in the W-SVC codec for all macroblocks. In this example, no further refinement or additional modes were considered. The motion information used in both codecs is the same. The results are shown in Fig. 5.1 as rate distortion plots, where the x axis shows the rate, in kbps, and the y axis shows the distortion, measured as the PSNR.

In the figure, two different results are shown: the results for the trivial transcoder with full motion estimation (referred as the reference transcoder with full search, RT-FS) and the results for the complete reuse of the H.264/AVC motion vectors (referred as RT-H264). It can be seen that, when a lower QP (i.e., a better quality) was used, a very high loss is present for low bitrates (up to 1000 kbps) when the motion vectors are fully reused (as shown in Figs. 5.1(a) and 5.1(a)). For higher bitrates, the performance for RT-H264 is the same, or even slightly better, than the performance of RT-FS. When a higher QP is used (i.e., a lower quality), there is still a considerable loss in the low bitrates (up to 800 kbps) when the motion vectors are fully reused (shown in Figs. 5.1(a) and 5.1(d)). However, this case provides a slightly better performance for higher bitrates. To better understand this problem, Table 5.1 shows the bitstream profile for these examples.

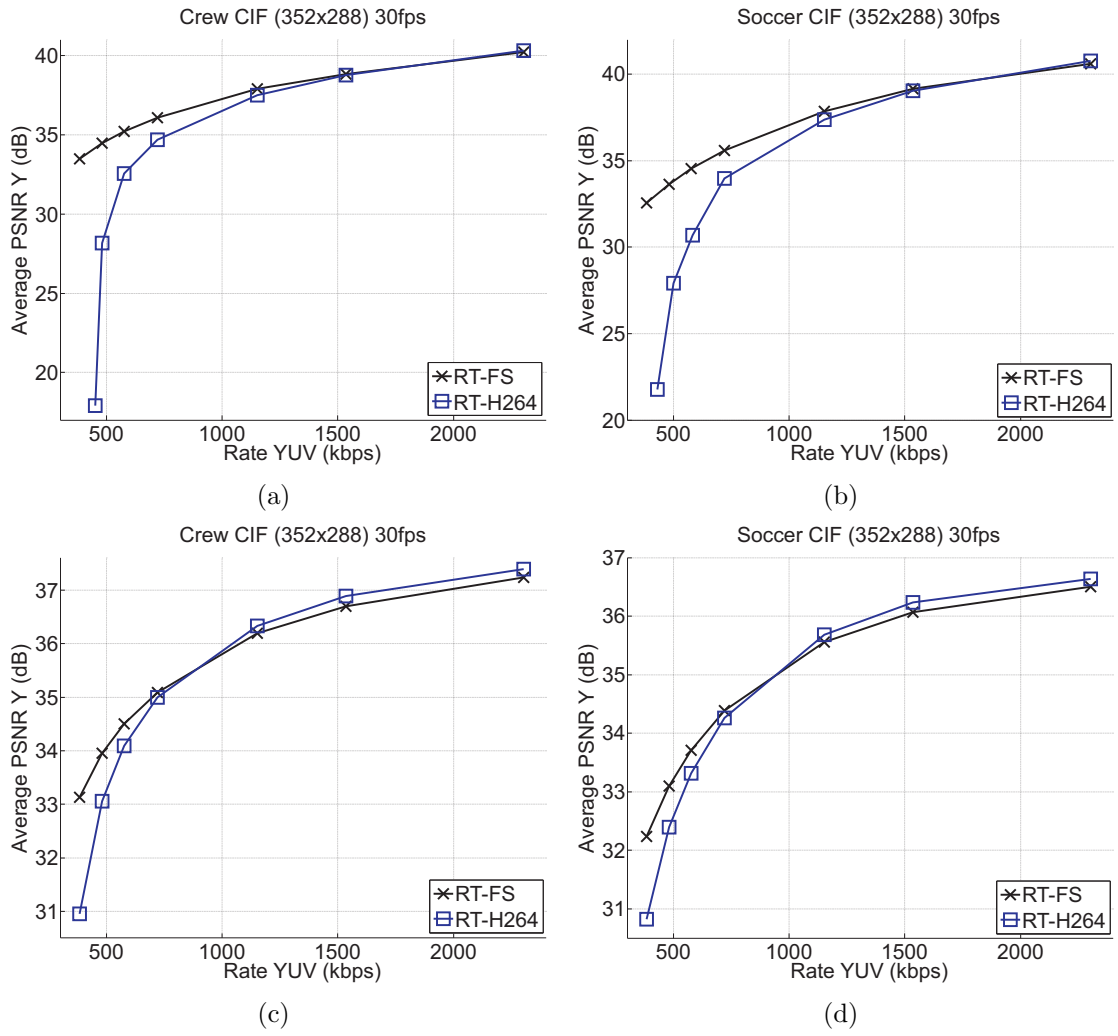


Figure 5.1.: Transcoder results for RT-FS and RT-H264, from a H.264/AVC bitstream using hierarchical motion estimation with 3 levels, for: (a) Crew using $QP = 20$; (b) Soccer using $QP = 20$; (c) Crew using $QP = 28$; and (d) Soccer using $QP = 28$.

The first thing that can be noted in the table is that the motion information (MI) bitrate is constant for all bitrates. This is because the W-SVC codec uses the same motion information for all spatio-temporal and quality resolutions. It can also be seen that, for the cases being considered, the motion information bitrate for RT-H264 is always much higher than the motion information bitrate for RT-FS. For the lower QP, the motion information bitrate is so high that the target bitrate cannot be achieved for the lower bitrates (for both Crew and Soccer sequences using QP 20). Thus, the bitrate for the texture is very low in these cases, resulting in the very high quality loss shown in Figs. 5.1(a) and 5.1(b). For the higher QP, the motion information bitrate for RT-H264 is smaller, but still higher than that for RT-FS. However, since there is still space for

Table 5.1.: Bitstream profile for RT-FS and RT-H264. Crew and Soccer sequences are used, and the H.264/AVC bitstreams used $QP = 20$ and $QP = 28$. The shaded cells indicate that the W-SVC codec could not achieve the target bitrate with that motion information. The bitrates are given in kbps.

		Crew CIF				Soccer CIF			
	Target	RT-FS		RT-H264		RT-FS		RT-H264	
	Bitrate	MI	Texture	MI	Texture	MI	Texture	MI	Texture
H.264/AVC $QP = 20$	384	162	222	**430**	**20**	156	228	**402**	**29**
	480	162	318	430	50	156	324	**402**	**99**
	576	162	414	430	146	156	420	**402**	**182**
	720	162	558	430	290	156	564	402	318
	1152	162	990	430	722	156	996	402	750
	1536	162	1374	430	1106	156	1380	402	1134
	2304	162	2142	430	2305	156	2148	402	1902
H.264/AVC $QP = 28$	384	163	221	301	83	155	229	269	115
	480	163	317	301	179	155	324	269	211
	576	163	413	301	275	155	421	269	307
	720	163	557	301	419	155	565	269	451
	1152	163	989	301	851	155	997	269	883
	1536	163	1373	301	1235	155	1381	269	1267
	2304	163	2141	301	2003	155	2149	269	2035

the texture bits in the bitstream, the PSNR results for this case, shown in Figs. 5.1(c) and 5.1(d), are much better than when a lower QP was used, but it is still considerably lower than RT-FS.

Thus, even when the H.264/AVC motion vectors can be reused, they might not be optimal for use in the W-SVC codec. To cope with these issues, an efficient framework for the transcoder was developed, which reuses the information about the H.264/AVC partitioning and motion vectors, taking into account the optimization issue. Also, efficient MV Composition algorithms (seen in Chapter 4), that are able to cope with complex motion structures, and new strategies to reduce the transcoder complexity are used.

5.2. The Proposed Transcoder

The transcoder decisions are based on the W-SVC macroblock, which can be set as $N \times N$. The proposed transcoder works with $N \in \{16, 32, 64\}$. In order to maximize the use of the H.264/AVC motion vectors, the minimum size is chosen as 4×4 for any

macroblock size. For each macroblock, the transcoder attempts to reuse the H.264/AVC motion information (motion vectors, partitioning scheme and the mode) along with other information from the H.264/AVC bitstream (such as the transform coefficients) to avoid testing partitions that are unlikely to be chosen by the W-SVC rate-distortion optimization.

Here, the testing of a partition is defined as the approximation (or reuse) and refinement of motion vectors for each direction (forward and backward) and the mode selection (forward, backward or bidirectional prediction). When testing a given partition, the transcoder attempts to directly reuse the H.264/AVC motion vectors (or, if that is not possible, use them to approximate new motion vectors using the methods seen on Chapter 4) in the W-SVC codec. This way, complexity is reduced by testing less partitions than the trivial transcoder, and also by using a better starting point in the fast motion estimation algorithm for each partition, utilised in the refinement step.

In the transcoder, the partitions are always tested in a recursive way. As an example, the path to test all partitions in a 16×16 macroblock is shown in Fig. 5.2. In this example, the first partition to be tested is the 16×16 partition, labeled $P1$ in the figure. Then, the top-left 8×8 partition is tested, labeled as $P2$. Then, the four top-left 4×4 partitions within $P2$ are tested, labeled $P3$, $P4$, $P5$ and $P6$. If the cost of the partitions $\{P2, P3, P4, P5\}$ is lower than the cost of $P2$, then this partition is split. On the other hand, if the cost of the first two partitions, $P3$ and $P4$, is already higher than the cost of $P2$, the remaining 4×4 partitions, $P5$ and $P6$, are not tested (as they would not be chosen). This approach is repeated to test and decide between the remaining 8×8 and 4×4 partitions, and then finally between the 16×16 partitions or the split. The transcoder uses this approach for other macroblock sizes, even if not all partitions are to be tested.

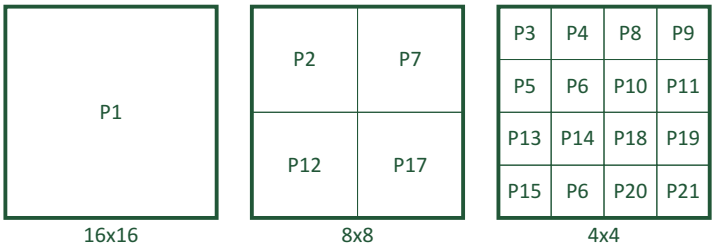


Figure 5.2.: Order of partitions tested for a 16×16 block.

The transcoder is divided in three modules, which are presented in the following sections: handling the partitions for testing, the Framework for MV Approximation and Refinement and the optional Reduced Complexity module.

5.2.1. Handling the Partitions for Testing

To decide how a 16×16 partition will be tested, the transcoder uses the H.264/AVC macroblock partition. However, as said previously, the H.264/AVC partitioning may not be optimal for the W-SVC codec, as the latter favours larger partitions if compared to the former. Thus, if the H.264/AVC macroblock was encoded in inter mode, then only those partitions of the same size or larger than the H.264/AVC partitions are tested, regardless whether the motion vectors can be directly reused. However, since the W-SVC codec uses a strict quadtree partitioning, this strategy cannot be applied straightforwardly. As an example, if the H.264/AVC macroblock was partitioned into two 16×8 blocks, then both 16×16 and 8×8 partitions will be tested in the W-SVC codec. If the macroblock was encoded in intra mode in H.264/AVC, then all partitions within that block will be tested in the W-SVC codec.

If the transcoder is using larger partitions (64×64 or 32×32), then all these partitions are tested, and the testing of smaller partitions (from 16×16 and smaller) are decided using the H.264/AVC macroblocks. Examples of how the transcoder selects which partitions are tested are shown in Fig. 5.3.

5.2.2. Framework for Motion Vector Approximation and Refinement

To efficiently use the motion information found in the H.264/AVC stream a framework for approximation and refinement for motion vectors was developed. As discussed in the previous section, the H.264/AVC partition is used to decide which partitions will be tested in the W-SVC codec. This framework specifies how these partitions are tested. The flowchart of the algorithm is shown in Fig. 5.4.

The transcoder keeps a motion vector candidate list for each direction, which starts empty. All H.264/AVC motion vectors that can be reused are added to the appropriate candidate list. Otherwise, if no motion vectors can be reused, several strategies are used

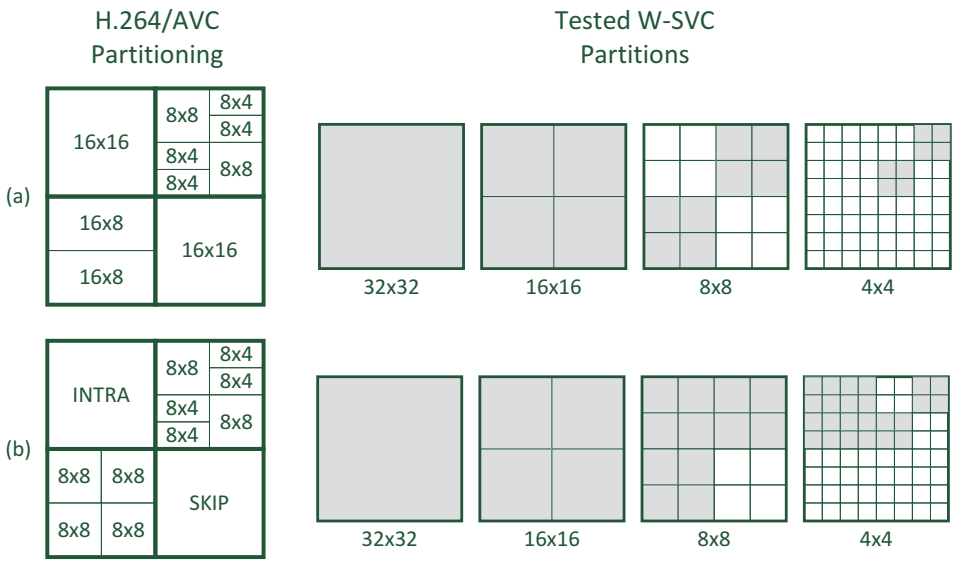


Figure 5.3.: Example of partitions tested in the H.264/AVC to W-SVC transcoder. In both examples, the W-SVC macroblock is set to 32×32. The shaded partitions are the ones that will be tested. Note that, in the example (b), the PSKIP mode is used in the same way as the inter 16×16.

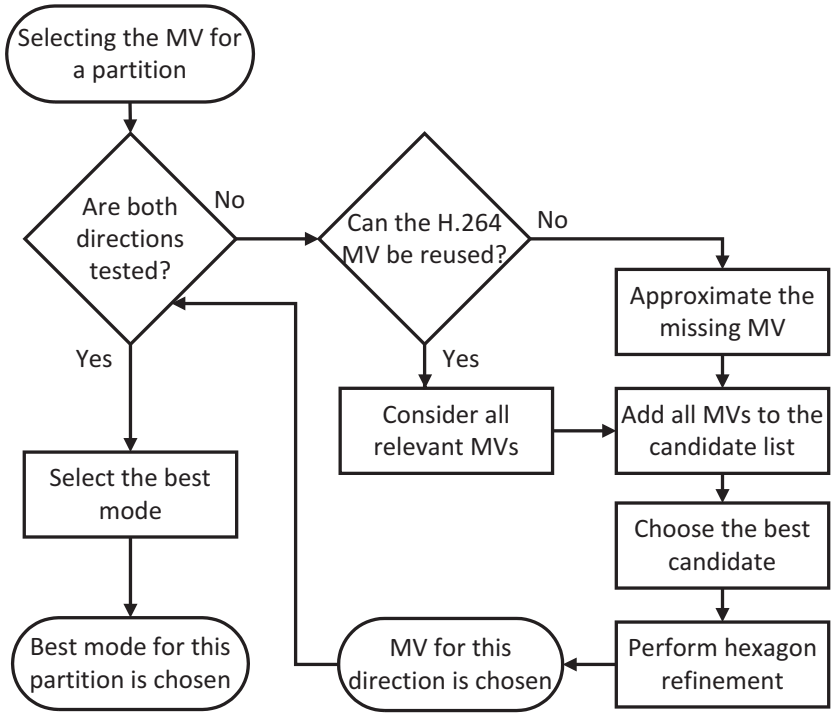


Figure 5.4.: Motion vector decision for a given partition.

to populate the candidate list(s). Following the discussion on Chapter 4, the proposed transcoder uses the following methods to populate the candidate lists:

(i) Spatial: This is the weighted average of MVs of the neighbouring blocks, as seen in Sec. 4.4.1.

(ii) Median: This is the median of the MVs of the neighbouring blocks, as seen in Sec. 4.4.1.

(iii) MV Scaling - mode: This is the MV Scaling algorithm (Sec. 4.2.1) using mode as the grouping algorithm (Sec 4.3.3).

(iv) Proposed MV Composition - mode: This is the proposed MV Composition algorithm (Sec 4.3.1) using the mode as the grouping algorithm (Sec 4.3.2), and aligning the partition with the 4×4 grid.

(v) MV Inversion: This is the method shown in Sec. 4.4.2. When used in the transcoder, this method inverts all motion vectors from the forward candidate list, adding them to the backward candidate list. This method can be useful for instance in *IPP* coding configuration when only forward motion vectors are available.

The rationale of using several methods is that each method performs well in a particular case, complementing each other. The added complexity of using more methods, and thus having a longer candidate list, is largely offset by the gains of having a better motion vector prediction. The transcoder then evaluates the cost of all motion vectors in each candidate list. Here, it is important to notice that the motion vectors in the list are kept at quarter-pixel precision. For each candidate in the two lists the cost is computed in a conventional way, considering the residual and the rate of the chosen motion vector. This cost is computed for every candidate in each list, and the best motion vector is used as the starting point for the refinement step, which is applied for each direction, and it is discussed in the following section. Finally, after the refinement is applied, the cost of bi-directional prediction is evaluated, using the best MV of each direction. No further refinement is made to test for the bi-directional prediction. The transcoder then decides the mode for this partition (forward, backward or bi-directional).

Motion Vector Refinement

The motion vector refinement is considered for all motion partitions tested, even if the H.264/AVC motion vectors are directly reused. The motion vector refinement first rounds the motion vectors for a integer pixel level. Then, it applies a refinement consisting of a Hexagon search [153]. More details on this algorithm are given in App. A.

Once the best integer motion vector is found, it tests all 8 neighbours at half-pixel level and then all 8 neighbours at quarter pixel level.

5.3. The Reduced Complexity Module

The goal of the techniques described so far is to achieve maximum quality (in terms of decoded video), while reducing the transcoder's complexity. The Reduced Complexity (RC) Module allows for a further reduction in complexity, at the cost of a small impact on quality. Three techniques are used in the RC Module: using the H.264/AVC CBP, the similarity of motion vectors and Adaptive Refinement Decision [6]. They are discussed in the following sections.

5.3.1. Using the H.264/AVC CBP

In the H.264/AVC codec, a decoded block is given as:

$$B_n^{'k} = P_n^k + R_n^{'k} + D_F\left(P_n^k + R_n^{'k}\right) \tag{5.1}$$

where $B_n^{'k}$ is the decoded block, P_n^k is the prediction for this block, $R_n^{'k}$ is the decoded residual and $D_F(\cdot)$ is the effect of the deblocking filter [78], which is applied to $P_n^k + R_n^{'k}$. Thus, if no coefficient is transmitted, then the residual is zero, and the decoded block is given as:

$$B_n^{'k} = P_n^k + D_F\left(P_n^k\right) \tag{5.2}$$

Since the W-SVC codec uses MCTF, motion estimation is performed using the original frames, which, in the transcoder, are the decoded H.264/AVC frames. To further reduce the complexity, the transcoder avoids refining the motion vectors for a partition if the whole partition uses a unique motion vector and if the residual for that block is zero. When this happens, the H.264/AVC motion vector is directly assigned to that partition, without further refinement. This is applied regardless of the partition size

considered. Other partitions may also be tested even when this happens, in which case the usual approach of approximation and refinement is used.

In the transcoder implementation, the syntax parameter coded block pattern (CBP) is used to check if the residual is zero. However, the CBP only tells the presence or absence of AC DCT coefficients in each 8×8 block inside the macroblock [62]. Thus, if the CBP for a given block is zero, it does not necessarily mean that the residual is zero, since it can still have a DC coefficient. Furthermore, the CBP does not have the information for each 4×4 block separately. However, tests comparing using the CBP or using the actual number of non-zero DCT coefficients to drive the transcoder yield very similar results. Therefore, the transcoder uses the CBP, since it is more readily available.

5.3.2. Motion Vector Similarity

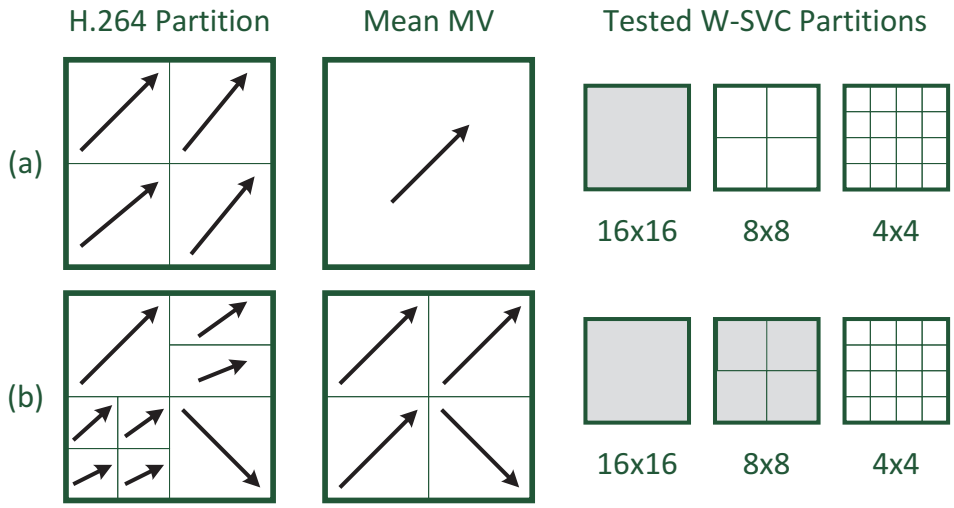


Figure 5.5.: Example of application of MV Similarity: (a) since all MVs are similar to the mean MV, only the 16×16 partition will be tested (the shaded partitions in the figure); (b) the MVs are similar for each 4×4 partition inside the 8×8 partition, but not for the 8×8 partition. Therefore, both 16×16 and 8×8 partitions will be tested. In this example, the W-SVC macroblock is 16×16, but the same principle is used if a larger macroblock size is used.

Since the W-SVC codec favours larger partitions, the transcoder may avoid testing smaller partitions when the H.264/AVC motion vectors corresponding to these partitions are similar [4]. A similar idea was reported in the context of a H.264/AVC to H.264/AVC transcoder based on spatial resolution reduction [77]. In this work, how-

ever, the similarity of the motion vectors is only used in order to map the modes for 8×8 sub-macroblocks for the reduced resolution.

In the technique developed here, first a mean MV, $\overline{mv^k_{n \to n-\alpha}}$, is computed for each 8×8 partition, using the four 4×4 blocks within each 8×8 partition. Then, each of the four motion vectors is compared to the mean MV of the encompassing 8×8 partition, using a similarity measure that considers each component separately.

Let $k0$, $k1$, $k2$ and $k3$ be the four 4×4 partitions within a given 8×8 partition k, and $mv^i_{n \to n-\alpha}.x$ and $mv^i_{n \to n-\alpha}.y$ be the horizontal and vertical components of $mv^i_{n \to n-\alpha}$, respectively. Then, a single motion vector $mv^i_{n \to n-\alpha}$ is considered similar to the mean MV if both these conditions are satisfied:

$$\begin{aligned} \left| \overline{mv^k_{n \to n-\alpha}}.x - mv^i_{n \to n-\alpha}.x \right| &< T \\ \left| \overline{mv^k_{n \to n-\alpha}}.y - mv^i_{n \to n-\alpha}.y \right| &< T \end{aligned} \tag{5.3}$$

where T is a threshold, which can be set as a parameter. Note that, inside each 8×8 partition, all motion vectors share the same reference frame. If all motion vectors for the 4×4 partitions $k0$, $k1$, $k2$ and $k3$ are considered similar for the 8×8 partition k, then 4×4 partitions will not be tested for this 8×8 block, regardless of the H.264/AVC partitioning.

The similarity measure is applied to the four 8×8 blocks within a 16×16 block in a similar manner: if the mean MV for the 8×8 partitions are considered similar, 8×8 partitions will not be tested for this block. In this case, however, the reference frame is not necessarily the same for all partitions. If they are not the same for all motion vectors, then they are considered as not similar. If larger block sizes are used, this process iterates in the same manner. This process applies whether or not the H.264/AVC motion vectors can be directly reused. An example is shown in Fig. 5.5.

The effect of the MV Similarity threshold in some test sequences is shown in Table 5.2. In this test, the first 100 frames of the test sequences were used, at CIF resolution and the H.264/AVC bitstream used QP 20. A macroblock size of 16×16 was used in the W-SVC. Although other thresholds could be used, it was decided to use a threshold of 0.5, in integer pixel scale, in order to keep the rate distortion performance when using

Table 5.2.: Effect of the MV Similarity threshold on the proposed transcoder. The average PSNR loss is computed using the trivial transcoder with full motion estimation as anchor, and the speed-up results are compared to the trivial transcoder with hexagon search. The threshold used in the remainder of this chapter is highlighted.

	Average PSNR Loss (dB)				Speed-up			
Threshold	City	Coastguard	Foreman	Soccer	City	Coastguard	Foreman	Soccer
not used	+0.06	+0.06	+0.01	0.00	1.45	1.16	1.21	1.31
0	+0.06	+0.06	+0.01	0.00	1.44	1.15	1.21	1.30
0.25	+0.05	+0.07	−0.02	−0.01	2.11	1.67	1.44	1.65
0.50	**+0.04**	**+0.07**	**−0.06**	**−0.03**	**2.13**	**1.82**	**1.62**	**1.71**
1	−0.07	+0.06	−0.10	−0.08	2.32	2.02	1.80	1.81
2	−0.09	+0.04	−0.14	−0.11	2.43	2.17	1.97	1.94

this technique close to the performance when this technique is not used (i.e., in order to limit the loss of the reduced complexity module). Also, it was found that, when the distance between the current frame and the reference frame becomes too large, the correlation between the H.264/AVC and the W-SVC motion information drops. Thus, this strategy is only used if the current frame and the target reference frame are distant from each other up to a threshold, which is set to 4 frames [3, 6]. A brief description of the test sequences used is given in Appendix C, and more details on the Average PSNR Loss can be found in Appendix D.

5.3.3. Adaptive Refinement Decision

After all motion vector candidates are considered, a SAD criterion is used to decide if this motion vector will be refined or not. This criterion is similar to that used in the Enhanced Predictive Zonal Search (EPZS) algorithm [132], and is also commonly found in other fast search algorithms. Here, a static threshold is used:

$$T_{SAD} = 2 \cdot N \cdot N \cdot \left(\sqrt{2}\right)^{l} \tag{5.4}$$

where N denotes the current partition size and l denotes the temporal decomposition level of the current frame (starting at $l = 0$). The factor $\left(\sqrt{2}\right)^{l}$ accounts for the change in the dynamic range of the frame after l temporal decompositions. If the SAD for the best candidate is lower than this threshold, no refinement is performed (not even

sub-pixel refinement), and the motion vector for that direction is chosen among the MV candidates for each candidate list.

5.4. Experimental Results

This section presents the experiments made to evaluate the proposed transcoder, both in terms of rate-distortion performance and complexity savings. The H.264/AVC implementation used is the JM 14.2 [64], while the W-SVC software used is the version 13.6 [124]. For all cases, the PSNR shown is the average among all frames of the luma component, and it is always computed using the original sequence as the reference. On the other hand, the bit-rate always includes both the luminance and the chrominance components.

In order to evaluate the transcoder flexibility, four different coding structures are used in the H.264/AVC (as seen in Sec. 3.1.3). They are:

(i) *IPP1*: this is the *IPP* configuration with 1 reference frame (as seen in Fig. 3.5(a)).

(ii) *IPP5*: this is the *IPP* configuration with 5 reference frames.

(iii) *IBBP*: this is the *IBBP* configuration with 1 reference frame each side for B-frames and 5 reference frames for the P-frames (as seen in Fig. 3.5(b)).

(iv) *Hierarchical*: this is the *Hierarchical* configuration with three levels of hierarchy, and 1 reference frame each side for B-frames and 1 reference frame for P-frames (as seen in Fig. 3.5(c)).

Four different sequences at CIF resolution (352×288) with 30 frames per second (fps) and CIF resolution (704×576 pixels) with 60 fps are used in the experiments: *City*, *Crew*, *Harbour* and *Soccer*. A brief description of these sequences is given in Appendix C. The length of all sequences is 10 seconds, and the number of frames used depends on the H.264/AVC coding configuration used, as is shown in Table 5.3. In addition to this, two quantization parameters are used in the H.264/AVC: 20 and 28 for CIF resolution, and 26 and 34 for 4CIF resolution.

The experiments follow the same idea as the experiment described in Sec. 4.5.2. First, the original sequence is encoded using the H.264/AVC, using a specific QP and coding configuration. The output bitstream is then transcoded to the W-SVC using

Table 5.3.: Number of frames used for each coding configuration.

	CIF	4CIF
IPP1	300	600
IPP5	300	600
IBBP	298	597
Hierarchical	297	593

Table 5.4.: Number of temporal decompositions performed in the W-SVC codec for each sequence and resolution.

	CIF	4CIF
City	5	5
Crew	3	4
Harbour	6	6
Soccer	3	4

the H.264/AVC to W-SVC transcoder, generating a scalable bitstream. This scalable bitstream is then extracted and decoded at specific bitrates. In the W-SVC codec, independently on the coding configuration and the QP used in the H.264/AVC, the following bitrates were used: $\{384, 480, 576, 720, 1152, 1536, 2304\}$ kbps, for CIF resolution; and $\{1280, 1536, 1792, 2048, 2304, 2688, 3072\}$ kbps, for 4CIF resolution. Also, in order to account for different W-SVC codec configurations, a different number of temporal decompositions was used for each sequence. This is shown in Table 5.4. Finally, three macroblock sizes are tested in the W-SVC codec: 16×16, 32×32 and 64×64.

5.4.1. Reference Transcoders

In this section, the proposed transcoder is defined as the transcoder using the techniques in Sec. 5.2 (i.e., the algorithm to handle H.264/AVC partitions and the Framework for MV Approximation and Refinement), while the proposed transcoder with the RC module also uses techniques found in Sec. 5.3 (i.e., using the H.264/AVC CBP, MV Similarity and Adaptive Refinement Decision). They are referred to as PT and PT-RC, respectively, in the remainder of this section.

The proposed transcoders are compared to three reference trivial transcoders: (i) using full motion estimation; (ii) using Hexagon Search [153]; and (iii) using EPZS [132].

They are referred as RT-FS, RT-HS and RT-EPZS, respectively, in the remainder of this section. The full motion estimation uses a search window of 32, while the others use a search window of 60, in integer-pixel scale. The search window for RT-FS is smaller due to the complexity, which increases with the window size. For all cases, including the proposed transcoder, the search window enlarges for higher levels of temporal decomposition. The motion estimation for both RT-FS and RT-HS start at the predicted motion vector. The EPZS algorithm implemented here is the same as that implemented in the H.264/AVC reference code, JM 14.2 [64]. The only modification of the algorithm is that the dynamic range change due to temporal filtering is considered. Both the Hexagon search and the EPZS algorithm are fully detailed in Appendix A. Furthermore, in order to reduce complexity, all three reference transcoders test the partitions in the following pattern: first, the $N \times N$ and $\frac{N}{2} \times \frac{N}{2}$ partitions are tested, then, only if the latter has a lower cost, the $\frac{N}{4} \times \frac{N}{4}$ partitions are tested. This procedure is repeated until the minimum size is reached (always 4×4, regardless of the macroblock size). Testing all partitions leads to a marginal gain in terms of quality, but it increases complexity by 115%, on average, if a macroblock size of 16×16 is used.

5.4.2. Transcoder Loss

The first point analyzed here is the loss caused by transcoding. Figs. 5.6 and 5.7 show the performance of the W-SVC codec operating on the original sequence (Original W-SVC in the figures), and the performance of the trivial transcoder using full search (RT-FS) operating on two H.264/AVC bitstreams, at $QP = 26$ and $QP = 34$. The figures also show the quality of the H.264/AVC sequence on which the transcoder operates. Furthermore, the figure shows these results for the three macroblock sizes considered in the W-SVC. For both H.264/AVC bitstreams, the *IBBP* coding configuration is used. Other configurations show similar results, as the quantization parameter used is the same and, thus, the quality of the H.264/AVC sequence is similar.

First, it can be seen that a certain loss of quality is present in transcoding, especially at medium and higher bitrates. This is expected, since the transcoder operates on a quantized sequence, and it is therefore limited to the quality of this sequence. This loss can be reduced by using larger block sizes, with a significant improvement in performance when using the 32×32 macroblock size (compared to the 16×16) and a small improvement when using the 64×64 macroblock size (compared to the 32×32). The average impact (among all bitrates) of the macroblock size used is shown in Table 5.5.

Table 5.5.: Average PSNR gain for different MB Sizes in the transcoder.

	Original W-SVC			RT-FS (QP 26)			RT-FS (QP 34)		
	MB 16	MB 32	MB 64	MB 16	MB 32	MB 64	MB 16	MB 32	MB 64
City	0.00	+0.29	+0.27	0.00	+0.28	+0.30	0.00	+0.11	+0.12
Crew	0.00	+0.55	+0.59	0.00	+0.48	+0.52	0.00	+0.22	+0.25
Harbour	0.00	+0.41	+0.44	0.00	+0.47	+0.52	0.00	+0.23	+0.25
Soccer	0.00	+0.79	+0.87	0.00	+0.69	+0.77	0.00	+0.28	+0.32
Average	**0.00**	**+0.51**	**+0.54**	**0.00**	**+0.48**	**+0.53**	**0.00**	**+0.21**	**+0.23**

As expected, the impact is higher when the transcoder is operating in a higher quality bitstream (+0.51 dB when using the 32×32 size, and +0.54 dB when using the 64×64 size, on average for all sequences, comparing to the 16×16), but it is noticeable even for lower quality bitstreams (+0.21 dB when using the 32×32 size, and +0.23 dB when using the 64×64 size, on average for all sequences, comparing to the 16×16). The highest impact on the transcoder quality is of +1.01 dB using a 32×32 MB size and +1.17 dB using a 64×64 MB size (as seen in Fig. 5.7(e) and Fig. 5.7(f), respectively), both for the lowest bitrate (1280 kbps) of Soccer sequence using QP 26, both compared to the 16×16 size (seen in Fig. 5.7(d)). Thus, using larger block sizes can significantly reduce the transcoding loss, however, most of the gain is already present at the 32×32 MB size.

5.4.3. Evaluating the Quality of the Proposed Transcoder

Here, the transcoder is evaluated in terms of the decoded video quality. The results for RT-HS, RT-EPZS, PT and PT-RC are shown as the PSNR loss relative to RT-FS, which is used as benchmark. The Average PSNR Loss is detailed in Appendix C. Selected results are shown in Figs. 5.8 and 5.9. Figs 5.10 and 5.11 show the minimum, average and maximum loss for each option and coding configuration (among the four sequences and two H.264/AVC QPs), for CIF and 4CIF resolutions, respectively. Finally, Table 5.6 summarizes the results. The average PSNR Loss for each sequence tested can be found in the Appendix E, from Tables E.1 through E.4, for each coding configuration.

It can be seen that PT consistently outperforms all reference transcoders (RT-HS and RT-EPZS), for all sequences, configurations and resolutions tested. For some sequences, it even outperforms RT-FS by a small margin, and in a few cases it outperforms RT-FS by a larger margin (up to +0.22 dB, for Soccer 4CIF sequence using $QP = 26$, MB size

Table 5.6.: Average PSNR Loss for all coding configurations, comparing to RT-FS.

		16 × 16				32 × 32				64 × 64			
	Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF	*IPP1*	−0.39	−0.07	+0.03	+0.00	−0.24	−0.04	+0.05	−0.02				
	IPP5	−0.40	−0.07	+0.03	+0.01	−0.24	−0.04	+0.05	+0.01				
	IBBP	−0.37	−0.07	+0.01	−0.02	−0.23	−0.04	+0.03	−0.01	Not Available			
	Hierarchical	−0.39	−0.08	+0.03	+0.00	−0.25	−0.05	+0.04	+0.01				
	Average	**−0.38**	**−0.07**	**+0.02**	**+0.00**	**−0.24**	**−0.04**	**+0.04**	**+0.00**				
4CIF	*IPP1*	−0.35	−0.08	+0.05	−0.03	−0.20	−0.02	+0.04	+0.03	−0.14	−0.01	+0.06	+0.04
	IPP5	−0.36	−0.08	+0.05	−0.01	−0.20	−0.02	+0.05	+0.03	−0.13	+0.00	+0.06	+0.04
	IBBP	−0.35	−0.08	+0.04	+0.01	−0.20	−0.02	+0.04	+0.02	−0.13	−0.01	+0.06	+0.03
	Hierarchical	−0.36	−0.08	+0.06	−0.03	−0.20	−0.02	+0.03	+0.00	−0.13	−0.01	+0.05	+0.03
	Average	**−0.36**	**−0.08**	**+0.05**	**−0.01**	**−0.20**	**−0.02**	**+0.04**	**+0.02**	**−0.13**	**−0.01**	**+0.06**	**+0.03**

of 64 × 64 and *IPP1* and *IPP5* configurations, seen in Tables E.1 and E.2, respectively). Again, as seen in Sec. 4.5.2, there are a number of reasons why the proposed transcoder may outperform the full search technique: (i) the full search is only performed at integer pixel level, not at sub-pixel level; (ii) the proposed transcoder uses a larger search window for motion estimation (60, instead of 32); and (iii) the proposed transcoder also benefits from the partitioning found in the H.264/AVC.

Using the RC Module (PT-RC) has a very low impact on the transcoder quality, of −0.02 and −0.04 dB, using a MB size of 16 × 16 and 32 × 32, respectively, for CIF resolution (on average, as seen in Table 5.6), and −0.06, −0.02 and −0.02 dB, using a MB size of 16 × 16, 32 × 32 and 64 × 64, respectively, for 4CIF resolution (on average, as seen in Table 5.6). Moreover, in only few cases the loss exceeds −0.1 dB (for two out of the 32 cases tested using a MB size of 32 × 32 and CIF resolution, and for eight out of the 32 cases tested using a MB size of 16 × 16 and 4CIF resolution - these cases can be seen in Tables E.1 through E.4). The maximum loss occurs for Crew 4CIF *Hierarchical* 16 × 16, of −0.29 dB, on average, which is shown in Fig. 5.9(b). In only very few cases (3% of the cases tested), RT-EPZS shows a slightly better average performance than PT-RC. However, even for these cases, PT-RC still yields a better PSNR performance for high bitrates.

Subjective Evaluation of the Proposed Transcoder

In order to further verify the performance of the transcoding options, a subjective test was conducted, with similar settings as the test in Sec. 4.5.3. The subjects are exposed to two versions of the video: one of them (in random order) is always the original,

Table 5.7.: Comparison among Transcoding methods. In all cases, the MB size used is 16×16, the bitrate is 1280 kbps, the coding configuration is *IPP1* and the resolution is 4CIF.

	Metric	Original	RT-FS	RT-HS	RT-EPZS	PT	PT-RC
City	PSNR	∞	33.6	32.8	33.3	33.8	33.6
	MSSIM	1.00	0.91	0.89	0.90	0.91	0.91
	MOS	78	61	43	54	61	60
Crew	PSNR	∞	33.6	33.1	33.4	33.7	33.4
	MSSIM	1.00	0.86	0.85	0.86	0.86	0.86
	MOS	70	40	30	34	32	35
Harbour	PSNR	∞	29.8	29.4	29.7	29.9	29.8
	MSSIM	1.00	0.88	0.87	0.88	0.88	0.88
	MOS	68	53	44	54	51	47
Soccer	PSNR	∞	32.0	31.0	31.9	32.2	32.0
	MSSIM	1.00	0.84	0.80	0.84	0.84	0.84
	MOS	77	42	37	37	39	39

uncompressed video, and the other is the video to be tested. Then, they are asked to rate each video, in a scale from 0 to 100. The test was conducted with 14 subjects, and the results are shown in Table 5.7, as the MOS (mean opinion score), along with the PSNR and MSSIM for each sequence (see Appendix D for more details on quality metrics).

It can be seen that the MOS for four of the transcoding options (RT-FS, RT-EPZS, PT and PT-RC) are close to each other. In fact, only for City sequence the MOS for RT-HS is significantly lower than the other options. Also, in this experiment, the MOS agrees with the PSNR and MSSIM values shown in Table 5.7, showing that both PT and PT-RC have comparable quality to the trivial transcoder using full motion estimation (RT-FS).

5.4.4. Evaluating the Complexity of the Proposed Transcoder

Since the transcoder is mainly based on reducing the motion estimation and mode decision complexity, the analysis in the remainder of this section is focused on this module. However, the complexity of the other modules (such as the wavelet transforms, entropy coding and interpolation) is briefly discussed here, compared to the motion estimation and mode decision modules. Some experiments have shown that the motion estimation and mode decision modules account for 97% of the execution time of RT-FS, for 4CIF resolution. It is also worth mentioning that the W-SVC codec used is not entirely optimised in the software part, especially the interpolation and entropy coding modules.

Table 5.8.: Higher bound for transcoder speed-up, comparing to RT-HS, measured as the number of SAD Calculations.

			16 × 16				32 × 32				64 × 64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
Speed Up (SAD Operations)	CIF	*IPP1*	1.00	0.77	1.45	2.42	1.00	0.76	1.03	1.70				
		IPP5	1.00	0.77	1.45	2.31	1.00	0.76	1.03	1.54				
		IBBP	1.00	0.77	1.54	2.30	1.00	0.77	1.07	1.56	Not Available			
		Hierarchical	1.00	0.78	1.69	2.89	1.00	0.77	1.14	1.78				
		Average	**1.00**	**0.77**	**1.53**	**2.48**	**1.00**	**0.77**	**1.07**	**1.64**				
	4CIF	*IPP1*	1.00	0.83	1.74	3.43	1.00	0.85	1.10	2.13	1.00	0.83	0.85	1.49
		IPP5	1.00	0.83	1.75	3.40	1.00	0.85	1.11	2.01	1.00	0.83	0.86	1.39
		IBBP	1.00	0.83	1.82	3.14	1.00	0.85	1.13	2.01	1.00	0.83	0.86	1.42
		Hierarchical	1.00	0.84	1.90	5.10	1.00	0.86	1.18	2.56	1.00	0.84	0.87	1.58
		Average	**1.00**	**0.83**	**1.81**	**3.77**	**1.00**	**0.85**	**1.13**	**2.18**	**1.00**	**0.83**	**0.86**	**1.47**

Therefore, it is expected that the complexity of the motion estimation and mode decision modules may account for an even higher part of total complexity of the codec.

In this section, the complexity of the transcoder is measured both in terms of the average number of SAD operations and running time. As seen in previous chapters, in the W-SVC codec, the cost of each motion vector is computed as $J = D + \lambda \cdot R$. The complexity of estimating the rate R is fairly low, since R is calculated for the whole block by simple prediction of the neighbouring motion information. However, calculating the distortion D, which is measured as the SAD, involves many more operations, and it is the most complex operation in the motion estimation module, even if fast motion estimation methods are used. The SAD of a block B can be expressed as:

$$SAD_B = \sum_{i=0}^{M-1} |p_i - \hat{p}_i| \tag{5.5}$$

where M is the number of considered pixels in the block, p_i is the luma component of the pixel with the index i in the block B and $\hat{p}_i$ is the luma component of the pixel with the index i in the reference block. Following the equation, a SAD operation is defined as calculating a difference between two pixels, calculating the absolute value of that difference and adding the absolute difference to the sum of previously calculated absolute differences. Thus, calculating the SAD for the block B consists of M SAD operations. Using the number of SAD operations as an indication of complexity has the advantage of being an objective measure, independent of the software optimizations and the hardware used for experiments. However, it does not account for the complexity

Table 5.9.: Implemented transcoder speed-up, comparing to RT-HS.

			16 × 16				32 × 32				64 × 64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
Speed Up (Elapsed Time)	CIF	*IPP1*	1.00	0.58	1.16	1.84	1.00	0.64	0.85	1.39				
		IPP5	1.00	0.58	1.15	1.72	1.00	0.64	0.85	1.24				
		IBBP	1.00	0.58	1.26	1.79	1.00	0.64	0.90	1.28	Not Available			
		Hierarchical	1.00	0.59	1.48	2.31	1.00	0.64	1.04	1.57				
		Average	**1.00**	**0.58**	**1.26**	**1.91**	**1.00**	**0.64**	**0.91**	**1.37**				
	4CIF	*IPP1*	1.00	0.65	1.52	2.62	1.00	0.75	0.96	1.72	1.00	0.78	0.76	1.27
		IPP5	1.00	0.65	1.53	2.57	1.00	0.75	0.97	1.63	1.00	0.78	0.77	1.20
		IBBP	1.00	0.65	1.62	2.50	1.00	0.76	1.00	1.66	1.00	0.78	0.77	1.22
		Hierarchical	1.00	0.65	1.77	3.84	1.00	0.76	1.10	2.19	1.00	0.79	0.83	1.45
		Average	**1.00**	**0.65**	**1.61**	**2.88**	**1.00**	**0.76**	**1.01**	**1.80**	**1.00**	**0.78**	**0.78**	**1.29**

Table 5.10.: Efficiency of the implemented transcoder.

			16 × 16		32 × 32		64 × 64	
		Sequence	PT	PT-RC	PT	PT-RC	PT	PT-RC
Efficiency (Percentage)	CIF	*IPP1*	80	76	83	82		
		IPP5	79	75	83	81		
		IBBP	82	78	84	82	Not Available	
		Hierarchical	87	80	91	88		
		Average	**82**	**77**	**85**	**83**		
	4CIF	*IPP1*	87	76	87	81	89	86
		IPP5	87	76	87	81	90	87
		IBBP	89	80	89	83	89	86
		Hierarchical	93	75	93	86	95	92
		Average	**89**	**77**	**89**	**82**	**91**	**88**

of the MV approximation techniques, or computing the motion vector similarity in the RC module, for instance, and it is therefore a lower bound of the transcoder complexity. This is useful to better understand the maximum speed-up that can be achieved for each transcoding option, and also to check how efficient is a given transcoding implementation.

The average, minimum and maximum transcoder speed up, for each option and coding configuration, measured as the number of SAD operations, is shown in Figs 5.12 and 5.13, for CIF and 4CIF resolutions, respectively. The average speed up is shown in Table 5.8. In both cases, the speed-up for RT-EPZS, PT and PT-RC are shown relative to RT-HS. The complete results, for each sequence tested, are shown in Appendix E, from Tables E.5 through E.8. In these tables, it is shown the number of SAD operations, using RT-FS as anchor.

To measure the running time, a PC with an Intel Core i5-2400 running at 3.10 GHz with 8 Gb of RAM and Windows 7 64-bit was used. Again, since the transcoder is mainly based on motion estimation and mode decision module, only the time spent

on these modules is considered here. Each sequence was encoded 3 times, and the average time spent on the motion estimation and mode decision for all frames was measured. These results are shown as the Speed-Up relative to the RT-HS, which is used as a benchmark for speed. Figures 5.14 and 5.15 show the minimum, average and maximum speed-ups, for different options and coding configurations, for CIF and 4CIF resolutions, respectively, and Table 5.9 shows the average results. The complete results, for each sequence tested, are shown in Appendix E, from Tables E.9 through E.12. In this case, all modules of the transcoder are accounted for, including the MV Approximation methods, computing MV Similarity, and deciding the partition using the H.264/AVC motion information. Therefore, this is the complexity of the implemented transcoder, and could be even reduced (to the limit given by the lower bound, discussed previously) if a thorough optimisation of the software is carried out. Still, it can be seen that the efficiency of the transcoder implementation, shown in Table 5.10 and computed as the ratio between the implemented transcoder speed-up and the lower bound given by the number of SAD operations, is good, ranging from 79% (on average, for *IPP5* configuration, CIF resolution and MB size of 16×16) to 95% (on average, for *Hierarchical* configuration, 4CIF resolution and MB size of 64×64) for PT, and from 75% (on average, for *IPP5* configuration, CIF resolution and MB size of 16×16 and also for *Hierarchical* configuration, 4CIF resolution and MB size of 16×16) to 92% (on average, for *Hierarchical* configuration, 4CIF resolution and MB size of 64×64) for PT-RC. Note that an efficiency of 100% cannot be achieved, since the lower bound does not consider many operations in the transcoder.

Both transcoding options (PT and PT-RC) and the two trivial transcoders using fast motion estimation (RT-HS and RT-EPZS) are much faster than RT-FS. In a test for *IPP1* configuration, PT was from 130 to 300 times faster than RT-FS, for CIF resolution, and from 250 to 400 times faster, for 4CIF resolution. For other coding configurations, even larger speed-ups can be achieved.

From the tables, it can be seen that the complexity of PT and PT-RC are dependent on the coding configuration used, the QP of the H.264/AVC, the resolution of the sequence and the macroblock size used.

Using a MB size of 16×16, PT is always faster than RT-EPZS (using the same MB size), and it is faster than RT-HS for most coding configurations and resolutions, except for a few cases (8% of all cases tested) for CIF resolution, for Harbour and Crew sequences in *IPP1* and *IPP5* configurations. On average, PT is 1.26 times faster, for CIF resolution, and 1.61 times faster, for 4CIF resolution (as seen in Table 5.9). In the

worst case, PT is slower than RT-HS (with a speed-up of 0.81, for Harbour CIF *IPP5* QP 20, seen in Table E.10), and in the best case it is 2.21 times faster than RT-HS (for City 4CIF *Hierarchical* QP 26, seen in Table E.12). Using a MB size of 32×32, PT is, on average, as fast as RT-HS (using the same MB size), for 4CIF resolution (1.01), but slightly slower for CIF resolution (0.91), as shown in Table 5.9. In the worst case, it is slower than RT-HS (0.57, for Harbour CIF *IPP5* QP 20, shown in Table E.10), and in the best case it is 1.34 times faster than RT-HS (for City 4CIF *Hierarchical* QP 34, shown in Table E.12). Finally, for a MB size of 64×64, PT is slower, on average, than RT-HS (0.78 times), with a worst case of 0.53 (Harbour 4CIF *IPP1* QP 26, shown in Table E.9) and in the best case of 0.97 (for City 4CIF *Hierarchical* QP 34, shown in Table E.12).

Using the Reduced Complexity module significantly speeds up the transcoder, with PT-RC being 1.66 times faster than PT, on average (which can be concluded from Table 5.9). Using a MB of 16×16, PT-RC is always faster than both RT-EPZS and RT-HS for CIF resolution, and at least 1.5 times faster than RT-HS for 4CIF resolution (and 2.88 times faster, on average, as seen in Table 5.9). In the worst case, it is 1.07 times faster (Harbour CIF *IPP5* QP 20, shown in Table E.9), and in the best case, it is 7.2 times faster than RT-HS (City 4CIF *Hierarchical* QP 34, shown in Table E.12). Using a MB size of 32×32, PT-RC is faster than RT-HS (using the same MB size), on average, for both CIF (1.37 times) and 4CIF resolutions (1.80 times), as shown in Table 5.9. In only 12% of the cases tested PT-RC was slower than RT-HS. In the worst case, it is slower (0.77, for Harbour CIF *IPP5* QP 20, shown in Table E.10), and in the best case, it is 3.89 times faster than RT-HS (City 4CIF *Hierarchical* QP 34, shown in Table E.12). Finally, for a MB size of 64×64, PT-RC is still faster than RT-HS, on average (1.29 times), for 4CIF resolution. However, in a quarter of the cases tested PT-RC was slower than RT-HS. In the worst case, it is slower (0.66, for Harbour 4CIF *IBBP* QP 26, shown in Table E.11), and in the best case, it is 2.38 times faster than RT-HS (City 4CIF *Hierarchical* QP 34, seen in Table E.12).

Note that both PT and PT-RC are generally slower for Harbour and Crew sequences, and are generally faster for City sequence, regardless of the coding configuration. As an example, Table 5.11 shows the partitioning profile of the H.264/AVC bitstreams for *IPP1* configuration, 4CIF resolution and $QP = 26$, as well as the partitions tested in the W-SVC codec by PT and the speed up figures for PT and PT-RC. It can be seen that PT tests a much higher number of 4×4 and 8×8 partitions for both Crew and

Table 5.11.: Profile of the H.264/AVC Partitions (in percentage), partitions tested by the W-SVC transcoder (in percentage) and speed up figures for the implemented transcoder for *IPP1* configuration and 4CIF resolution.

H.264/AVC Mode	City	Crew	Harbour	Soccer
Intra	0.57	20.63	1.24	3.79
SKIP	31.31	19.04	6.08	34.31
Inter 16×16	31.83	30.37	36.71	29.45
Inter 16×8	10.16	10.26	13.67	11.05
Inter 8×16	10.35	12.27	17.91	9.41
Inter 8×8	9.58	5.16	15.76	7.34
Inter 8×4	2.67	1.01	3.31	2.52
Inter 4×8	3.08	1.15	4.98	1.80
Inter 4×4	0.45	0.11	0.34	0.34

Partitions Tested by PT				
W-SVC Partition	City	Crew	Harbour	Soccer
16×16	100	100	100	100
8×8	36.86	50.59	57.21	36.25
4×4	6.77	22.9	9.87	8.45

Speed up of the Implemented Transcoder				
Transcoder	City	Crew	Harbour	Soccer
PT	1.57	1.22	1.19	1.44
PT-RC	2.83	1.68	1.70	2.46

Harbour sequences, comparing to City and Soccer sequences. The reason is the high number of intra blocks for Crew sequence (20.63%) and the high partitioning of Harbour sequence (where only 36.71% of the partitions are encoded as 16×16, and only 6.08% are skipped). For both City and Soccer sequences, the amount of larger partitions is considerably higher, speeding up the transcoder. When the RC module is used, the high number of skipped macroblocks of City and Soccer sequence also helps to speed up the transcoder.

It can be seen that both PT and PT-RC are faster for *Hierarchical* configuration. This is due to the fact that more H.264/AVC motion vectors can be directly reused in this configuration, compared to the other configurations tested. Also, the transcoder is usually faster when operating on the lower quality sequences due to the fact that less partitions are tested, since the incidence of larger blocks is higher when the QP is higher. Finally, the way PT decides which partitions will be tested works well if the same MB size of the H.264/AVC is used in the W-SVC, otherwise, PT tests too many partitions (as all 64×64 and 32×32 partitions are tested, plus some partitions derived from the H.264/AVC MBs). Using the Reduced Complexity module, this problem is largely addressed, as smaller partitions are not tested if their motion vectors are similar.

5.4.5. Evaluating the effect of the MB size

In the previous section, the speed-up for each transcoding option was always compared to RT-HS using the same MB size. This is useful to understand how each transcoding option performs against the trivial transcoder, but it gives no information on how the same transcoder performs if different MB sizes are used. Here, the effect of using different

Table 5.12.: Comparing the transcoder speed-up for different MB Sizes.

		PT			PT-RC		
		MB 16	MB 32	MB 64	MB 16	MB 32	MB 64
CIF Res.	*IPP1*	1.00	0.71		1.00	0.73	
	IPP5	1.00	0.73		1.00	0.71	
	IBBP	1.00	0.69	N/A	1.00	0.70	N/A
	Hierarchical	1.00	0.68		1.00	0.66	
	Average	**1.00**	**0.70**		**1.00**	**0.70**	
4CIF Res.	*IPP1*	1.00	0.69	0.49	1.00	0.72	0.48
	IPP5	1.00	0.70	0.50	1.00	0.70	0.47
	IBBP	1.00	0.67	0.47	1.00	0.72	0.48
	Hierarchical	1.00	0.68	0.47	1.00	0.63	0.39
	Average	**1.00**	**0.68**	**0.48**	**1.00**	**0.69**	**0.45**

MB sizes is discussed. Table 5.12 shows the speed-up of both PT and PT-RC using the MB size of 16×16 as reference.

It can be seen that using a MB size of 32×32 increases the complexity of about 44%, for both PT and PT-RC. Increasing the MB size to 64×64 increases the complexity by another 47%, compared to the 32×32 MB size (or about 112% compared to the 16×16 MB size).

Recall from Sec. 5.4.3, in particular Table 5.6, that the PSNR performance of both PT and PT-RC does not vary much when different MB sizes are used (i.e., the PSNR loss compared to RT-FS using the same block size is steady). In addition, from Sec. 5.4.2, in particular Table 5.5, it can be seen that most of the PSNR gain of using larger MB sizes is present when using a MB size of 32×32. In particular, using a MB size of 32×32 presented a +0.48 dB PSNR gain, on average, compared to using a MB size of 16×16, and using a MB size of 64×64 presented only a +0.05 dB PSNR gain on top of the 32×32 MB size, when considering higher quality H.264/AVC bitstreams (4CIF, QP 26). For lower quality bitstreams, these figures are lower, but the conclusion remains true (a PSNR gain of +0.21 dB and 0.02 dB, respectively, for QP 34).

Subjective Evaluation of the MB Sizes

A subjective test was conducted to investigate the effect of using different MB sizes within the proposed transcoder (PT). The conditions were identical to the test in Sec. 5.4.3, except that this test involved 13 subjects. The results are shown in Table 5.13.

Table 5.13.: Comparison among MB Sizes. In all cases, the bitrate is 1280 kbps, the coding configuration is *IPP1*, the resolution is 4CIF and the transcoding option studied is the proposed transcoder PT.

	Metric	Original	PT 16×16	PT 32×32	PT 64×64
City	PSNR	∞	33.8	34.4	34.4
	MSSIM	1.00	0.91	0.91	0.92
	MOS	77	66	64	58
Crew	PSNR	∞	33.7	34.4	34.5
	MSSIM	1.00	0.86	0.88	0.88
	MOS	78	35	41	41
Harbour	PSNR	∞	29.9	30.9	31.0
	MSSIM	1.00	0.88	0.90	0.90
	MOS	74	47	58	58
Soccer	PSNR	∞	32.2	33.3	33.5
	MSSIM	1.00	0.84	0.87	0.88
	MOS	79	42	54	48

It can be seen from the table that, except for City sequence, the MOS when using a 16×16 MB size is lower than that for larger MB sizes. Also, it can be seen that there is little perceivable difference between the MB sizes of 32×32 and 64×64, in fact, the MOS when using a MB size of 32×32 is always the same or even slightly higher than that when using a MB size of 64×64. Finally, even though the MOS when using a MB size of 16×16 is the highest for City sequence, it is actually very close to that when using a MB size of 32×32.

5.5. Conclusions

In this chapter, a transcoder from H.264/AVC to a wavelet-based SVC (W-SVC) was presented. The transcoder is mainly based on transcoding the motion information between the two codecs. Its key features are flexible motion approximation techniques able to cope with multiple coding configurations in H.264/AVC, efficient optimization of partition sizes used in motion compensation, and a reduced complexity configuration based on H.264/AVC motion vectors' similarity, Coded Block Pattern information and adaptive decision on MV refinement.

Two main configurations for this transcoder are proposed. The first configuration, denoted as the proposed transcoder, PT, is designed to offer the best quality possible, while still saving complexity, compared to the trivial transcoder. The transcoder works at the macroblock level: the partition of the H.264/AVC macroblock is used to decide

which partitions are tested on the transcoder (Sec. 5.2.1), and a framework to make optimal use of the H.264/AVC motion vectors on the transcoder was developed (Sec. 5.2.2).

The second configuration is denoted as the proposed transcoder with the reduced complexity module, PT-RC (Sec. 5.3), and it is designed to further reduce the transcoding complexity. To achieve this goal, it allows a small loss of quality on top of the proposed transcoder. It uses the information on the DCT coefficients for each H.264/AVC macroblock, the similarity of the H.264/AVC motion vectors, and an adaptive MV refinement decision to avoid testing some partitions and modes on the transcoder, therefore reducing the complexity. As it tests a smaller number of modes and partitions, a small loss of quality is also present, compared to the proposed transcoder.

All proposed transcoders are evaluated against three reference trivial transcoders, using full motion estimation, Hexagon search and EPZS (RT-FS, RT-HS and RT-EPZS, respectively), both in terms of decoded video quality and complexity. Thorough evaluation shows that the proposed transcoder offers a performance close to RT-FS, which is confirmed with subjective evaluation, but keeps the complexity much lower than RT-FS and RT-EPZS.

Following the discussion in Sec. 5.4, it can be concluded that, if the best rate-distortion and complexity compromise is sought, then PT-RC using a block size of 32×32 offers the best performance, as it is significantly faster than other options (such as PT using a MB size of 32×32, or PT and PT-RC using a MB size of 64×64), but still offers a quality close to these options. If complexity is not the issue, than PT using a block size of 64×64 offers the best rate-distortion performance, even outperforming RT-FS. However, if the minimum complexity is sought, then PT-RC using a MB size of 16×16 offers the fastest transcoding, while still yielding a similar rate distortion performance to PT and RT-FS using the same MB size.

The next chapter discusses several transcoding options to the new HEVC codec. Since the HEVC also follows the DPCM/DCT model, the transcoder for this codec presents different issues from the transcoder seen in this chapter.

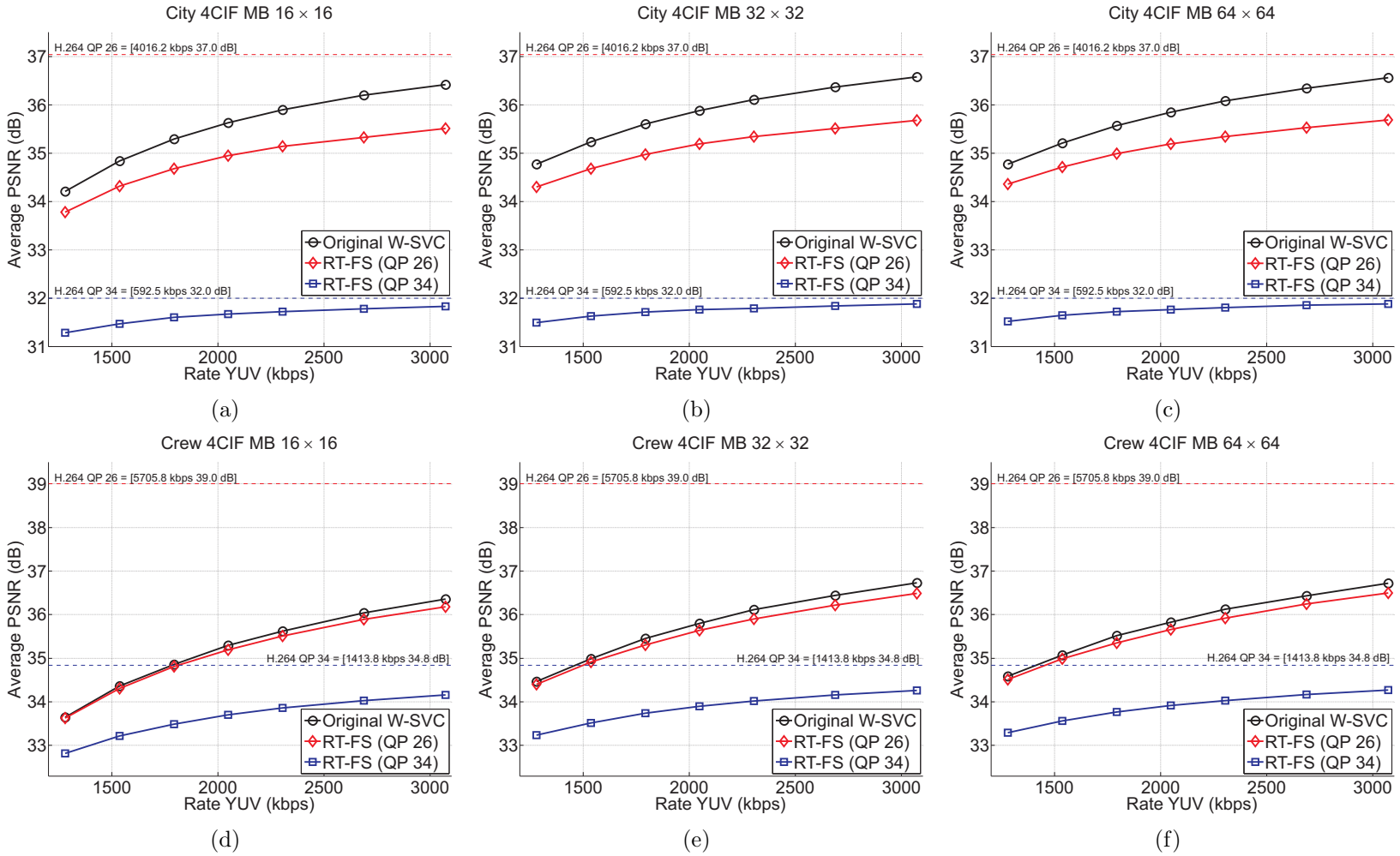


Figure 5.6.: Results of the W-SVC codec and RT-FS for: City 4CIF (a) 16×16; (b) 32×32; and (c) 64×64; and Crew 4CIF (d) 16×16; (e) 32×32; and (f) 64×64. Note that Fig. (a), (b) and (c), and Figs. (d), (e) and (f) are shown using the same axis.

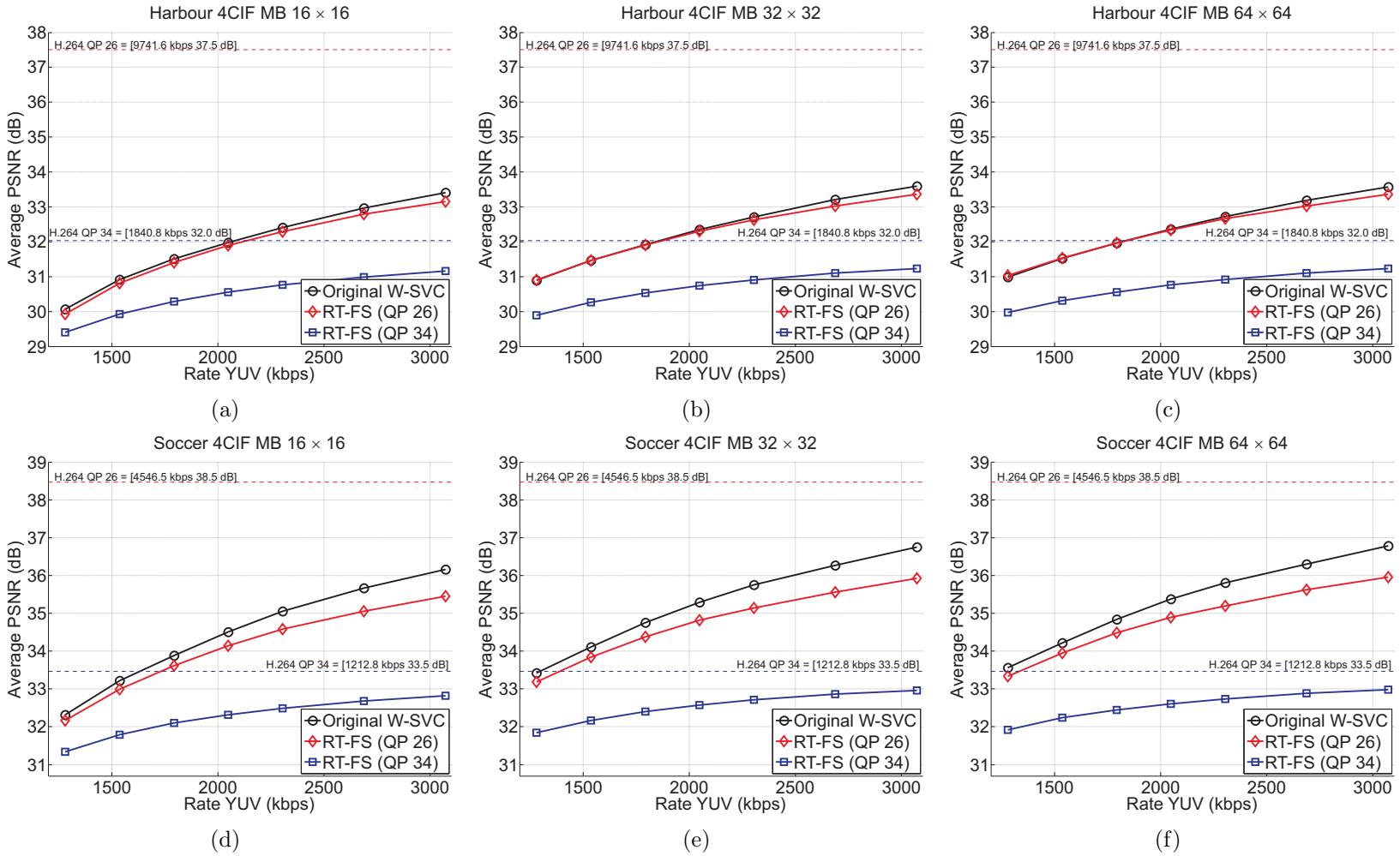


Figure 5.7.: Results of the W-SVC codec and RT-FS for: Harbour 4CIF (a) 16×16; (b) 32×32; and (c) 64×64; and Soccer 4CIF (d) 16×16; (e) 32×32; and (f) 64×64. Note that Fig. (a), (b) and (c), and Figs. (d), (e) and (f) are shown using the same axis.

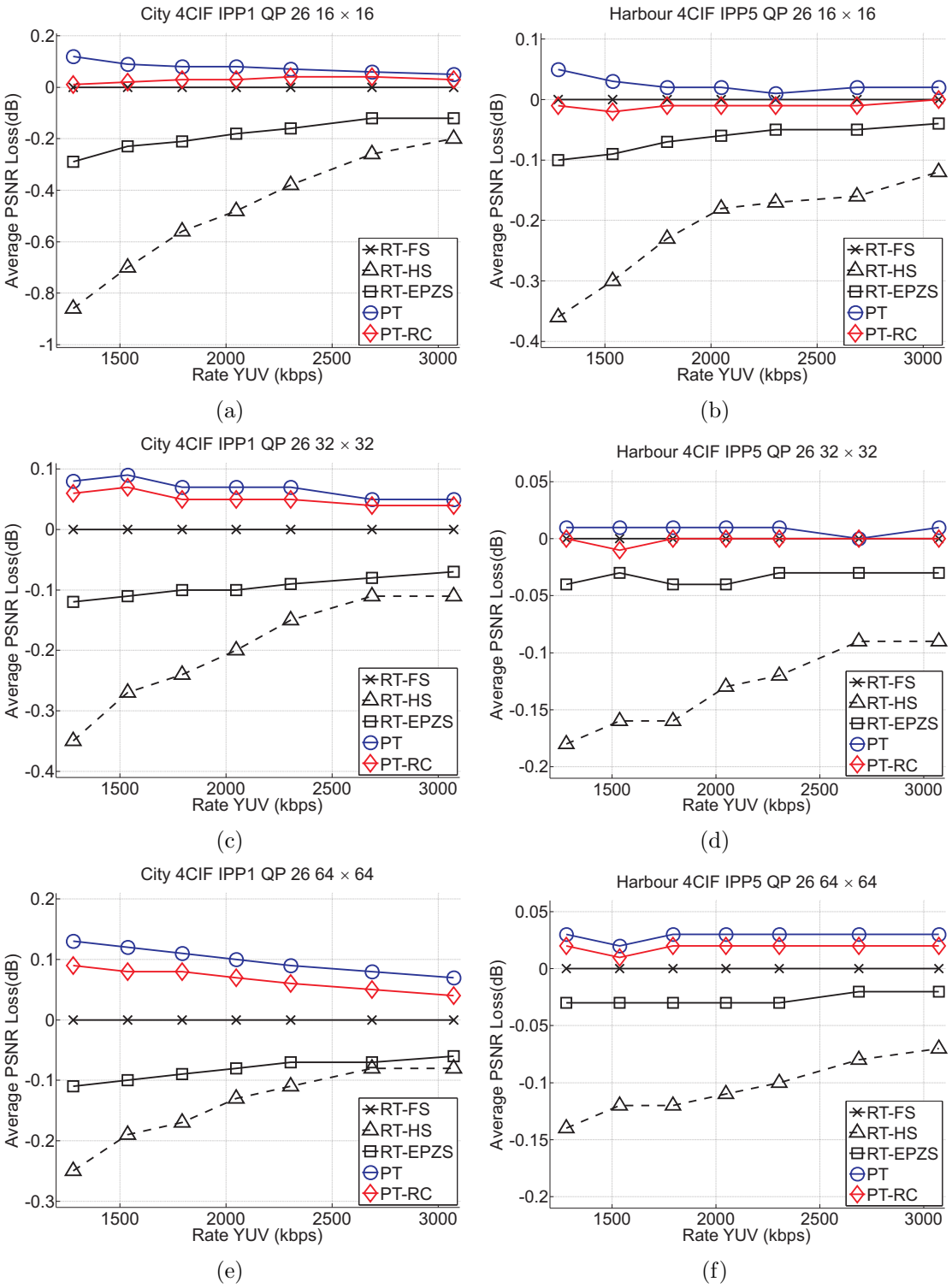


Figure 5.8.: Selected transcoder results for: City 4CIF *IPP1* QP 26 (a) 16×16; (c) 32×32 and (e) 64×64; and Harbour 4CIF *IPP5* QP 26 (b) 16×16; (d) 32×32; and (f) 64×64.

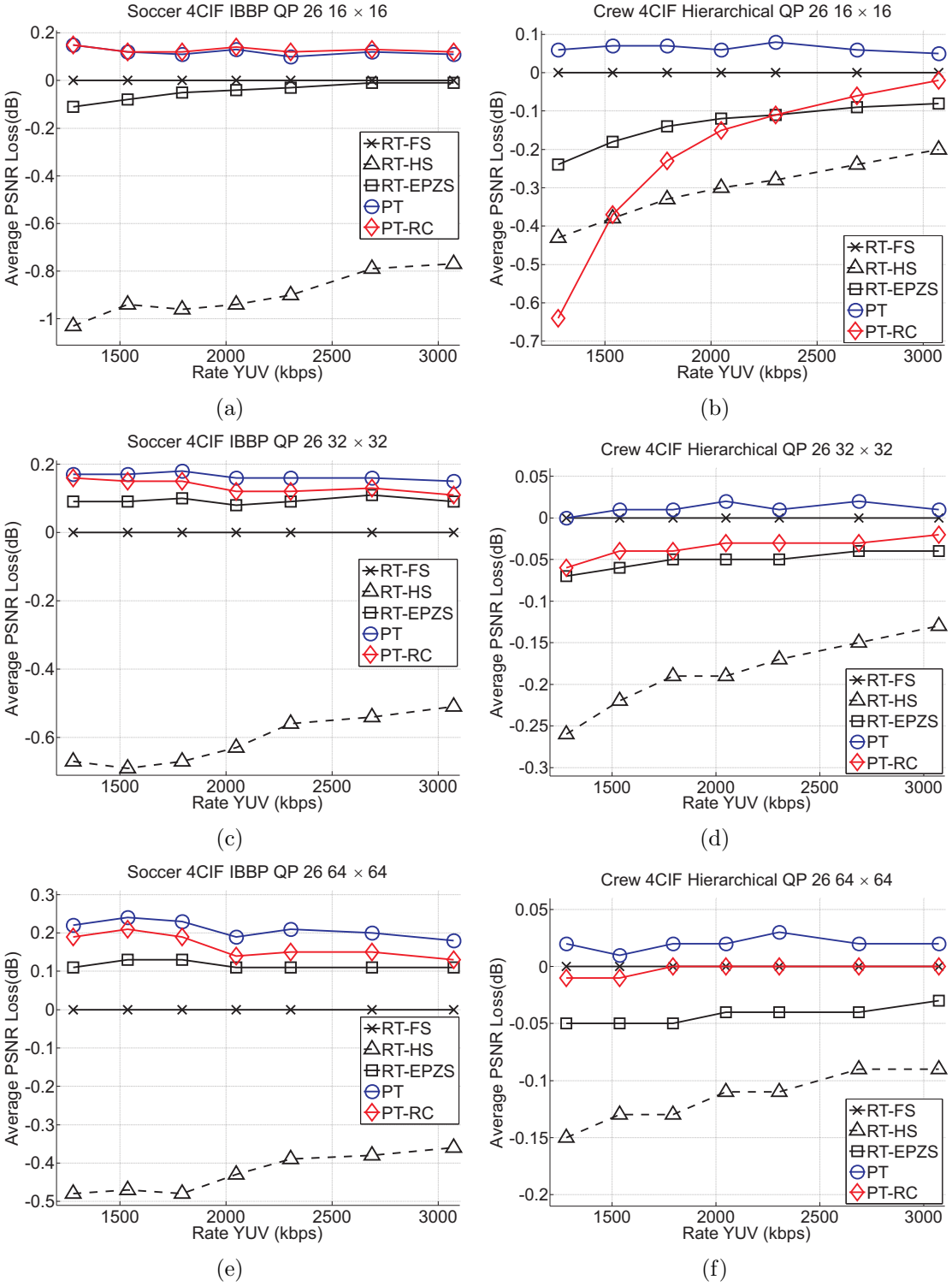


Figure 5.9.: Selected transcoder results for: Soccer 4CIF *IBBP* QP 26 (a) 16×16; (c) 32×32; and (e) 64 × 64; and Crew 4CIF *Hierarchical* QP 26 (b) 16 × 16; (d) 32 × 32; and (f) 64 × 64.

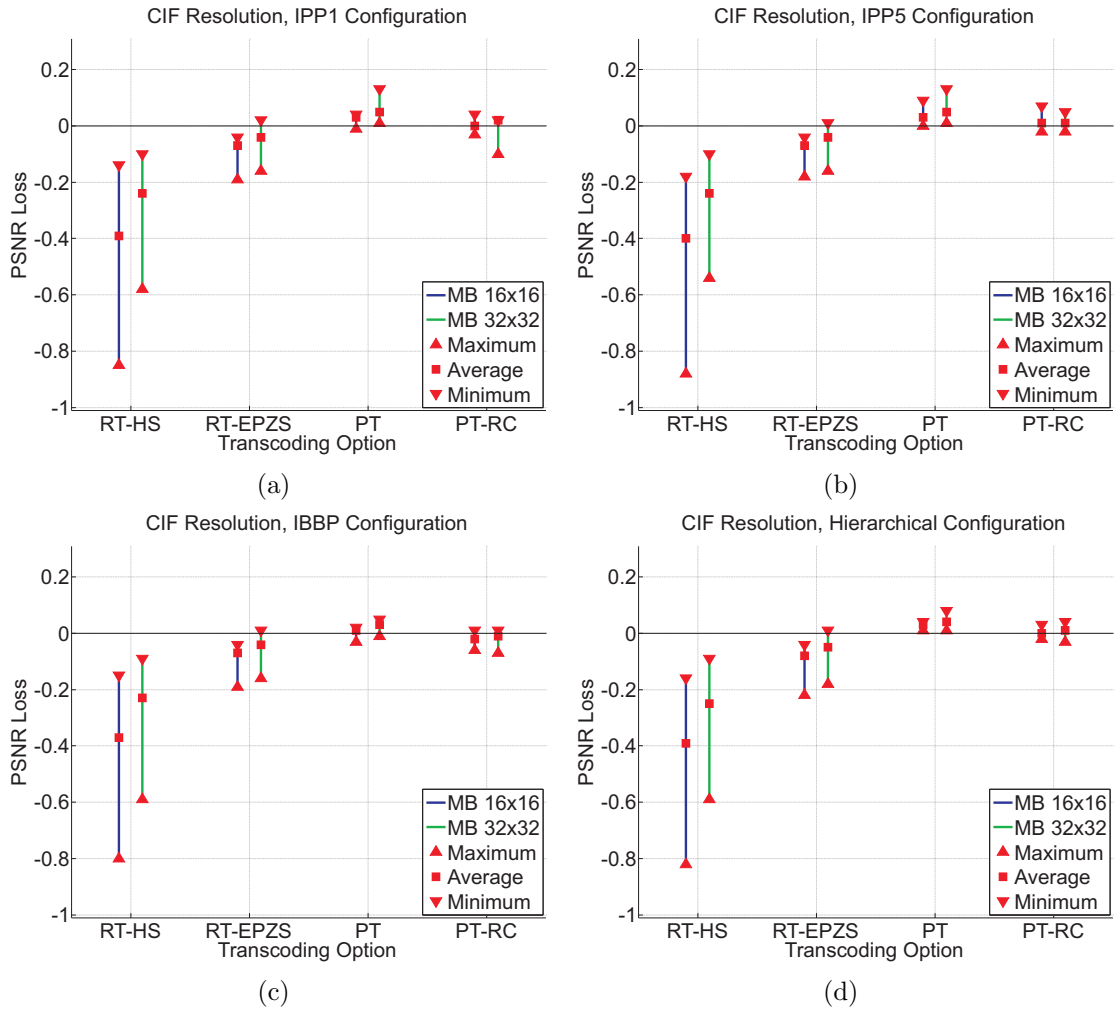


Figure 5.10.: Transcoder PSNR loss for CIF resolution for: (a) *IPP1*; (b) *IPP5*; (c) *IBBP*; and (d) *Hierarchical* coding configurations.

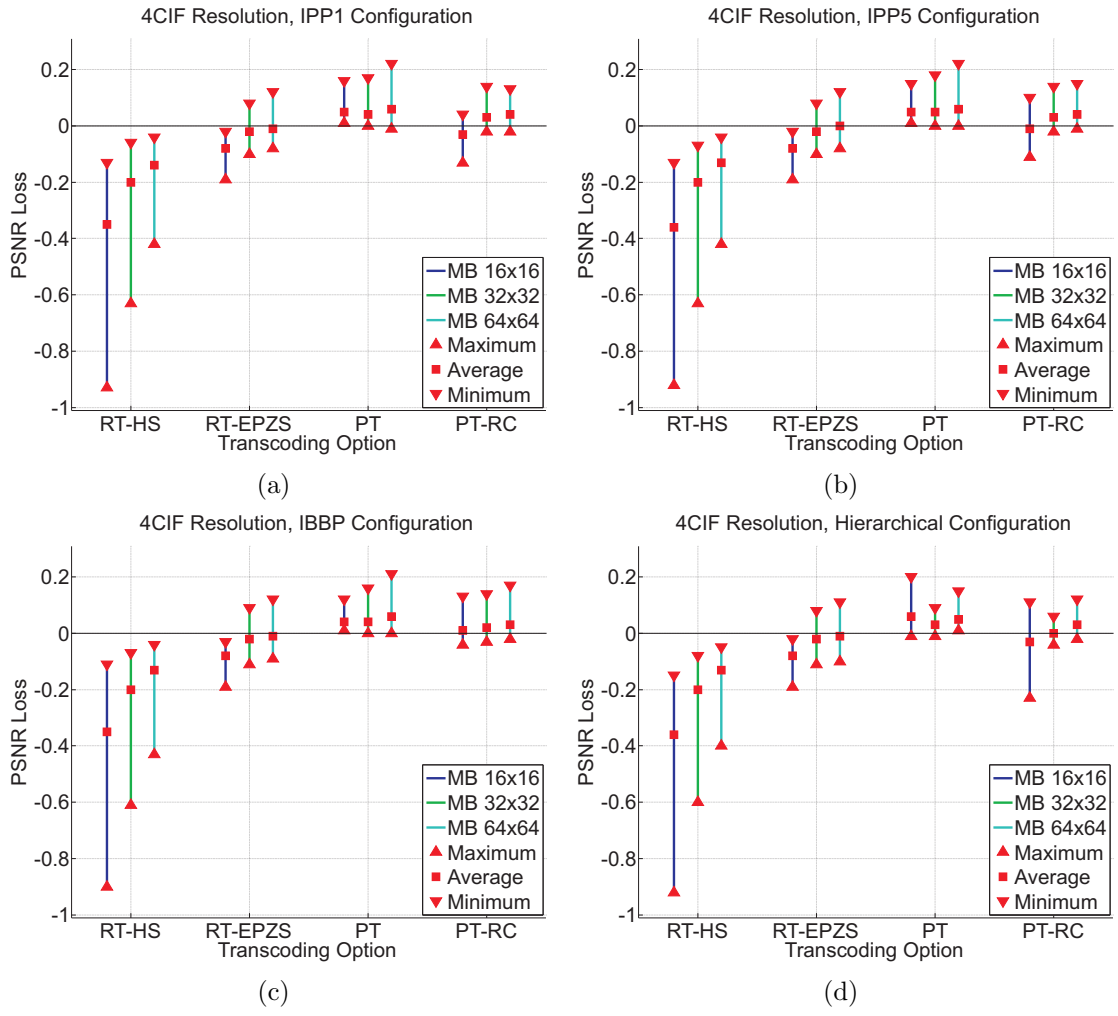


Figure 5.11.: Transcoder PSNR loss for 4CIF resolution for: (a) *IPP1*; (b) *IPP5*; (c) *IBBP*; and (d) *Hierarchical* coding configurations.

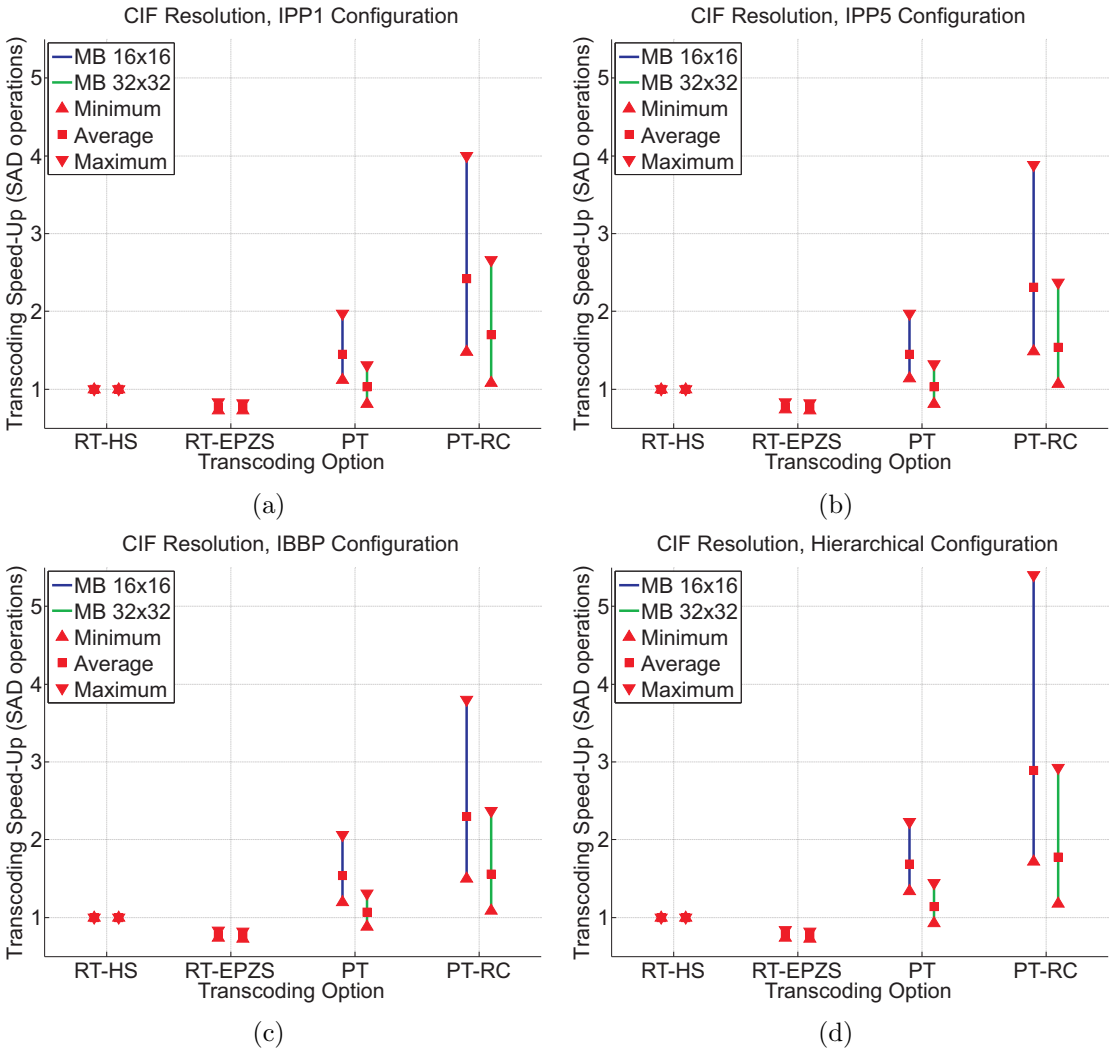


Figure 5.12.: Higher bound for transcoder speed-up for CIF resolution measured as the number of SAD Calculations for: (a) *IPP1*; (b) *IPP5*; (c) *IBBP*; and (d) *Hierarchical* coding configurations.

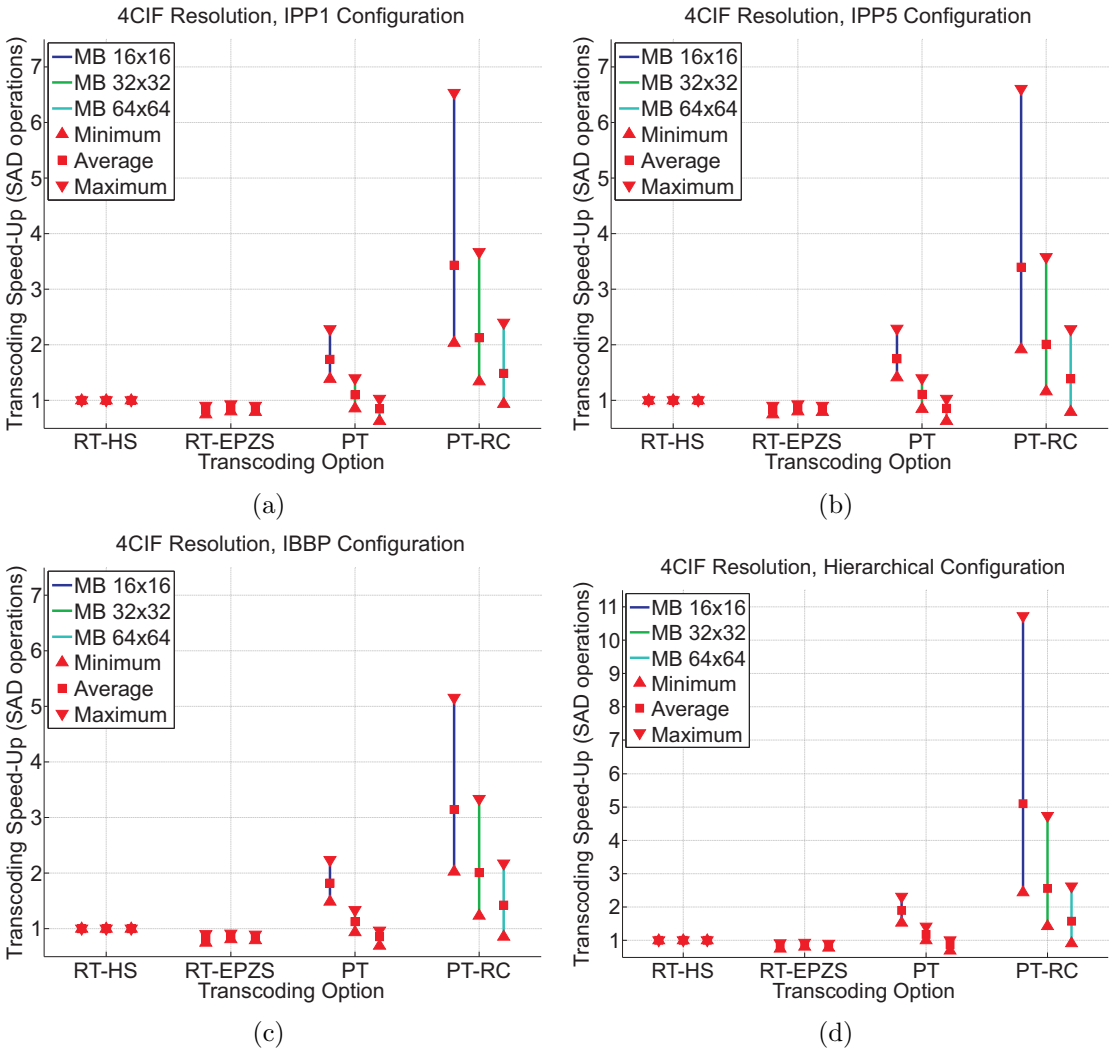


Figure 5.13.: Higher bound for transcoder speed-up for 4CIF resolution measured as the number of SAD Calculations for: (a) *IPP1*; (b) *IPP5*; (c) *IBBP*; and (d) *Hierarchical* coding configurations.

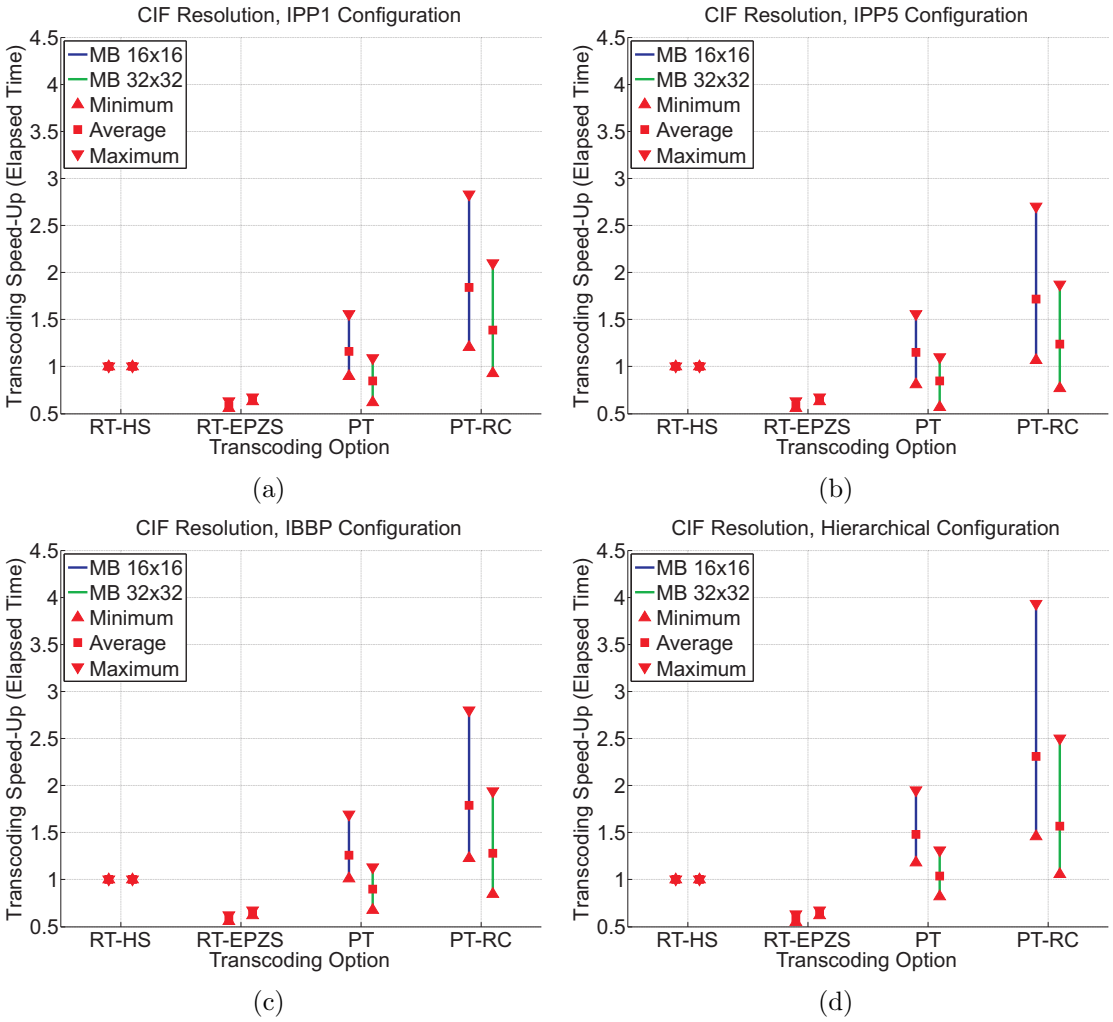


Figure 5.14.: Implemented transcoder speed-up for CIF resolution measured as the motion estimation Elapsed Running Time for: (a) *IPP1*; (b) *IPP5*; (c) *IBBP*; and (d) *Hierarchical* coding configurations.

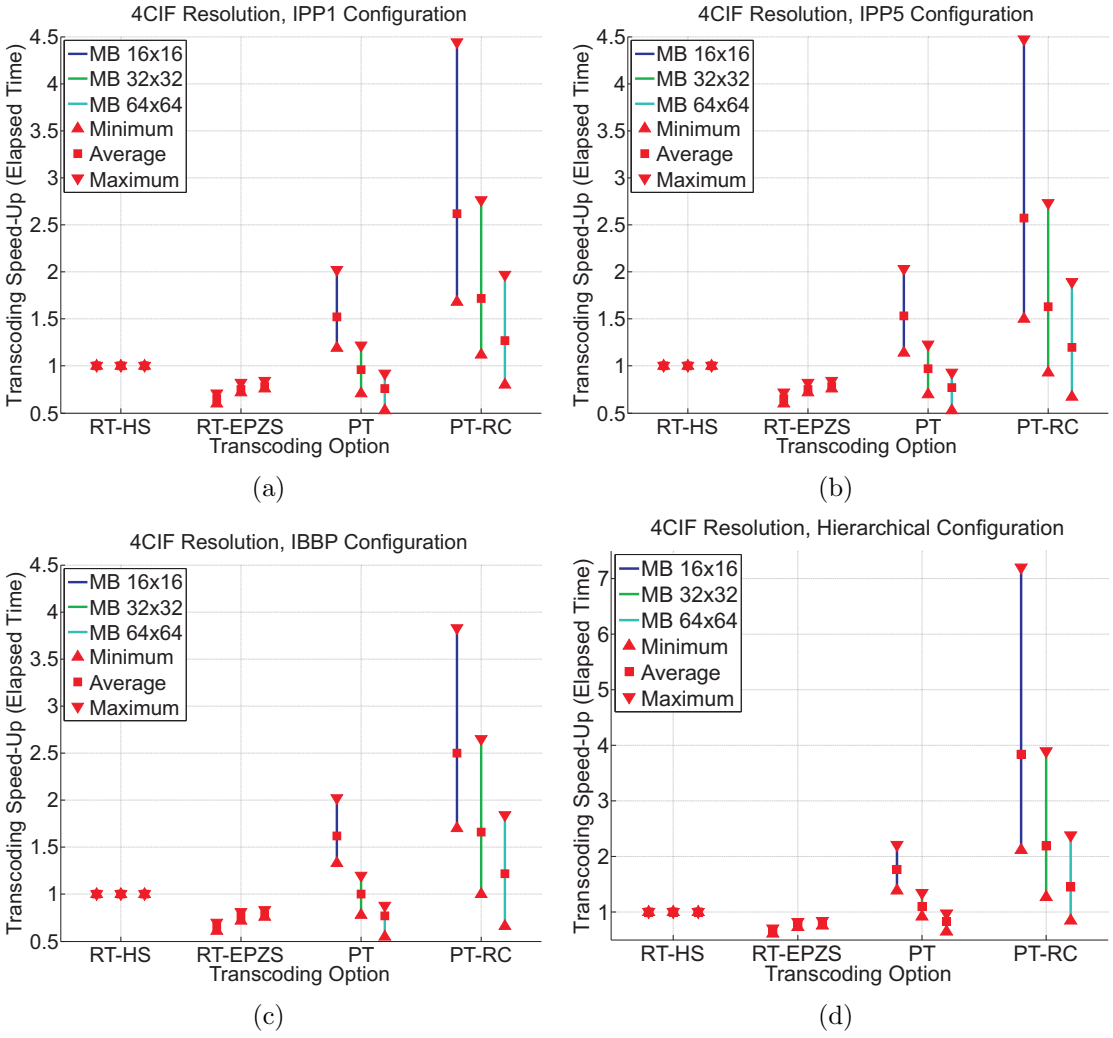


Figure 5.15.: Implemented transcoder speed-up for 4CIF resolution measured as the motion estimation Elapsed Running Time for: (a) *IPP1*; (b) *IPP5*; (c) *IBBP*; and (d) *Hierarchical* coding configurations.

Chapter 6.

Transcoding from H.264/AVC to HEVC

Even though the HEVC codec is not yet finalised, transcoding to and from the new codec will surely be needed when the codec becomes an international standard [37]. This chapter presents a H.264/AVC [62] to HEVC [58] transcoder, exploring several transcoding options to the HEVC codec, making use of different information found in the H.264/AVC bitstream, such as the motion vectors, partitioning and transform coefficients. All transcoding options presented in this chapter follow the approach of an heterogeneous transcoder [9], seen in Sec. 2.2.2.

A simple transcoder was implemented, based on the reuse of motion vectors, to verify the performance of this technique and identify potential issues that need to be addressed on the transcoder. Then, an efficient transcoder capable of exchanging rate-distortion performance for complexity is presented. Afterwards, a transcoder based on content modeling, in which the parameters of the transcoder are adapted to the current sequence being encoded, is presented. Finally, the experiments designed to evaluate these transcoding options, and the results of these experiments, are presented and discussed.

6.1. Transcoder Issues

As seen in Chapter 2, transcoding from H.264/AVC to the HEVC requires a change of format, characterizing it as an heterogeneous transcoder. Furthermore, seen on Chapter 4, one technique that is ubiquitous among heterogeneous transcoders is the motion vector reuse [21, 148]. The rationale of this technique is to use the motion vectors from

the source codec in the target codec, avoiding performing a costly motion estimation operation. Since motion estimation is the most time consuming operation in a video encoder, this technique alone can significantly reduce the transcoder complexity.

Reusing motion vectors from the H.264/AVC codec in the HEVC codec can be made easier than in the W-SVC transcoder seen in the previous chapters, as the coding configuration in the HEVC codec is not fixed. In fact, the coding configuration in the HEVC codec can be chosen to be as flexible as in the H.264/AVC codec, even though the codec favours two specific coding configurations (low delay and random access, seen in Sec. 3.3.3).

In this thesis, only the low delay configuration is studied, and the H.264/AVC configuration is selected to match the coding configuration on the HEVC. However, even though these steps were taken in order to promote the reuse of motion vectors, there are still two main challenges in applying this technique in the H.264/AVC to HEVC transcoder.

The first, and obvious one, is that the HEVC uses much larger blocks for motion estimation than the H.264/AVC (64×64 [86] against a fixed 16×16 largest unit). Thus, the motion information has to be combined and merged in order to be efficiently reused in the HEVC.

The second challenge is that, in the main anchor profiles for the HEVC, full motion estimation is not used, being dropped in favor of a fast motion estimation technique that can achieve a performance close to the full motion estimation, at a much lower complexity cost, especially for HD and larger content. In the HEVC encoder, an EPZS [132] algorithm is used, with some modifications. The reason for this change is that, even if complexity is not an issue, there are other techniques that could be used to further improve encoding performance while still using less complexity than full motion estimation. For example, instead of using full motion estimation, fast algorithms could be used with more reference frames, or the rate distortion optimization module could be allowed to test more quantization parameters - in both cases, the complexity added would be much lower than using full motion estimation, but the potential gain is higher.

Therefore, even though the motion vectors can be reused in the HEVC, this alone might not be sufficient to make an efficient transcoder. In order to study the impact of this technique, a H.264/AVC to HEVC transcoder based solely on the motion vector reuse technique was implemented. This transcoder is discussed in the next section.

6.2. A H.264/AVC to HEVC Transcoder Based on MV Reuse

In order to study the performance of an H.264/AVC to HEVC transcoder, a simple transcoder based on the cascaded pixel domain approach was implemented. This transcoder is mainly based on the motion vector reuse technique [21, 148] that was discussed in Chapter 4, and will be referred here as the MV Reuse transcoder.

In this section, and for the remainder of this chapter, the testing of a mode is defined as the assessment of the best way to encode that particular block using a given mode (i.e., deciding the parameters - motion vectors, transforms, etc...) and producing a rate-distortion cost, which will then be compared to the other tested modes to decide which mode will be used to encode that block. Similarly, the testing of a motion vector is defined as the evaluation of the cost of that motion vector, and comparing this cost with the motion vectors that were previously tested for that particular mode.

The workflow of the algorithm is the same for each coding unit (CU) in the HEVC, and it is based on two main ideas:

1) If any part of this CU was encoded in intra mode in the H.264/AVC, then all possible intra and inter modes are tested; otherwise, only the inter modes are tested.

2) For any inter partition unit (PU), all H.264/AVC motion vectors within the current PU are tested. The motion vectors are reused at integer-pixel level, without any further refinement at this level. Then, at half-pixel and quarter-pixel, the default HEVC search is applied (testing the eight neighbours at half-pixel level, then the eight neighbours at quarter-pixel level).

Note that this transcoder reuses the incoming motion vectors, but not the partitioning. All inter modes available in the HEVC are considered, including the Asymmetric Motion Partition, AMP [69] - for these partitions, the AMP speed-up setting is enabled. The remaining HEVC settings are the same as the low-delay configuration for $HM4.0rc1$ [86], including the fast mode decision flag (which is enabled). Therefore, this transcoder saves complexity only by avoiding the motion estimation (which is performed using a fast motion algorithm, EPZS [132]), and by not testing all intra modes.

6.2.1. Evaluating the MV Reuse Transcoder

The MV Reuse transcoder is evaluated against the trivial transcoder, which, for the HEVC, uses a fast EPZS algorithm for motion estimation. In this section, and through the remaining of this chapter, the trivial transcoder is referred as RT-EPZS, while the MV Reuse transcoder is referred as RT-MVR.

As the trivial transcoder, the reference software for the $HM4.0rc1$ encoder is used with the default settings [86], with a 64×64 LCU size and LCUs encoded in raster scan order. When encoding one LCU (largest coding unit - as seen in Sec. 3.3, the concept of the LCU in the HEVC is analogous to the concept of a macroblock in other codecs) in an inter frame, the encoder starts at depth 0 (i.e., 64×64), and it first tests the following PU sizes, computing a RD cost for each one: SKIP/MERGE, inter $2N \times 2N$, inter $2N \times N$ and inter $N \times 2N$. Before testing the asymmetric partitions (AMP), additional conditions are checked to speed-up the encoder [69]. In particular, the best PU size for the current depth and the best PU size for the previous depth are used in order to decide which AMP partitions will be tested. Thus, the four AMP partitions are not tested for every CU tested. The encoder also uses a conditional decision before testing the intra modes. In this case, the intra modes are not tested if the best mode tested so far has a residual equal to zero (i.e., if all transform coefficients for the residual are quantised to zero). Otherwise, the intra $2N \times 2N$ mode is tested, and the intra $N \times N$ is also tested if the CU size is 8×8. In the configuration used in this thesis, the inter $N \times N$ and the intra PCM modes are never tested. Afterwards, the encoder splits the CU, and it repeats this procedure for each CU at the next depth. The AMP encoding speed-up reduces the encoding time by 35%, on average [69], compared with testing all AMP modes for every CU.

For the H.264/AVC, a similar low-delay *IPP4* configuration and High Profile was used, using the reference software JM 14.2 [64]. The QPs used are $\{37, 32, 27, 22\}$ in both codecs, and the transcoder uses the same QP as the H.264/AVC bitstream. The results for RT-MVR are shown in Fig. 6.1, along with the results for RT-EPZS. The sequences used are part of the HEVC test sequences (see Appendix C for more details).

Observing the results for RT-EPZS in Fig. 6.1, it can be seen that, as expected, there is a loss of quality in transcoding, since the transcoder does not operate on the original sequences ($-1.52dB$, $-1.72dB$, $-1.18dB$ and -2.18, on average, for Basketball Drill, BQMall, Vidyo1 and RaceHorses sequences, respectively). However, there is also

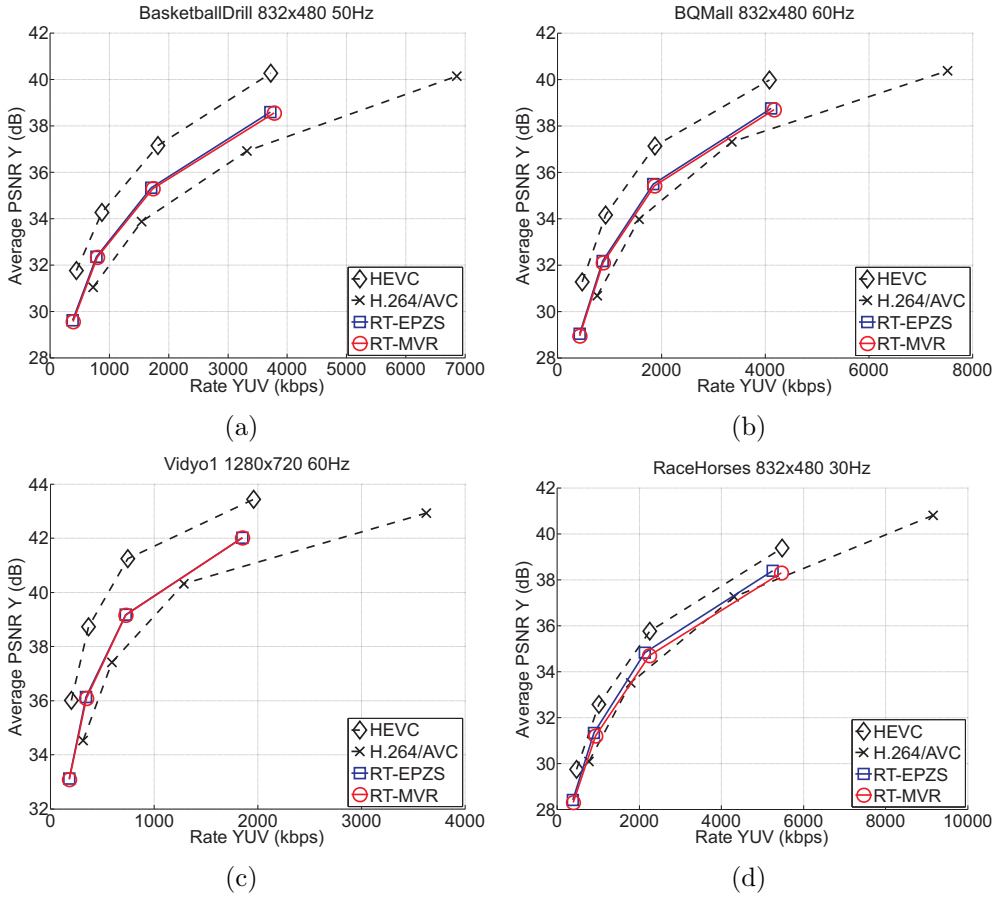


Figure 6.1.: Results for RT-MVR and RT-EPZS for: (a) Basketball Drill; (b) BQMall; (c) Vidyo1; and (d) Race Horses.

Table 6.1.: Results for RT-MVR, shown as the BD-bitrate and speed-up, for: Basketball Drill, BQMall, Vidyo1, and Race Horses. The BD-Rate is shown as a percentage, using RT-EPZS as anchor, and the Speed-Up is relative to RT-EPZS.

	Basketball		BQMall		Vidyo1		RaceHorses	
Method	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate
RT-EPZS	1.00	0.0	1.00	0.0	1.00	0.0	1.00	0.0
RT-MVR	1.40	2.78	1.56	3.16	1.12	0.74	1.77	7.71

a considerable bitrate reduction (48%, 45%, 44% and 49%, on average, for Basketball Drill, BQMall, Vidyo1 and RaceHorses sequences, respectively). The figure also shows the performance of the HEVC codec working on the original sequences (labeled as HEVC in the figures), which showcases the superior rate distortion performance of the HEVC codec, compared to the H.264/AVC. The performance for the HEVC working on other sequences has been extensively evaluated elsewhere [75].

The complexity results can be found in Table 6.1. As opposed to Chapter 5, the complexity is measured here as the total running time, and shown as the relative speed-up, relative to RT-EPZS. In the HEVC, there are steps external to the motion estimation that are relevant to the encoder complexity. For instance, in the $HM4.0rc1$, when testing a mode for each CU, the transform unit is computed for that mode, along with entropy coding, in order to compute the actual rate and distortion for that mode (which will then be compared to other modes, in rate-distortion sense). Also, other operations, such as the interpolation, are applied on-the-fly, instead of being applied to the whole frame prior to motion estimation, and thus they have a different impact on the complexity if different modes are tested. For the same reasons, the SAD computations are not used because motion estimation accounts only for a small part of the mode decision complexity.

In addition to the complexity, the table shows the Bjøntegaard Delta bitrate (BD-bitrate) [20], which is the average bitrate difference relative to an anchor, in percentage. This is a metric that has become very popular recently, as it gives a better understanding when comparing rate distortion curves that are close to each other, which is the case for the curves shown in Fig. 6.1. The calculations on how to compute the BD-bitrate can be found in Appendix D.2.2. The speed-up shown in the table is the average between the four QPs used. Also, note that the trivial transcoder uses fast motion estimation and fast mode decision.

Analysing the PSNR results for the MV Reuse transcoder (RT-MVR), shown in Fig. 6.1, it can be seen that the PSNR of this transcoder is very close to the PSNR of the trivial transcoder (RT-EPZS). In fact, the highest PSNR loss of RT-MVR, compared to RT-EPZS, is -0.13 dB (for RaceHorses sequences at 933 kbps, shown in Fig. 6.1(d)). In terms of BD-bitrate, the maximum loss is 7.71%, for the RaceHorses sequence, and considerably lower for the other sequences (as seen in Table 6.1). This rate-distortion performance is close to the trivial transcoder, demonstrating a high correlation between the H.264/AVC and the HEVC motion vectors. However, the speed-up figures of this transcoder are very low, ranging from 1.12 (for Vidyo1 sequence) to 1.77 (for RaceHorses sequences), even though four reference frames are used. This indicates that this technique alone is not sufficient for an efficient transcoder, and other techniques to reduce the number of modes tested in the HEVC should be studied.

6.3. A H.264/AVC to HEVC Transcoder Based on MV Variance Distance

As seen in the previous section, the rate-distortion performance of the MV Reuse transcoder is good, but its complexity savings are quite low. For this reason, we proposed a transcoder [1] that aims primarily to reduce the complexity of the MV Reuse transcoder. The main idea is to avoid testing less likely CUs, so that a larger amount of computation can be saved.

This transcoder is based on a similar idea to the H.264/AVC to W-SVC transcoder, especially the MV similarity metric seen in Sec. 5.3.2. The rationale is to use this new similarity metric, the MV Variance Distance, and, according to this metric, make the decision of how to test a particular CU.

6.3.1. MV Variance Distance

The MV Variance Distance metric produces a value $\upsilon \geq 0$ for each CU that can be tested in the HEVC. This metric is based on the variance of the H.264/AVC motion vectors, and it is computed as:

$$\upsilon = \sqrt{\left(\sigma_x^2\right)^2 + \left(\sigma_y^2\right)^2} \tag{6.1}$$

where σ_x^2 and σ_y^2 are the variances of each component of the H.264/AVC motion vectors within the CU. If the motion vectors do not have the same reference frame, or if a part of this CU was encoded using an intra mode, then the metric does not produce a value. Before the metric is computed, the motion vectors are propagated to the 4×4 blocks (i.e., the minimum size in the H.264/AVC), and then the variance is calculated. This way, the motion vectors are weighted according to the area that they represent. This is shown in Fig. 6.2.

The idea of using this metric is that, if a large area has a low value υ, it means that all motion vectors in this area are similar, and thus it is more likely that this partition will be encoded using a larger CU in the HEVC (or, at least, that the loss caused by encoding this region as a larger CU will be smaller), as it is more likely that a single

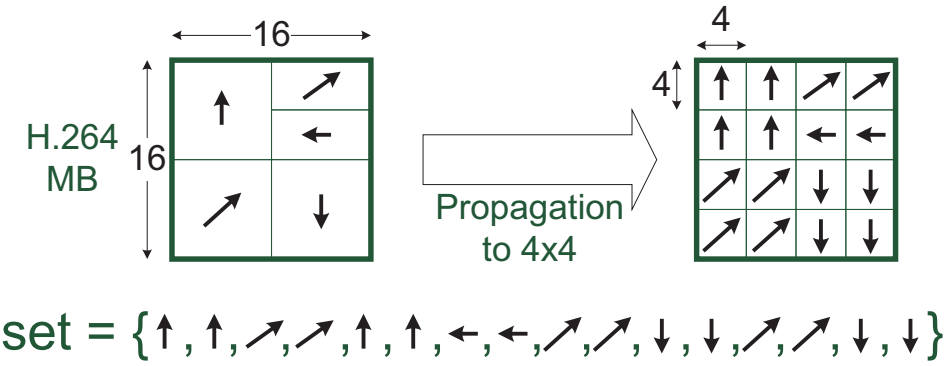


Figure 6.2.: Computing the MV Variance Distance metric for a 16×16 CU. First, the H.264/AVC motion vectors are propagated to the 4×4 level, and then the variance is computed on this set of motion vectors.

motion vector will accurately predict the whole CU. On the other hand, if the same area has a high value υ, then the motion vectors within this area are very different, and thus it is less likely that this block will be encoded using a large CU in the HEVC (meaning it is more likely that it will be split). This way, it is possible to combine the information for different H.264/AVC macroblocks and make a decision for a large block in the HEVC codec.

Two thresholds are used to decide how a particular CU will be tested, namely T_{low} and T_{high}, which defines three different regions R_1 ($\upsilon \leq T_{low}$), R_2 ($T_{low} < \upsilon \leq T_{high}$) and R_3 ($\upsilon > T_{high}$). These thresholds and the defined regions are shown in Fig. 6.3.

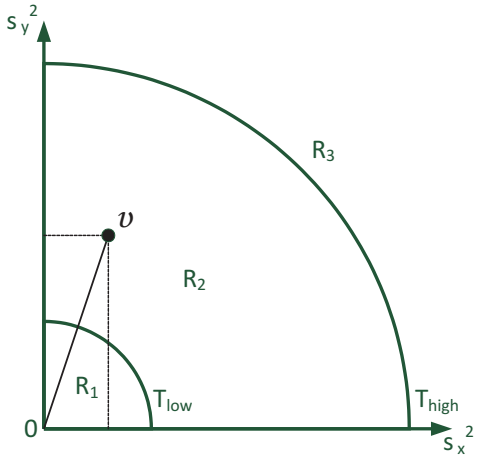


Figure 6.3.: Visualization of the MV Variance Distance metric. In the example, it is shown $T_{low} < \upsilon < T_{high}$, thus the modes associated with region R_2 are tested for this CU.

6.3.2. The Transcoder Algorithm

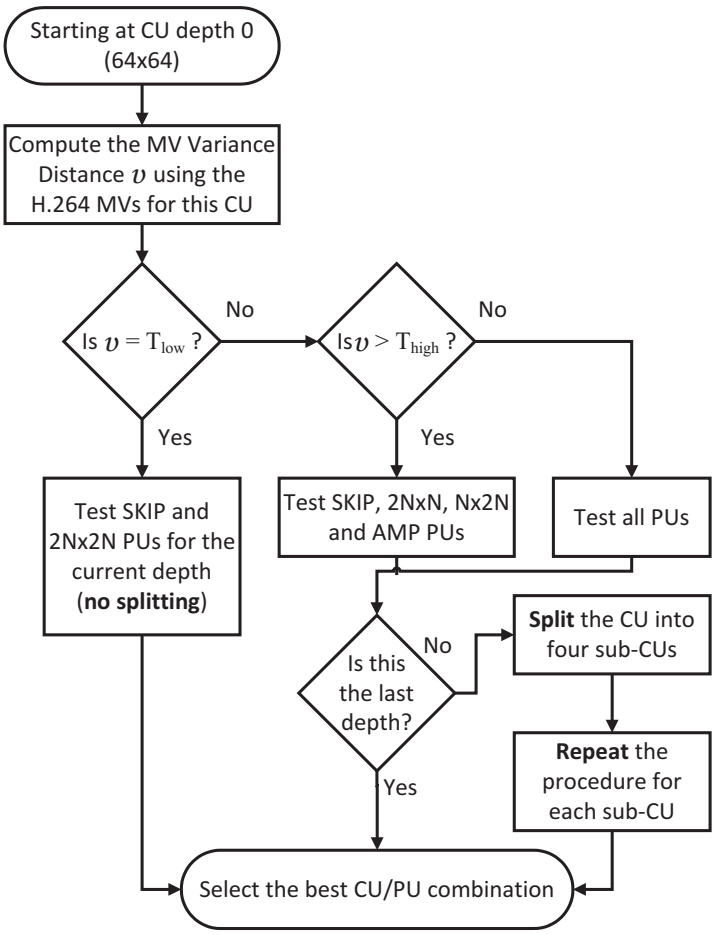


Figure 6.4.: Transcoder algorithm for a given CU. Note that the algorithm is recursive, starting at the LCU at depth 0, and repeating itself for the sub-CUs if the current CU is split.

The transcoder algorithm works independently for each CU, regardless of the CU size, and it is shown in Fig. 6.4. The possible prediction units (PUs) that can be tested are divided in four groups: (i) SKIP; (ii) inter $2N \times 2N$; (iii) all remaining inter modes ($2N \times N$, $N \times 2N$, the AMP modes, and $N \times N$); and (iv) the intra modes ($2N \times 2N$, $N \times N$ and PCM). This division can be seen in Fig. 6.5. In addition, the transcoder can decide if the CU will be split or not (if so, the CU is split in four sub-CUs, as usual). Then, depending on the value of the MV Variance Distance υ for this particular CU, four different settings can be used:

1) if the CU is considered similar (i.e., if $\upsilon \leq T_{low}$), then only the PU groups (i) and (ii) will be tested and the CU will not be split;

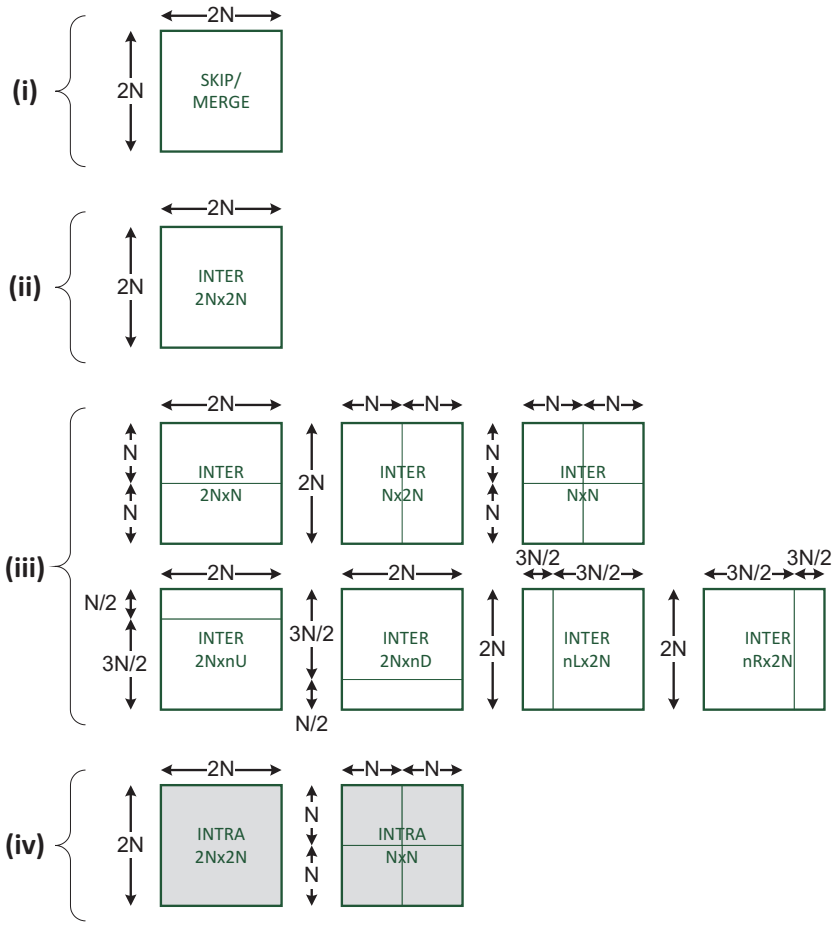


Figure 6.5.: Groups of the possible coding modes for a CU.

2) if no information can be gathered for this particular CU (i.e., if $T_{low} < \upsilon \leq T_{high}$), then the PU groups (i), (ii) and (iii) (i.e., all inter modes) will be tested and the CU will be split; and

3) if the CU is considered as dissimilar (i.e., if $\upsilon > T_{high}$), then only the PU groups (i) and (iii) will be tested, and the CU will be split.

4) if the value υ cannot be computed (i.e., if one H.264/AVC partition within the CU was encoded as intra, or if the reference frames do not match), then all PU groups are tested and the CU is split.

The algorithm starts from the largest CU size (64×64), computing the MV Variance Distance υ for that CU. Then, according to the υ value for the CU, the transcoder tests only the PUs for the groups indicated by the aforementioned rules. If the rules state that the CU should not be split (i.e., if $\upsilon \leq T_{low}$), then the transcoder selects the best mode, according to the modes tested, and proceeds to the next CU. If the CU is split,

the algorithm is applied in the same manner to each of the four sub-CUs, computing a new similarity value v for the sub-CUs, until the final possible depth is reached. Note that the SKIP mode is always tested, even if the CU is considered dissimilar, as the complexity to test the SKIP is small, compared to the other modes. Furthermore, as in the MV Reuse transcoder shown in Sec. 6.2, the intra modes are only tested if some part of this CU was coded as intra in the H.264/AVC bitstream, and the default fast mode decision algorithms in the HEVC are still applied (for testing the intra modes and the AMP modes).

6.3.3. MV Reuse and Refinement

In addition to using the similarity metric to decide which CU sizes are tested, the H.264/AVC motion vectors are also reused in the transcoder. By default, the HEVC uses a EPZS search [132] as the fast motion algorithm, that is similar, in spirit, to the one shown on Appendix A.2 (the search in the HEVC EPZS implementation uses less predictors, making it faster). Otherwise, it works in the same way, by creating a small motion vector candidate list (initially populated by the $(0, 0)$ and the median motion vector), and then, according to a criterion, performing an iterative search.

There are two main ways by which the MVs are reused in the transcoder: (i) for a given PU, the H.264/AVC MV that covers the largest area within that CU is also tested as a motion vector candidate (along with default predictors), then the default HEVC fast motion search is applied; and (ii) all H.264/AVC MVs are considered for integer ME, and no further refinement is performed at the integer pixel level. In both cases, the default HEVC sub-pixel search is applied.

6.3.4. Using MV Scaling to compute the similarity metric

The MV Variance Distance can only produce a value if all motion vectors being considered at a given point share the same reference frame. However, in order to overcome this limitation and further reduce the complexity, the motion vectors can be scaled to the same reference frame before the similarity value is computed. Here, a simple scaling formula is used:

Table 6.2.: BD-bitrate and speed-up results for PT-MVVD, for: Basketball Drill, BQMall, Vidyo1 and Race Horses. The BD-Rate is shown as a percentage, using RT-EPZS as anchor, and the Speed-Up is relative to RT-EPZS.

	Basketball		BQMall		Vidyo1		RaceHorses	
Method	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate
RT-EPZS	1.00	0.0	1.00	0.0	1.00	0.0	1.00	0.0
RT-MVR	1.40	2.78	1.56	3.16	1.12	0.74	1.77	7.71
PT-MVVD (i)	1.81	2.57	1.88	4.25	2.90	2.28	1.48	2.57
PT-MVVD (ii)	2.08	4.60	2.10	6.81	3.42	7.30	1.56	3.25
PT-MVVD (iii)	2.58	6.34	2.31	7.80	3.49	7.66	1.78	4.71
PT-MVVD (iv)	3.69	7.58	3.50	9.55	4.13	8.24	3.05	10.92

$$mv_{n\to n-\beta} = \left(\frac{\beta}{\alpha}\right) \cdot mv_{n\to n-\alpha} \tag{6.2}$$

where n is the current frame, $n-\alpha$ is the reference frame used by the H.264/AVC motion vector and $n-\beta$ is the target reference frame. If the scaling is necessary, then all motion vectors are scaled to the frame which is closest to the current frame. The scaling is only used in order to compute the MV Variance Distance, not to reuse the motion vectors.

6.3.5. Experimental Results

The test settings in this section are the same as those in Sec. 6.2.1. Here, several combinations of the methods discussed in previous sections are evaluated, each one targeting a different complexity level. The exact parameters can be seen in Table 6.5, where the proposed transcoder using the MV Variance Distance is referred as PT-MVVD. The transcoder options are named in descending order of complexity. The results, in terms of BD-Rate and speed-up, are shown in Table 6.2, and as rate-distortion curves in Fig. 6.6.

The effect of varying the thresholds T_{low} is shown in Table 6.3, and the effect for the threshold T_{high} is shown in Table 6.4. For this experiment, the first 100 frames of each sequence were used, with QP 22 and four reference frames. As expected, by varying the value of T_{low} one can affect the complexity of the proposed transcoder. The higher the value, the lower the complexity of the proposed transcoder (as more CUs are considered

Table 6.3.: Effect of the threshold T_{low} on the proposed transcoder. For all sequences, the results shown are for the first 100 frames with QP 22, compared to RT-EPZS. The threshold used in the remainder of Sec. 6.3.5 is highlighted.

		Basketball Drill			BQMall			RaceHorses		
Method		Δ Bitrate	Δ PSNR	Speed-Up	Δ Bitrate	Δ PSNR	Speed-Up	Δ Bitrate	Δ PSNR	Speed-Up
RT-EPZS		0.0	0.0	1.00	0.0	0.0	1.00	0.0	0.0	1.00
PT-MVVD	$T_{low} = 0$	1.51	−0.06	2.24	1.66	−0.09	2.36	4.40	−0.12	2.43
	$\boldsymbol{T_{low} = 1}$	**3.71**	**−0.18**	**3.36**	**4.16**	**−0.20**	**2.89**	**4.78**	**−0.16**	**2.70**
	$T_{low} = 2$	4.25	−0.19	3.54	5.86	−0.22	3.11	5.18	−0.17	2.81
	$T_{low} = 4$	4.98	−0.21	3.66	8.61	−0.25	3.33	6.11	−0.19	2.96
	$T_{low} = 6$	5.49	−0.21	3.72	10.64	−0.27	3.47	7.04	−0.20	3.04
	$T_{low} = 10$	6.62	−0.22	3.82	14.15	−0.30	3.68	8.86	−0.23	3.20
	$T_{low} = 20$	8.41	−0.22	4.00	19.97	−0.35	4.05	12.45	−0.27	3.50
	$T_{low} = 40$	10.20	−0.25	4.21	29.01	−0.40	4.56	16.83	−0.31	3.86

Table 6.4.: Effect of the threshold T_{high} on the proposed transcoder. For all sequences, the results shown are for the first 100 frames with QP 22, compared to RT-EPZS. Also, for all results shown here, it was used $T_{low} = 1$. The threshold used in the remainder of Sec. 6.3.5 is highlighted.

		Basketball Drill			BQMall			RaceHorses		
Method		Δ Bitrate	Δ PSNR	Speed-Up	Δ Bitrate	Δ PSNR	Speed-Up	Δ Bitrate	Δ PSNR	Speed-Up
RT-EPZS		0.0	0.0	1.00	0.0	0.0	1.00	0.0	0.0	1.00
PT-MVVD	$T_{high} = notused$	3.71	−0.18	3.36	4.16	−0.20	2.89	4.78	−0.16	2.70
	$T_{high} = 1000$	3.86	−0.19	3.45	4.30	−0.20	2.94	4.88	−0.16	2.74
	$T_{high} = 500$	3.79	−0.19	3.51	4.47	−0.20	2.99	4.92	−0.16	2.80
	$T_{high} = 200$	3.99	−0.18	3.79	4.62	−0.20	3.14	5.07	−0.16	2.93
	$\boldsymbol{T_{high} = 100}$	**4.41**	**−0.19**	**4.07**	**4.82**	**−0.20**	**3.37**	**5.22**	**−0.17**	**3.05**
	$T_{high} = 50$	4.49	−0.20	4.29	5.13	−0.20	3.62	5.33	−0.17	3.21
	$T_{high} = 10$	4.69	−0.19	4.76	5.55	−0.21	4.37	5.67	−0.17	3.79

Table 6.5.: Parameters used for different options for PT-MVVD.

Method	T_{low}	T_{high}	Scaling MVs	Refinement
PT-MVVD (i)	1	Not Used	No	Yes
PT-MVVD (ii)	1	Not Used	Yes	Yes
PT-MVVD (iii)	1	100	Yes	Yes
PT-MVVD (iv)	1	100	Yes	No

similar), and the higher the impact on the RD performance (as less CUs are tested). It can be seen in Table 6.3 that using a value of T_{low} higher than 1 generally offers small complexity savings (compared to using $T_{low} = 1$) but may have a higher impact on rate-distortion performance. This is, however, dependent from sequence to sequence. Here, it was chosen to use $T_{low} = 1$ in order to improve the transcoder performance. The effect of the threshold T_{high} is shown in Table 6.4. Here, it was chosen to use $T_{high} = 100$. It can be seen in both tables that the effect of these thresholds is sequence dependent, and that it is very difficult to choose an optimal threshold that works for all cases. This will be discussed more thoroughly in the next sections.

It can be seen from Fig. 6.6 that, for low bit-rates, all four proposed transcoders have a very good rate-distortion performance. For medium and higher bitrates, options (i) and (ii) still perform well, but options (iii) and (iv) incur a higher loss.

Fig. 6.7 shows the speed-up versus the bitrate loss for various sequences. The speed-up shown is the average for the four QPs used. Again, note that the speed-up results of the transcoder are shown comparing to the trivial transcoder, RT-EPZS, which uses fast motion estimation and fast mode decision.

The proposed transcoder techniques are up to 4 times faster than RT-EPZS, and up to 3.7 times faster than the transcoder based on MV Reuse, RT-MVR. The only case where RT-MVR is faster than the proposed transcoder (PT-MVVD) is for Race Horses sequence, where it is faster than options (i) and (ii) of PT-MVVD (18% and 13%, respectively, as seen in Table 6.2). However, the rate-distortion performance of options (i) and (ii) is much better in this case. The options (iii) and (iv) are always faster than RT-MVR. This complexity reduction is achieved at a loss of rate-distortion performance, however, even for the same levels of rate-distortion performance, PT-MVVD (option (i)) is up to 2.6 faster than the RT-MVR (for Vidyo1 sequence).

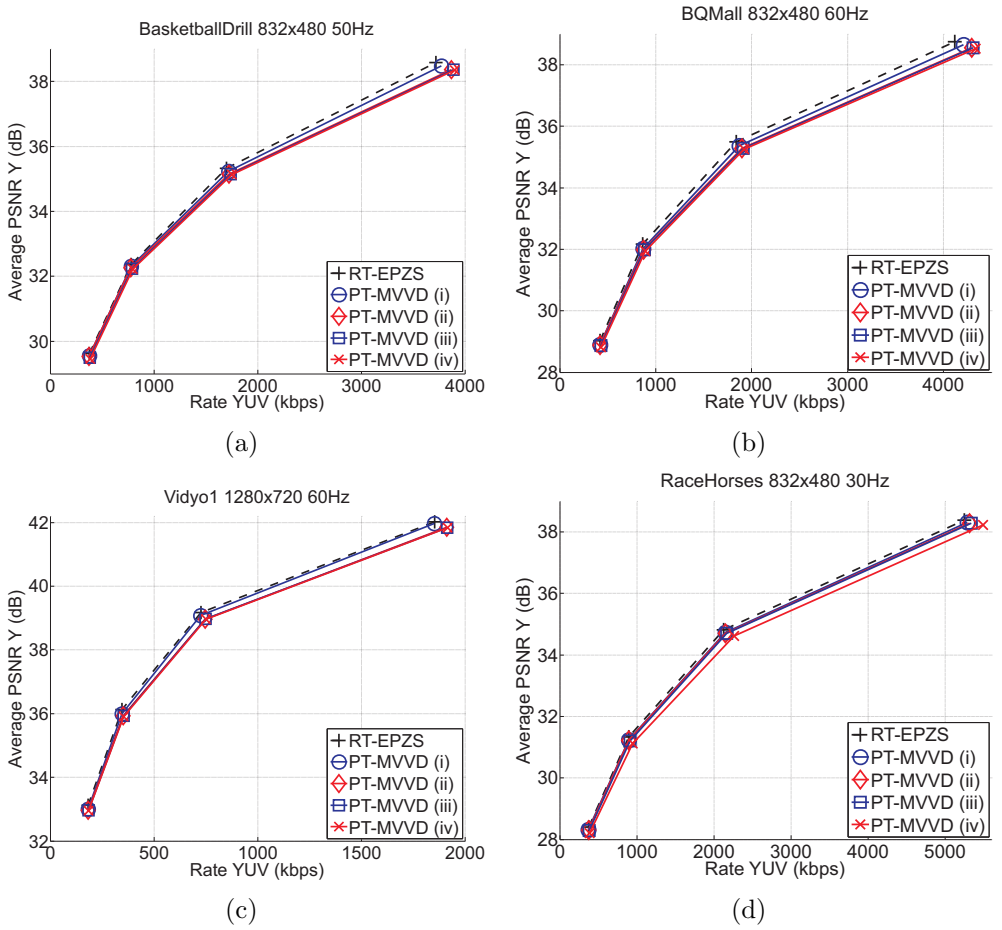


Figure 6.6.: Results for PT-MVVD for: (a) Basketball Drill; (b) BQMal; (c) Vidyo1; and (d) RaceHorses.

6.4. A H.264/AVC to HEVC Transcoder Based on Content Modeling

The transcoder presented in the previous section offers a good rate-distortion and complexity performance, and offer an interesting insight on a way to tackle the H.264/AVC to HEVC transcoder. If the best mode to encode a given CU could be accurately predicted from information in the H.264/AVC bitstream, then a large amount of computation could be saved, with virtually no quality loss. Even if the best mode cannot be predicted, if the less likely modes could be ruled out, then the transcoding could still be made faster with a small penalty in rate distortion performance.

In the proposed transcoder based on MV Variance Distance, however, the choice of the thresholds is still a main issue. First, the thresholds are used regardless of the depth

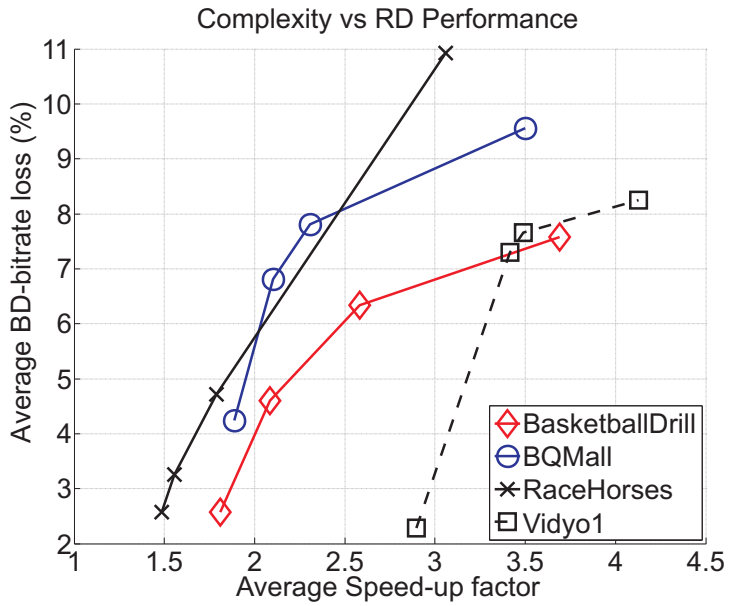


Figure 6.7.: Complexity and RD performance results for PT-MVVD for different sequences. For each sequence, each point in the curve refer to the results using a particular option for PT-MVVD (from (i) to (iv)).

of the CU, which gives the same tolerance to decide the split for a 64×64 partition and a 16×16 partition. Second, and most important, the same thresholds are used for different sequences, regardless of the content of the sequences or the quantization parameters used to encode it. Finally, only the MV Variance distance is used in the decision loop - other information from the H.264/AVC bitstream are ignored.

In order to overcome these issues, a new transcoder was developed and implemented. This transcoder also uses features computed from information in the H.264/AVC bitstream (such as the MV Variance Distance) to decide how to test a given CU. However, the thresholds used in this transcoder are computed adaptively for the current sequence being transcoded. Also, the features are only used to decide the modes for the 64×64 and 32×32 CUs - for the 16×16 and 8×8 CUs, the mode used in the H.264/AVC bitstream is used in the decision process instead. Four features are investigated, and also a combination of these features using a simple linear classifier is explored.

Other works involving mode mapping using machine learning algorithms have been proposed [42, 41, 52, 85]. However, all these works attempt to build a single, generalized, mapping that can be used for transcoding any bitstream. Also, most of these works use a large number of features [42, 41, 52], and use a machine learning algorithm whose training is complex, not being suitable for use in the proposed transcoder.

A relevant approach to the proposed training solution, although much simpler and reported in a different context, was proposed to reduce computational complexity of the H.264/AVC encoder mode selection [34]. In this work, when encoding a given frame, the encoder would test all modes for a small number of randomly placed macroblocks (the macroblocks are thus not encoded in the traditional raster scan order). When testing these macroblocks, the mode decision choices are stored, and the dominant modes used to encode these macroblocks are stored. Then, the remaining macroblocks are encoded with a sub-optimal decision, using the dominant modes only.

The following sections explain each part of the transcoder.

6.4.1. Features

Four different features are considered: the MV Variance Distance (exactly as seen in Sec. 6.3.1), the MV Phase Variance, the number of DCT Coefficients and the Energy of DCT Coefficients. The features are computed for each CU, and they are only computed if all H.264/AVC macroblocks within that CU are encoded in *inter* mode. Also, all features produce values in the range $[0, \infty)$.

MV Phase Variance

Different from the MV Variance Distance, which measures the variance of the magnitude of the motion vectors, this feature is computed as the variance of the phases of all motion vectors within a particular block. The idea to use the phase, instead of the magnitude of the motion vectors, is to overcome the limitation of scaling the motion vectors so that they have the same reference frame (note that scaling a motion vector does not change its phase). The phase is computed as:

$$phase = atan2\left(mv^{k}_{n \to n-\alpha}.y, mv^{k}_{n \to n-\alpha}.x\right) \tag{6.3}$$

Note that the phase lies in the closed interval $[-\pi, \pi]$, according to Fig. 6.8. Also, of particular interest, the phase of the $(0, 0)$ motion vector is considered to be 0. Before computing the variance, the motion vectors are also propagated to each 4×4 block.

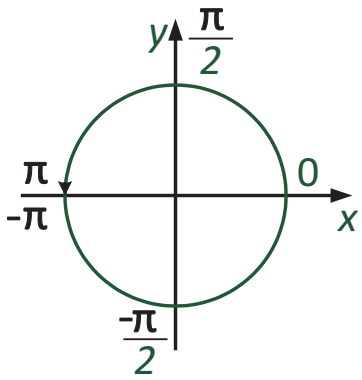


Figure 6.8.: Computing the motion vector phase. Additionally to the phases shown on the plot, the phase of the $(0, 0)$ motion vector is considered to be 0.

Number of DCT Coefficients

This simple feature is the number of non-zero DCT coefficients encoded in the H.264/AVC bitstream for a particular block. The magnitude of these coefficients is not used, just whether or not a coefficient was transmitted. The idea of using the number of DCT coefficients is that, if there is a small number of DCT coefficients for a given block, it means that the prediction for that block was good and the residual is small and, therefore, a good prediction may be found using a larger block. On the other hand, if there is a larger number of DCT coefficients for a given block, then the prediction for that block is not good, and the block may need to be sub-partitioned to find a good prediction. Naturally, the information on the motion vectors may help on this decision, but this will be studied later in this chapter.

Energy of DCT Coefficients

This feature is computed as $E_C = \sum_i C_i^2$, where C_i denotes the DCT coefficients within a particular block. The idea of using the energy of DCT coefficients is the same as the number of DCT coefficients, but the energy gives a more complete information about the magnitude of the residual than just the number of coefficients.

6.4.2. Computing the thresholds

In order to adapt the thresholds to the content of the current sequence being encoded, the first k frames of the sequence are used for training, and the transcod-

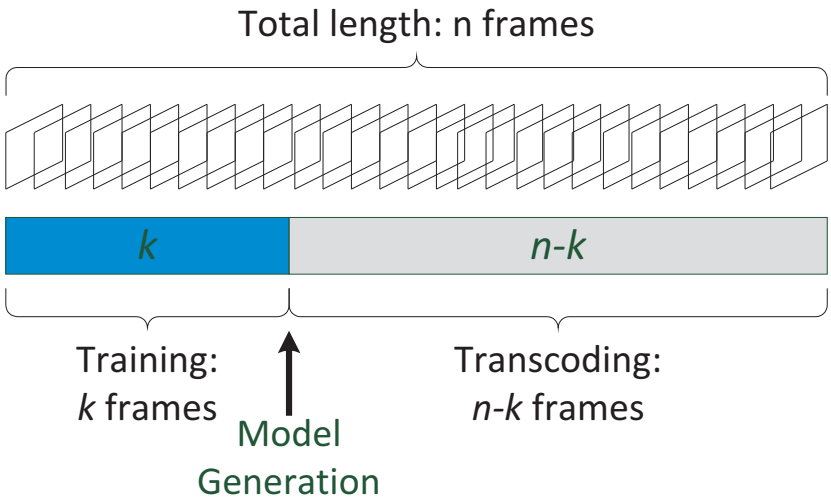


Figure 6.9.: Transcoding operation. When transcoding the first k frames, all possible modes in the HEVC are tested. This part is called *training* phase. Then, the transcoder builds the model (with information gathered in the training phase). Finally, it starts the *transcoding* phase, where the model is used to select which partitions will be tested.

ing operates in the following $n - k$ frames, as shown in Fig. 6.9. When transcoding the training sub-sequence, the transcoder computes the relevant feature for each block using the H.264/AVC information, and stores these features in an array $\mathbf{F}^d = \left[f_0^d\ ,\ f_1^d\ ,\ \ldots\ ,\ f_{N-1}^d\right]$, with elements f_i^d, where d refers to the depth of the CU and i refers to the feature computed for a particular CU i, and N refers to the number of CUs. For the training frames, the computed features are not used to decide which partitions will be tested - instead, all HEVC modes are tested for these frames. After the decision for a given CU has been made, the transcoder stores the mode chosen in an array $\mathbf{C}^d = \left[c_0^d\ ,\ c_1^d\ ,\ \ldots\ ,\ c_{N-1}^d\right]$, where c_i^d refers to the class of the i-th CU. In order to simplify the large number of modes in the HEVC codec, the transcoder stores the chosen mode information in three classes: (i) if the CU was split; (ii) if the CU was encoded as SKIP or with a $2N \times 2N$ PU; and (iii) if the CU was encoded using any other mode.

After it finishes transcoding the training frames, the transcoder uses the two arrays, $\mathbf{F}^d$ and $\mathbf{C}^d$, to compute the thresholds T_{low}^d and T_{high}^d (where d corresponds to the depth of the HEVC CU) using a percentile criterion. For each depth, the thresholds are computed as:

- T_{low}^d is chosen as the highest value for which 90% of the HEVC partitions with $f_i^d \leq T_{low}^d$ are encoded using either the SKIP or $2N \times 2N$ mode.

- T^d_{high} is chosen as the lowest value for which 90% of the HEVC partitions with $f^d_i > T^d_{high}$ are split.

Fig. 6.10 illustrates the computation of the thresholds using the four features. In the figure, it can be seen that, for all features, the higher the value of the feature, the more likely it is that that CU is split in the HEVC. On the other hand, the lower the value of the feature, the more likely it is that the CU is encoded with a PU size of $2N \times 2N$. Regardless of the feature value, a small amount of CUs are encoded using the other modes (class (iii)).

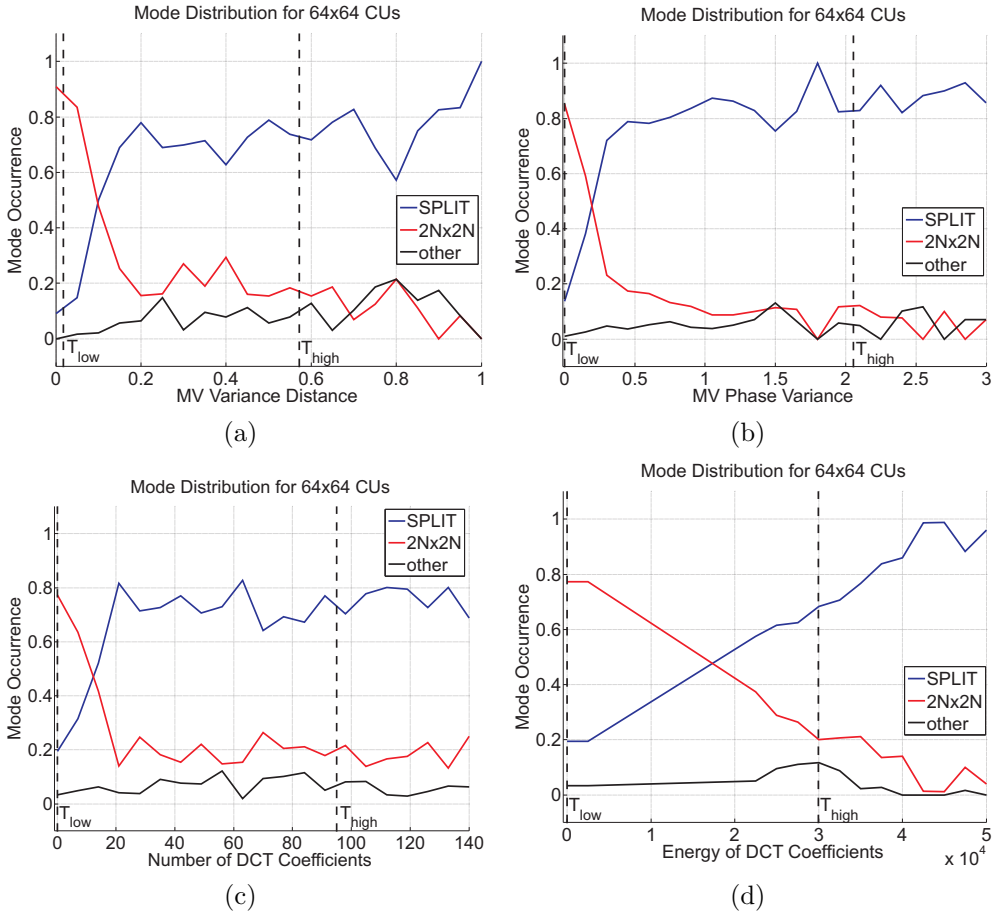


Figure 6.10.: Mode distribution and threshold computation for different features: (a) MV Variance Distance; (b) MV Phase Variance; (c) Number of DCT Coefficients; and (d) Energy of DCT Coefficients. The sequence being encoded is Basketball Drill, the data refers to the first 25 frames and the CU is 64×64 pixels.

Once the transcoding of the training sub-sequence is finished, the thresholds are computed. For the rest of the frames, the transcoder uses the computed thresholds to decide which HEVC partitions will be tested. For this transcoder, according to the

feature value f_i^d for the current CU, the transcoder will apply the following rules, which are shown in Fig. 6.11.

- If $f_i^d \leq T_{low}^d$ (R_0, in the figure), then only the SKIP and the $2N \times 2N$ modes are tested, and the CU is *not* split.
- If $f_i^d > T_{high}^d$ (R_2, in the figure), then only the SKIP mode is tested for this depth, and the CU is split.
- Otherwise (i.e., if $T_{low}^d < f_i^d \leq T_{high}^d$, R_1, in the figure) , then all modes are tested for this depth, and the CU is split.

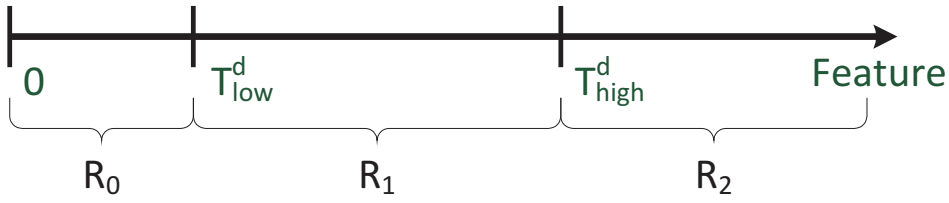


Figure 6.11.: Example of feature classification.

According to the feature value, the transcoder tests the PUs according to these rules. If the rules state the the CU should not be split, it chooses the best mode, among those tested, and proceeds to the next CU. Otherwise, if the CU is split, the algorithm is repeated for the four children CUs, unless the child CU is of size 16×16, in which case the transcoder applies the algorithm described next.

6.4.3. Mode Mapping for the 16x16 CUs

The thresholds seen on the previous section are used for the transcoder to decide which partitions are tested for the 64×64 and 32×32 CUs, for which there is no partitioning information in the H.264/AVC (since the macroblock size is 16×16). In order to decide how the 16×16 CUs are tested, the transcoder uses the H.264/AVC macroblock mode. Note that, depending on the feature value for the larger CUs, the algorithm described here may not be applied for a particular region.

Tests have shown that keeping the exact same partitions as the H.264/AVC leads to larges losses, and the gain in complexity is not large enough. For this reason, a different strategy has been designed. The rationale used is to test partitions that are of the same size or larger than the H.264/AVC partition. A simple look-up table is used to decide

Table 6.6.: PUs tested for a 16×16 CU according to the H.264/AVC macroblock type.

		H.264/AVC Macroblock Type					
		PSKIP	16×16	16×8	8×16	8×8	INTRA
HEVC PUs Tested	*SKIP/MERGE*	X	X	X	X	X	X
	$2N \times 2N$		X	X	X	X	X
	$2N \times N$			X		X	X
	$N \times 2N$				X	X	X
	$2N \times nU$			X		X	X
	$2N \times nD$			X		X	X
	$nR \times 2N$				X	X	X
	$nL \times 2N$				X	X	X
	$N \times N$						
	INTRA						X
	SPLIT					X	X

Table 6.7.: PUs tested for a 8×8 CU according to the H.264/AVC sub-macroblock type.

		H.264/AVC Sub Macroblock Type					
		BSKIP	8×8	8×4	4×8	4×4	INTRA
HEVC PUs Tested	*SKIP/MERGE*	X	X	X	X	X	X
	$2N \times 2N$		X	X	X	X	X
	$2N \times N$			X		X	X
	$N \times 2N$				X	X	X
	$2N \times nU$			X		X	X
	$2N \times nD$			X		X	X
	$nR \times 2N$				X	X	X
	$nL \times 2N$				X	X	X
	$N \times N$					X	X
	INTRA						X

which modes will be tested in the HEVC, according to the H.264/AVC macroblock and sub-macroblock types. The complete look-up tables for the 16×16 and 8×8 CUs can be seen in Tables 6.6 and 6.7, respectively.

6.4.4. MV Reuse and Refinement

The MV Reuse and Refinement are also used in this transcoder. For any PU size, all H.264/AVC motion vectors are considered for integer motion estimation, and no further refinement is performed at the integer pixel level. Then, the default HEVC sub-pixel search is applied.

6.5. Using Linear Discriminant Functions in the Transcoder

In the previous sections the transcoder uses only one feature at a time. However, perhaps a better solution would involve the use of different features to attempt to better predict the HEVC partitioning. There are several methods in the literature that could be used to classify a given set of features [16, 19, 38]. Here, a simple method is used, called Linear Discriminant Functions [13, 111]. The main reason this method was chosen is that both the training and the classification itself are very low complexity operations [13, 111], in fact, the training can be performed in a non-iterative way. Thus, this method can be used inside the transcoder loop without hindering the transcoder complexity. A brief overview of Linear Discriminant Functions is given in Appendix F.

As in the previous presented transcoders, the first k frames of the sequence are used for training, and the transcoding operates in the following $n-k$ frames, as shown in Fig. 6.9. Similarly, when transcoding the training sub-sequence, the transcoder computes all features for each block, storing it in an array $\mathbf{Fs}^d$ (where d refers to the depth of the CU). Also, for the training sub-sequence, all HEVC modes are tested, and the decision for each CU is stored in an array $\mathbf{Cs}^d$. The first modification, compared to the previous sections, is that the transcoder stores the chosen mode information in only two classes: (i) if the CU was split; and (ii) if the CU was not split. Therefore, the transcoder is attempting to classify only whether the CU was split or not.

It was noted that one of the features, the MV Variance Distance, has a very high correlation with one of the classes. A block with a high value for the MV Variance Distance is most likely to be split. For this reason, the incoming CUs with a MV Variance Distance v higher than a threshold (T^d_{high}, computed as the 90-th percentile, as explained in Sec. 6.4.2) are removed from the set on the assumption that they shall be split. For the rest of the CUs (i.e., the CUs for which $v \leq T^d_{high}$), the classification is applied using the linear discriminant function. This procedure is illustrated in Fig. 6.12.

In order to use the linear discriminant functions, seven features are considered:

1) The total number of H.264/AVC partitions in the incoming CU. This is the number of different inter-prediction blocks in the H.264/AVC for the region defined by the current CU.

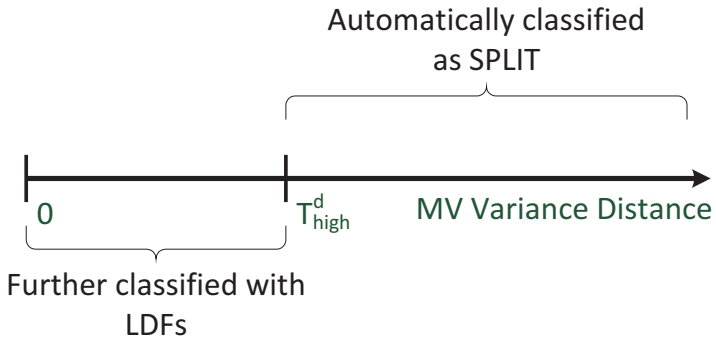


Figure 6.12.: Example of classification. All CUs with a MV Variance Distance higher than the threshold T^d_{high} are automatically classified as split. For the remaining CUs, linear discriminant functions (LDFs) are used to classify it between split or not split.

2) The MV Variance Distance, as seen in Sec. 6.3.1.

3) The variance of the x component for the motion vectors within the CU.

4) The variance of the y component for the motion vectors within the CU.

5) The MV Phase Variance, as seen in Sec 6.4.1.

6) The number of DCT coefficients, as seen in Sec. 6.4.1.

7) The average energy of the DCT coefficients. This is energy of the DCT coefficients (as seen in Sec. 6.4.1) divided by the number of DCT coefficients. If there are no non-zero DCT coefficients within the CU, then it is considered as zero.

In addition to these features, another two values are added to the feature vector: (i) 1; and (ii) the average of the previous seven feature vectors. Therefore, the total length of the feature vector is 9.

After it finishes transcoding the training sub-sequence, and after the CUs with MV Variance Distance $v > T^d_{high}$ are removed from the set, the remaining elements in the set are used to compute the optimal weights for the linear discriminant functions (see Appendix F). Then, the following algorithm is applied to decide which partitions are tested for each CU.

When deciding which modes will be tested for a given CU, the transcoder first computes the MV Variance Distance v. If $v > T^d_{high}$, then this partition is split and the algorithm repeats itself for the CUs at the next depth (only the SKIP mode is tested at this depth). Otherwise, the transcoder computes the remaining features and com-

putes the values for the two linear discriminant functions. If the outcome is split, then this partition is split and the algorithm repeats itself for the CUs at the next depth (again, only the SKIP mode is tested at this depth). Otherwise, if the outcome is not to split, then all modes at this depth are tested and the partition is *not* split. Finally, the transcoder decides for the best mode among those tested, and proceeds to the next CU.

This algorithm is applied for the 64×64 and 32×32 CUs. For the 16×16 and 8×8 CUs, the H.264/AVC macroblock and sub-macroblock types are used, as explained in Sec. 6.4.3. Also, if there is an intra block within the CU, then the algorithm is not used and all partitions are tested for that CU, and the CU is split.

6.5.1. MV Reuse and Refinement

Similarly to the transcoder options presented in Sec. 6.4, the MV Reuse and Refinement are also used in this transcoder. For any PU size, all H.264/AVC motion vectors are considered for integer motion estimation, and no further refinement is performed at the integer pixel level. Then, the default HEVC sub-pixel search is applied.

6.6. Experimental Results

The experimental settings are similar to those in Sec. 6.2.1, using QPs of 37, 32, 27 and 22 to compute both the BD-rate and speed-up figures. The only difference is that the coding configuration used here was simplified to a *IPP* configuration with 1 reference frame in both H.264/AVC and HEVC codecs. Four sequences are used to evaluate the proposed transcoders: Basketball Drill 832×480 50 Hz, BQMall 832×480 60 Hz, Party Scene 832×480 50 Hz and Race Horses 832×480 30 Hz. All of these sequences are part of the HEVC testing dataset (see Appendix C for details).

Table 6.8.: PTCM results using 10 frames for training. The BD-Rate is shown as a percentage, using RT-EPZS as anchor, and the Speed-Up is relative to RT-EPZS.

		Basketball Drill		BQ Mall		PartyScene		RaceHorses	
	Transcoder	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up
Length = 2.5 *s*	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.29	1.09	2.27	1.08	1.39	1.10	1.94	1.10
	PT-MVVD (iv)	5.34	2.15	7.80	1.95	15.1	2.19	3.99	1.65
	PTCM-MVVD	**3.61**	1.77	**4.36**	1.85	**2.69**	1.79	**2.77**	1.50
	PTCM-MVPV	7.11	1.73	6.61	1.72	3.21	1.72	2.79	1.45
	PTCM-NDCT	5.13	1.89	6.85	2.03	2.92	1.88	4.08	1.67
	PTCM-EDCT	5.57	**2.11**	7.91	**2.28**	3.19	**2.08**	4.54	**1.76**
	PTCM-LDF	4.93	2.00	5.98	2.12	3.23	1.99	5.42	1.71
Length = 5 *s*	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.24	1.09	2.15	1.07	1.35	1.10	1.95	1.10
	PT-MVVD (iv)	5.80	2.14	7.10	2.01	14.2	2.16	3.70	1.76
	PTCM-MVVD	**4.04**	1.82	**4.00**	1.94	**2.97**	1.86	**2.79**	1.58
	PTCM-MVPV	7.78	1.78	5.88	1.79	3.53	1.79	2.80	1.52
	PTCM-NDCT	5.31	1.97	6.55	2.18	3.36	1.96	3.84	1.79
	PTCM-EDCT	5.77	**2.20**	7.69	**2.48**	3.63	**2.18**	4.31	**1.89**
	PTCM-LDF	5.43	2.08	5.58	2.26	3.59	2.09	4.94	1.76
Length = 10 *s*	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.36	1.09	1.89	1.07	1.09	1.10	2.05	1.10
	PT-MVVD (iv)	6.15	2.13	7.17	1.96	16.2	2.33	4.37	1.69
	PTCM-MVVD	**4.43**	1.86	**3.92**	1.96	**2.68**	2.00	3.35	1.63
	PTCM-MVPV	8.26	1.81	5.33	1.80	4.15	1.96	**3.34**	1.56
	PTCM-NDCT	5.66	2.00	6.42	2.22	3.32	2.14	4.71	1.83
	PTCM-EDCT	6.09	**2.25**	7.58	**2.58**	3.58	**2.40**	5.33	**1.94**
	PTCM-LDF	5.84	2.12	5.93	2.34	3.34	2.27	6.59	1.87

Table 6.9.: PTCM results using 25 frames for training. The BD-Rate is shown as a percentage, using RT-EPZS as anchor, and the Speed-Up is relative to RT-EPZS.

		Basketball Drill		BQ Mall		PartyScene		RaceHorses	
	Transcoder	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up
Length = 2.5 s	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.29	1.09	2.27	1.09	1.39	1.09	1.94	1.08
	PT-MVVD (iv)	5.34	2.16	7.80	1.96	15.1	2.19	3.99	1.66
	PTCM-MVVD	**2.96**	1.57	**4.02**	1.63	**2.09**	1.55	2.05	1.31
	PTCM-MVPV	3.38	1.52	4.48	1.55	2.14	1.52	**2.04**	1.29
	PTCM-NDCT	4.42	1.67	6.35	1.82	2.36	1.64	2.95	1.42
	PTCM-EDCT	4.68	**1.79**	7.00	**1.98**	2.63	**1.78**	3.36	**1.47**
	PTCM-LDF	4.42	1.74	5.63	1.87	3.07	1.72	4.71	1.44
Length = 5 s	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.24	1.09	2.15	1.07	1.35	1.11	1.95	1.10
	PT-MVVD (iv)	5.80	2.18	7.10	2.05	14.2	2.21	3.70	1.79
	PTCM-MVVD	**3.64**	1.72	**3.82**	1.79	**2.55**	1.73	2.45	1.48
	PTCM-MVPV	4.17	1.65	4.12	1.69	2.65	1.68	**2.42**	1.44
	PTCM-NDCT	5.06	1.84	6.28	2.08	2.96	1.84	3.29	1.64
	PTCM-EDCT	5.28	**2.01**	7.09	**2.31**	3.29	**2.04**	3.78	**1.73**
	PTCM-LDF	5.14	1.95	5.69	2.13	3.62	1.96	5.00	1.68
Length = 10 s	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.36	1.09	1.89	1.07	1.09	1.10	2.05	1.10
	PT-MVVD (iv)	6.15	2.18	7.17	2.01	16.2	2.38	4.37	1.72
	PTCM-MVVD	**4.05**	1.80	**3.73**	1.84	**2.35**	1.87	3.19	1.56
	PTCM-MVPV	4.52	1.72	4.09	1.75	2.75	1.84	**3.08**	1.50
	PTCM-NDCT	5.54	1.94	6.33	2.19	2.96	2.04	4.40	1.74
	PTCM-EDCT	5.77	2.13	7.26	**2.48**	3.39	**2.31**	5.04	**1.85**
	PTCM-LDF	5.65	**2.18**	6.41	2.27	3.45	2.19	7.78	1.78

Table 6.10.: PTCM results using 50 frames for training. The BD-Rate is shown as a percentage, using RT-EPZS as anchor, and the Speed-Up is relative to RT-EPZS.

		Basketball Drill		BQ Mall		PartyScene		RaceHorses	
	Transcoder	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up	BD-Rate	Speed-Up
Length = 2.5 *s*	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.29	1.09	2.27	1.09	1.39	1.10	1.94	1.10
	PT-MVVD (iv)	5.34	2.15	7.80	1.96	15.1	2.19	3.99	1.66
	PTCM-MVVD	**2.24**	1.40	**2.93**	1.45	1.47	1.36	1.09	1.14
	PTCM-MVPV	2.36	1.36	3.10	1.40	**1.43**	1.33	**1.03**	1.13
	PTCM-NDCT	3.14	1.45	4.41	1.51	1.66	1.41	1.42	1.17
	PTCM-EDCT	3.40	**1.52**	5.31	**1.68**	1.80	1.46	1.61	**1.19**
	PTCM-LDF	3.24	1.49	5.01	1.61	2.52	**1.47**	2.59	**1.19**
Length = 5 *s*	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.24	1.09	2.15	1.07	1.35	1.11	1.95	1.10
	PT-MVVD (iv)	5.80	2.13	7.10	2.01	14.2	2.17	3.70	1.76
	PTCM-MVVD	**3.24**	1.60	**3.12**	1.65	**2.18**	1.56	1.97	1.35
	PTCM-MVPV	3.52	1.54	3.18	1.58	2.19	1.66	**1.91**	1.32
	PTCM-NDCT	4.44	1.69	4.90	1.79	2.59	1.74	2.46	1.44
	PTCM-EDCT	4.70	**1.79**	6.09	**2.05**	2.78	1.77	2.91	**1.50**
	PTCM-LDF	4.63	1.76	5.89	2.01	3.62	**2.17**	4.16	1.48
Length = 10 *s*	RT-EPZS	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	RT-MVR	2.36	1.09	1.89	1.07	1.09	1.10	2.05	1.10
	PT-MVVD (iv)	6.15	2.13	7.17	1.97	16.1	2.33	4.37	1.48
	PTCM-MVVD	**3.90**	1.72	**1.89**	1.76	**2.04**	1.75	2.86	1.43
	PTCM-MVPV	4.25	1.65	3.35	1.67	2.11	1.68	**2.80**	1.61
	PTCM-NDCT	5.16	1.85	5.40	1.95	2.68	1.90	3.81	1.70
	PTCM-EDCT	5.45	**1.99**	6.75	**2.32**	3.00	2.03	4.45	1.66
	PTCM-LDF	5.51	1.94	7.35	2.14	3.69	**2.06**	7.59	**1.68**

Five options of the proposed transcoder using content based modeling are evaluated, and three other options are used as benchmarks. The full list of tested transcoders, as referred on this section, follows:

(i) RT-EPZS: this corresponds to the trivial transcoder, decoding the entire sequence and re-encoding using the HEVC standard fast motion estimation and mode decision algorithms (Sec. 6.2.1). This is used as anchor both for BD-rate calculation and speed-up results.

(ii) RT-MVR: MV Reuse Transcoder: this corresponds to the transcoder shown in Sec. 6.2.

(iii) PT-MVVD: The proposed transcoder as seen in Sec. 6.3, with parameters $T_{low} = 1$, $T_{high} = 100$, scaling the motion vectors and not using integer motion vector refinement (this corresponds to option (iv) in Sec. 6.3.5).

(iv) PTCM-MVVD: The content modeling transcoder (Sec. 6.4) using MV Variance Distance.

(v) PTCM-MVPV: The content modeling transcoder (Sec. 6.4) using MV Phase Variance.

(vi) PTCM-NDCT: The content modeling transcoder (Sec. 6.4) using the number of DCT coefficients.

(vii) PTCM-EDCT: The content modeling transcoder (Sec. 6.4) using the energy of DCT coefficients.

(viii) PTCM-LDF: The content modeling transcoder (Sec. 6.5) using Linear Discriminant Functions.

The experiment was designed to provide answers to the following questions: (i) how many frames are needed to build an efficient model; and (ii) for how long the model works (i.e., for how many frames can the model be successfully applied). Thus, three different training lengths were tested: 10, 25 and 50 frames, and, in order to investigate the impact of using the same parameters for a longer period, three different lengths of the sequences were used: 2.5, 5 and 10 seconds. The complete results are shown in Tables 6.8, 6.9 and 6.10, both in terms of BD-rate and speed-up. Minimum, average and maximum figures for BD-Rate and Speed-Up are shown in Fig. 6.13, grouping the results by the length of the sequence being transcoded, and Fig. 6.14, grouping the

results by the number of frames used for training. For all cases, RT-EPZS is used as anchor for both BD-bitrate and speed-up figures.

The first thing that can be noticed is that the speed-up figures for this experiment are lower than those for the experiments shown in Sec. 6.3.5. The main reason is that the experiment in this section uses only one reference frame, instead of four, as in Sec. 6.3.5. Note that the results for PT-MVVD in Tables 6.8, 6.9 and 6.10 are different than those provided in Table 6.2, as the experimental settings are different.

It can be seen that reference transcoder based on the MV Reuse (RT-MVR) shows a loss of up to 2.29% in terms of BD-rate (for Basketball Drill sequence encoding 2.5 seconds, shown in Tables 6.8, 6.9 and 6.10), and 1.9% on average, but the speed-up is only of 1.09, on average. On the other hand, the proposed transcoder based on the MV Variance Distance (PT-MVVD, as seen in Sec. 6.3) shows good speed-up figures (up to 2.38, for PartyScene sequence encoding 10 seconds of the sequence, shown in Table 6.9, and 2.04 on average), but the RD loss is significantly higher, especially for the PartyScene sequence (up to 16.2%, shown in Table 6.9) and BQMall (up to 7.80%, shown in Table 6.10) sequences.

Using the content modeling approach, the rate-distortion performance loss is significantly lower, regardless of the features used. In the worst case, the BD-rate loss is 8.26%, when using the MV phase variance as feature (PTCM-MVPV) (for Basketball Drill sequence, using 10 frames for training and transcoding 10 *s*, seen in Table 6.8). Among the features, the MV Variance Distance (PTCM-MVVD) presented the lowest BD-rate loss for the majority of the cases, with a BD-rate loss ranging from 1.09% (for RaceHorses sequence, with 50 frames for training and encoding 2.5 *s*, seen in Table 6.10) to 4.43% (for Basketball Drill sequence, with 10 frames for training and encoding 10 *s*, seen in Table 6.8). For most of the tests, the MV Phase Variance (PTCM-MVPV) also presents a very low loss, close to PTCM-MVVD, with the notable exception of Basketball Drill and BQMall sequences using 10 frames for training (seen in Table 6.8), where the loss is among the highest. However, the speed-up figures for both PTCM-MVVD and PTCM-MVPV are the lowest for the content modeling approach. The fastest option for the majority of the cases is using the Energy of the DCT coefficients (PTCM-EDCT), at the expense of a worse RD loss. Using the Linear Discriminant Functions (PTCM-LDF), the speed-up figures are closer to PTCM-EDCT, while the rate distortion performance loss is lower.

Table 6.11.: Average results for PTCM-LDF. The BD-Rate is shown as a percentage, using RT-EPZS as anchor, and the Speed-Up is relative to RT-EPZS.

		BD-Rate			Speed-Up		
		Number of Frames used for Training					
		10	25	50	10	25	50
Length	$2.5s$	4.89	4.45	3.34	1.95	1.69	1.44
	$5s$	4.88	4.86	4.57	2.04	1.93	1.85
	$10s$	5.42	5.82	6.03	2.15	2.10	1.95

Analysing Figs. 6.13(a), 6.13(c) and 6.13(e), it can be seen that, as expected, the higher the number of frames used for training and the shorter the sequence being transcoded, the better the rate distortion performance (although there are a few exceptions, notably when using the Linear Discriminant Functions (PTCM-LDF) encoding 5 seconds, shown in Fig. 6.13(c), and 10 seconds, shown in Fig. 6.13(e)). However, apart from the MV Phase Variance feature (PTCM-MVPV), this gain in performance is rather low. Using 25 frames for training results in -0.9%, -0.57% and -0.40% average BD-rate gains when encoding 2.5, 5 and 10 seconds, respectively, compared to using only 10 frames for training, on average among all features and sequences. Using 50 frames for training results on -1.15%, -0.59% and -0.38% average BD-rate gains when encoding 2.5, 5 and 10 seconds, respectively, compared to using 25 frames for training. This happens for two reasons: generally, a longer training yields a better model, and thus the loss in the transcoding phase is lower; and because the best rate-distortion performance is obtained in the training part, when full encoding is being performed. Note that the largest difference occurs when the ratio of the number of frames used for training compared to the length of the sequence being encoded is the highest (50 frames used for training and encoding only 2.5 seconds).

On the other hand, analysing Figs. 6.14(b), 6.14(d) and 6.14(f), it can be seen that the speed-up gain when encoding a larger sequence is also small. Encoding 5 seconds results in a speed-up gain of 1.05, 1.13 and 1.20, when using 10, 25 and 50 frames for training, respectively, on average among all features and sequences, compared to encoding only 2.5 seconds. Encoding 10 seconds results in a speed-up gain of 1.04, 1.07 and 1.09, when using 10, 25 and 50 frames for training, respectively, on average among all features and sequences, compared to encoding 5 seconds. Notice that the largest difference occurs when using 50 frames for training and encoding shorter sequences (2.5 and 5 seconds), and even in this case the gain is only 20%.

Table 6.11 compares the number of frames used for training and the length of the sequence being encoded for the specific case of the Linear Discriminant Functions (PTCM-LDF). Although there are a few outliers, the behaviour discussed in the previous paragraphs can be observed: the longer the training sequence and the shorter the sequence, the better the rate distortion performance (the lowest average loss, 3.34%, occurs for training with 50 frames and encoding 2.5 seconds of the sequence). At the same time, the shorter the training sequence and the longer the sequence, the fastest the transcoder (the fastest option, at 2.15, occurs for training with 10 sequences and encoding 10 seconds). However, for both cases, this difference is rather small.

6.7. Conclusion

In this chapter, several transcoding options from H.264/AVC bitstreams to the HEVC codec are presented and evaluated. First, a transcoder based on motion vector reuse (RT-MVR) was implemented, in order to be used as benchmark and to identify potential strategies to improve transcoding. It has been verified that just reusing the motion vectors from the incoming bitstream is not enough for an efficient transcoder, as the HEVC codec still tests many CU sizes and PU modes for each LCU. However, it has been verified that the H.264/AVC motion vectors are highly correlated to those of the HEVC, as this transcoder yields a very small rate-distortion performance loss compared to the trivial transcoder (RT-EPZS).

Using the knowledge obtained from the transcoder based on MV Reuse (RT-MVR), a new transcoder to the HEVC is proposed. This transcoder is based on a new metric to compute the similarity of the H.264/AVC motion vectors, the motion vector variance distance (PT-MVVD), which is used to decide which HEVC partitions are tested on the transcoder. The proposed transcoder is capable of complexity scalability, trading off rate-distortion performance for complexity reduction. Compared to RT-MVR, PT-MVVD is up to 3.7 times faster. This complexity reduction is achieved at a loss of rate-distortion performance, however, even for the same levels of rate-distortion performance, PT-MVVD is up to 2.6 faster than RT-MVR, for a low-delay configuration using four reference frames.

Finally, a transcoder based on content modeling is proposed (PTCM). In this transcoder, a part of the sequence is used for training. For the training sequence, full re-encoding is applied, and the transcoder then uses this information, along with the infor-

mation from the H.264/AVC bitstream, to generate a content-specific model to map the H.264/AVC partitions in HEVC CUs. Afterwards, the transcoder applies this model to decide which partitions are tested for the rest of the sequence. Experiments have shown that the performance of this transcoder is much better than the previous transcoding options (the proposed transcoder based on the MV Variance distance, PT-MVVD, and the reference transcoder based on MV Reuse, RT-MVR), yielding a much lower rate-distortion loss (compared to the trivial transcoder, RT-EPZS) at a competitive complexity performance.

Among the features that have been tested, the best compromise between complexity and rate-distortion performance is obtained when a combination of all features, combined by a linear classifier (in this case, using linear discriminant functions, PTCM-LDF), is used. Experiments were carried to find out the number of frames needed for training and also if the model built is robust enough to be used for a long period. From the results of these experiments, it was observed that the gain in rate-distortion performance of using more frames to generate the model is rather low. At the same time, while the rate-distortion performance losses of applying this model for a long period are small, the gain in complexity is also rather low. This leads to the conclusion that a better transcoding option would involve using less frames for training and encoding a shorter sequence, repeating the training more often, in order to keep track of the properties of the sequence being transcoded. This way, the speed-up could be kept at roughly 2 (for a low-delay configuration using only one reference frame), while the benefits of a small rate-distortion loss would be kept. In addition, the transcoder would be robust if the sequence properties are changed. The exact parameters could be changed according to the application, or it could be done adaptively, using algorithms to detect scene changes [109, 74] to decide when to build a new model, or developing an algorithm specifically to decide when a new model should be built.

The next chapter presents the conclusions for this thesis, as well as directions for future research.

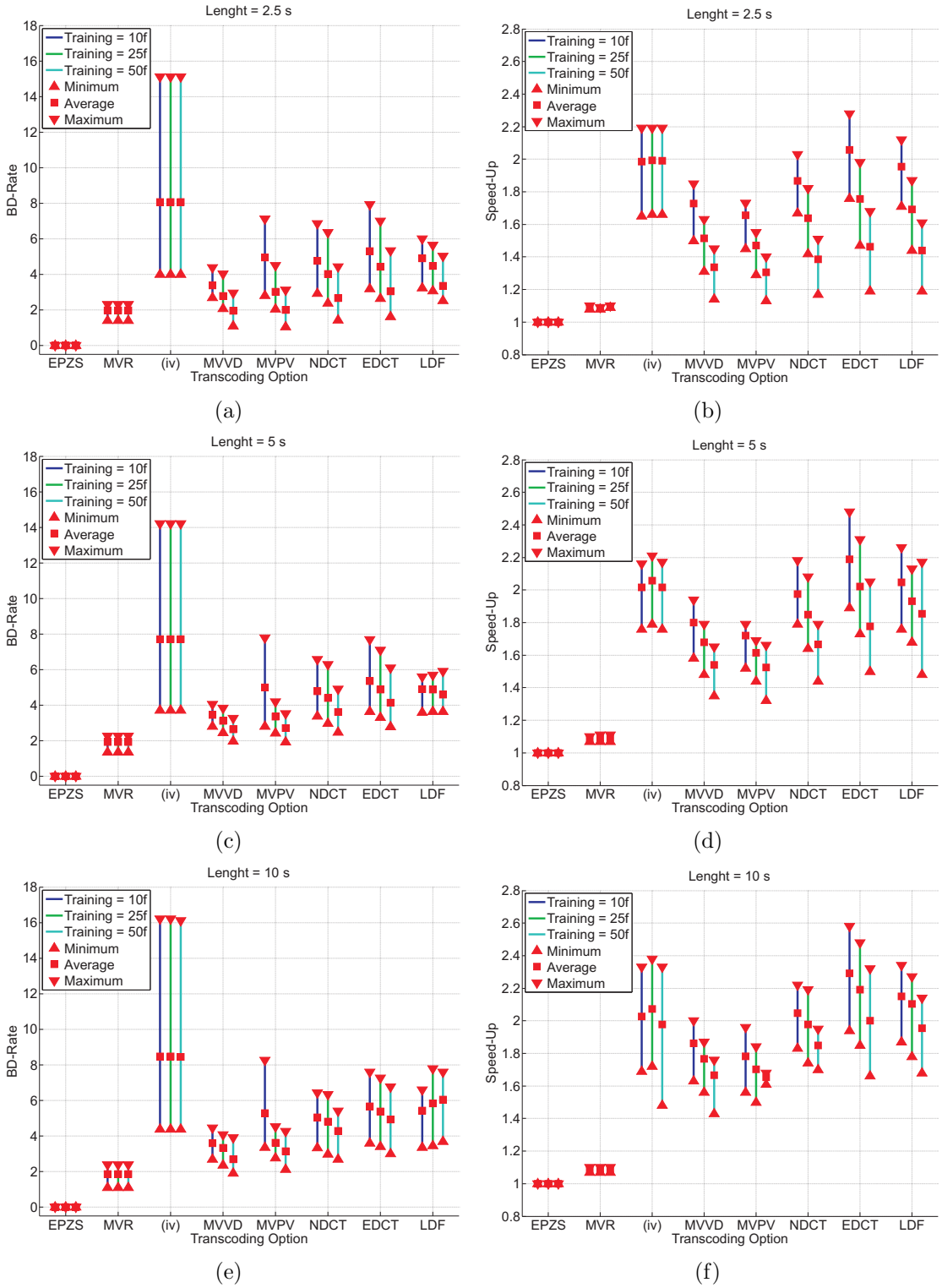


Figure 6.13.: Results for PTCM grouped by the sequence length. In the figure, the transcoding options are shown in the order: RT-EPZS, RT-MVR, PT-MVVD (iv), PTCM-MVVD, PTCM-MVPV, PTCM-NDCT, PTCM-EDCT, PTCM-LDF.

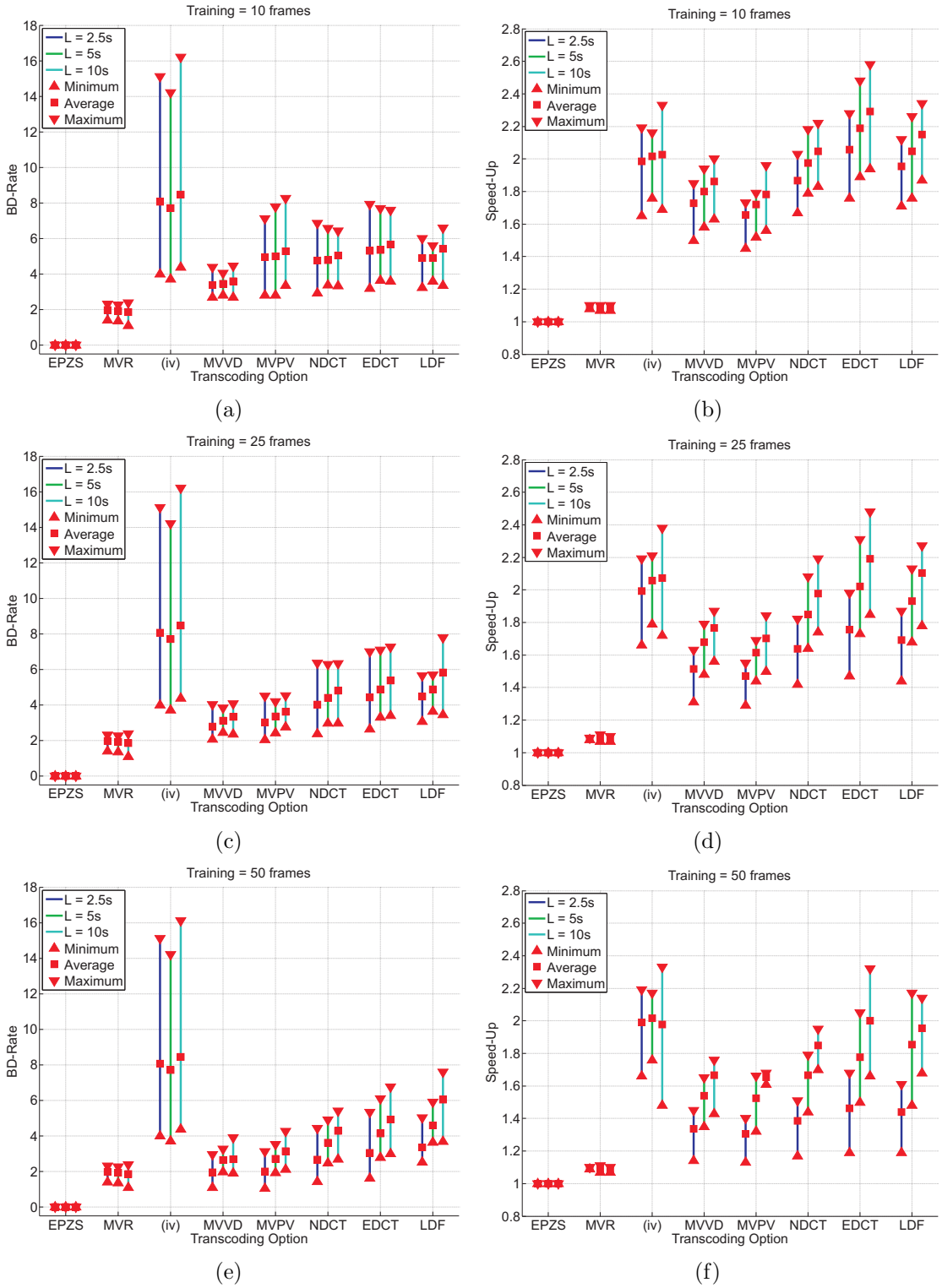


Figure 6.14.: Results for PTCM grouped by the training length. In the figure, the transcoding options are shown in the order: RT-EPZS, RT-MVR, PT-MVVD (iv), PTCM-MVVD, PTCM-MVPV, PTCM-NDCT, PTCM-EDCT, PTCM-LDF.

Chapter 7.

Conclusions and Future Developments

The research presented in this Thesis focuses on developing video transcoders from the current standard, H.264/AVC, to two different video codecs, a Wavelet-based Scalable Codec, W-SVC, and the new, emerging standard High Efficiency Video Coding, HEVC. The application of this research, however, is not constrained to these transcoders, as many of the developed techniques have the potential to be incorporated in many different video transcoding architectures. This chapter presents a brief summary of the thesis, and ends with new research directions that have been identified as potential areas for further performance improvements and interesting new functionalities.

After the introductory chapter, the second chapter presents an introduction to both video coding and transcoding, with special attention to the modules that are needed the most for a better comprehension of the later chapters. A survey on recent research on video transcoding is also given in this chapter. The next chapter presents overviews on the three codecs that are used in this thesis: the H.264/AVC, the HEVC and the W-SVC codecs. While a complete explanation of any of these three codecs would be out of the scope of this thesis, the main modules used on the transcoders presented in the later chapters are thoroughly explained, along with an explanation of the encoding and decoding flows of all three codecs.

The fourth chapter presents the state-of-the-art on one of the most important tools on video transcoding, the motion vector reuse and approximation, and proposes a new motion vector composition algorithm that is both more reliable and accurate than the current techniques. This chapter also presents a thorough evaluation of these motion vector approximation methods, with experiments designed to evaluate these methods

regardless of which codecs are used in the transcoders. Also, an evaluation of the effect of these methods within the H.264/AVC to W-SVC transcoder is given in this chapter. In the fifth chapter, a H.264/AVC to W-SVC transcoder is proposed. This transcoder uses some of the motion vector approximation methods presented in the fourth chapter, along with other techniques developed specifically to overcome some issues of this transcoder, to achieve full flexibility regarding the coding configuration or quantization parameter used in the H.264/AVC bitstream. It also explores the effect of using different macroblock sizes to reduce the transcoding loss. A comprehensive evaluation of this transcoder is given, both in terms of decoded video quality and complexity. Two versions of the proposed transcoder, targeting different rate-distortion and complexity requirements, are tested against both a trivial transcoder (i.e., decoding and re-encoding the sequence) using the full search algorithm, and two fast motion estimation methods, Hexagon Search [153] and EPZS [132]. The proposed transcoder outperforms both fast search algorithms in terms of decoded video quality, and even the full search algorithm for some cases. In terms of complexity, the proposed transcoder is always faster than EPZS, and it is faster than Hexagon search for the majority of the cases.

The sixth chapter presents the first transcoding strategies developed specifically to be used in the new HEVC codec. It starts by analysing the issues of this new challenge, evaluating the performance of classic transcoding strategies (the motion vector reuse) in the scope of this new transcoder. Then, it proposes a new complexity-scalable algorithm, that is able to trade off rate distortion performance for complexity savings, using a new feature to map the H.264/AVC partitioning into the HEVC modes, called Motion Vector Variance Distance. Finally, it tests other features, and a combination of these features, in a content-based algorithm designed to adapt itself to the contents of the sequence being encoded. Extensive evaluations of these features, and the effects of the division of the sequence into training and transcoding parts, are given.

7.1. Future Work

While this research has produced many different techniques, and answered some questions, there are still many areas for improvement. These issues are summarised for the two different transcoders. First, for the H.264/AVC to W-SVC transcoder:

- While the transcoder is able to use macroblock sizes of up to 64×64 pixels, the performance of the transcoder when using this macroblock size is not as good as the

performance when using 32×32 or 16×16 block sizes. The Reduced Complexity module is able to further reduce the complexity for these cases, but the transcoder is still testing more partitions than it is needed when very large macroblocks are used. A new study on how to map the H.264/AVC partitions targeting specifically these large block sizes could potentially improve the transcoder in this case. Even though this thesis shows that the potential gain in terms of quality when using larger block sizes in the W-SVC is mostly concentrated on 32×32 block sizes, using 64×64 macroblocks always yields better quality.

- Many of the transcoding algorithms are independent on the properties of the codec used for the incoming bitstream. However, other parts were built specifically to deal with H.264/AVC properties, such as the partitioning and, in the Reduced Complexity module, the use of the H.264/AVC CBP to drive the motion vector refinement. It would be interesting to adapt these functionalities to use more general properties, so that the transcoder could be used to transcode bitstreams from different codecs, such as MPEG-2, for instance.

And, for the second transcoder developed on this thesis, to the new HEVC codec:

- While the proposed transcoder complexity reduction is significant, it could still be reduced. New ways to map the H.264/AVC to the HEVC partitions could be developed, maybe using different machine learning techniques, or new features that use other information found in the H.264/AVC bitstream. Also, different sets of features could be used according to the sequence being encoded. This could be decided in the training phase. For instance, for some sequences the mapping could present a better performance when using the motion vectors (or features derived from the motion vectors, as the motion vector variance distance), while for other sequences the mapping could present a better performance when using the DCT coefficients (or features derived from the DCT coefficients, such as the energy of the coefficients).

- In this thesis, the effects of the division of the sequence for training and transcoding are studied. However, an improved and adaptive way to achieve this division, using algorithms to detect scene changes [109, 74] or a kind of "internal control" to detect when the model is no longer optimal, and to trigger a new training, could lead to a better and more robust transcoder.

- Also, this thesis only studied the effect of transcoding to the same coding configuration. In the HEVC, a hierarchical configuration (called random access configuration) is also used, due to its greater rate-distortion efficiency. The motion vector composition algorithm proposed in this thesis could be effectively used to change the coding configuration, providing more flexibility to the HEVC transcoder. This way, even greater rate distortion performance could be achieved, reducing the transcoding loss.

- The features used here are computed from the H.264/AVC bitstream, involving the partitioning, motion vectors and transform information. However, more information present in this bitstream could be used, such as the choice of the transform, or other combinations of the information that is already being reused could be used to improve the transcoder performance. For instance, a small variance in the H.264/AVC transform coefficients could potentially be mapped to a large partition in the HEVC, while the same coefficients, but concentrated in a small area (indicating that the prediction for that area is not as good), could mean that smaller HEVC partitions are more likely to be used.

- In this thesis, the large number of HEVC modes are combined into a small number of classes, in order to improve the classification. However, classifiers with more discriminatory capabilities could potentially be able to handle more classes, further reducing the transcoder complexity. Again, the choice of features plays an important role in the discriminatory capabilities of any classifier.

- The mode mapping in the content modeling transcoder presented, used to map H.264/AVC macroblock types into HEVC PUs at the 16×16 block size, could be vastly improved. While these modes contribute to the complexity to a smaller degree than the larger block sizes, it is expected that a better mapping could further reduce the transcoder's complexity.

- Finally, this thesis only explored transcoding techniques that aim to reduce the motion estimation and mode decision complexity. These are indeed the most complex parts of the new HEVC codec, however, the other modules of the codec, such as the transforms and quantization parameter optimisation, are also quite complex. It could be possible to improve the transcoder performance by using algorithms to reuse this information as well, as it has been done successfully for other DPCM/DCT transcoders [27, 28].

The previously enumerated challenges will be the subject of studies in the near future. In particular, we expect to continue the work on the HEVC transcoder, so that a more efficient transcoder is ready when the HEVC becomes a standard. An interesting feature for such a transcoder would be full flexibility regarding the coding configuration, as the W-SVC transcoder proposed, in particular using the proposed motion vector composition method to tackle multiple coding configurations in both source and target codecs.

Publications

[1] Eduardo Peixoto and Ebroul Izquierdo. A complexity-scalable transcoder from H.264/AVC to the new HEVC codec. In *IEEE International Conference on Image Processing (ICIP 2012)*, pages –, September 2012.

[2] Eduardo Peixoto, Tamer Shanableh, and Ebroul Izquierdo. A H.264/AVC to HEVC transcoder based on content modeling. 2012, in preparation.

[3] Eduardo Peixoto, Toni Zgaljic, and Ebroul Izquierdo. H.264/AVC to wavelet-based scalable video transcoding supporting multiple coding configurations. In *Picture Coding Symposium (PCS 2010)*, pages 562 –565, December 2010.

[4] Eduardo Peixoto, Toni Zgaljic, and Ebroul Izquierdo. Transcoding from H.264/AVC to a wavelet-based scalable video codec. In *IEEE International Conference on Image Processing (ICIP 2010)*, pages 2845 –2848, September 2010.

[5] Eduardo Peixoto, Toni Zgaljic, and Ebroul Izquierdo. Application of large macroblocks in H.264/AVC to wavelet-based scalable video transcoding. In *European Signal Processing Conference (EUSIPCO 2011)*, pages 2171 –2175, August 2011.

[6] Eduardo Peixoto, Toni Zgaljic, and Ebroul Izquierdo. Transcoding from hybrid non-scalable to wavelet based scalable video codecs. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(4):502–515, April 2012.

[7] Tamer Shanableh, Eduardo Peixoto, and Ebroul Izquierdo. MPEG-2 to HEVC video transcoding with content-based modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 2012, submitted.

References

[8] N. Adami, M. Brescianini, M. Dalai, R. Leonardi, and A. Signoroni. A fully scalable video coder with inter-scale wavelet prediction and morphological coding. In *Visual Communications and Image Processing (VCIP 2005)*, page 59601M. SPIE, July 2005.

[9] I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang. Video transcoding: An overview of various techniques and research issues. *IEEE Transactions on Multimedia*, 7(5):793–804, October 2005.

[10] H. Al-Muscati and F. Labeau. Temporal transcoding of H.264/AVC video to the scalable format. In *International Conference on Image Processing Theory Tools and Applications (IPTA 2010)*, pages 138 –143, July 2010.

[11] Y. Andreopoulos, A. Munteanu, J. Barbarien, M. Van der Schaar, J. Cornelis, and P. Schelkens. In-band motion compensated temporal filtering. *Signal Processing: Image Communication, Special Issue on Subband/Wavelet Interframe Video Coding*, 19(7):653 – 673, August 2004.

[12] K. Assaleh and H. Al-Nashash. A novel technique for the extraction of fetal ECG using polynomial networks. *IEEE Transactions on Biomedical Engineering*, 52(6):1148 –1152, June 2005.

[13] K. Assaleh and T. Shanableh. Robust polynomial classifier using L1-norm minimization. *Applied Intelligence*, 33:330–339, 2010. 10.1007/s10489-009-0169-8.

[14] P. A. A. Assuncao and M. Ghanbari. A frequency-domain video transcoder for dynamic bitrate reduction of MPEG-2 bit streams. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(8):953–967, December 1998.

[15] P.A.A. Assuncao and M. Ghanbari. Transcoding of MPEG-2 video in the frequency domain. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 1997)*, volume 4, pages 2633 –2636 vol.4, April 1997.

[16] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[17] E. Barrau. MPEG video transcoding to a fine-granular scalable format. In *IEEE International Conference on Image Processing (ICIP 2002)*, volume 1, pages I–717 – I–720 vol.1, 2002.

[18] J. Bialkowski, M. Barkowsky, and A. Kaup. Overview of low-complexity video transcoding from H.263 to H.264. In *IEEE International Conference on Multimedia and Expo (ICME 2006)*, pages 49 –52, July 2006.

[19] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2 edition, 2007.

[20] G. Bjontegaard. Improvements of the BD-PSNR model. Technical Report VCEG-AI11, ITU-T Telecommunications Standardization Sector, Video Coding Experts Group (VCEG), July 2008.

[21] N. Bjork and C. Christopoulos. Transcoder architectures for video coding. *IEEE Transactions on Consumer Electronics*, 44(1):88–98, February 1998.

[22] T. Borer and T. Davies. Dirac video compression using open standards. White Paper WHP 117, BBC R&D, September 2005.

[23] J.M. Boyce. Weighted prediction in the H.264/MPEG AVC video coding standard. In *International Symposium on Circuits and Systems (ISCAS 2004)*, volume 3, pages III – 789–92 Vol.3, May 2004.

[24] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, and T. Wiegand. WD4: Working draft 4 of high-efficiency video coding. Technical Report JCTVC-F803, Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, July 2011.

[25] W.M. Campbell, K.T. Assaleh, and C.C. Broun. Speaker recognition with polynomial classifiers. *IEEE Transactions on Speech and Audio Processing*, 10(4):205 –212, May 2002.

[26] S.-F. Chang and D.G. Messerschmitt. Manipulation and composition of MC-DCT compressed video. *IEEE Journal on Selected Areas in Communications*, 13(1):1–11, January 1995.

[27] G. Chen, S. Lin, and Y. Zhang. A fast coefficients convertion method for the transform domain MPEG-2 to H.264 transcoding. In *International Conference on Digital Telecommunications (ICDT 2006)*, page 17, August 2006.

[28] G. Chen, S. Lin, Y. Zhang, and G. Cao. An new coefficients transform matrix for the transform domain MPEG-2 to H.264/AVC transcoding. In *IEEE International Conference on Multimedia and Expo (ICME 2006)*, pages 321 –324, July 2006.

[29] P. Chen and J.W. Woods. Bidirectional MC-EZBC with lifting implementation. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(10):1183 – 1194, October 2004.

[30] Z. Chen, S. Haykin, J. J. Eggermont, and S. Becker. *Correlative Learning: A Basis for Brain and Adaptive Systems (Adaptive and Learning Systems for Signal Processing, Communications and Control Systems)*. Wiley-Blackwell, 1 edition, 2007.

[31] S.-J. Choi and J. W. Woods. Motion-compensated 3-D subband coding of video. *IEEE Transactions on Image Processing*, 8:155–167, February 1999.

[32] T. Chujoh, A. Tanizawa, and T. Yamakage. Adaptive loop filter for improving coding efficiency. Technical Report C402, ITU-T SG16, April 2008.

[33] J.D. Cock, S. Notebaert, P. Lambert, and R. V. de Walle. Architectures for fast transcoding of H.264/AVC to quality-scalable SVC streams. *IEEE Transactions on Multimedia*, 11(7):1209–1223, November 2009.

[34] T. A. da Fonseca, R. L. de Queiroz, and D. Mukherjee. Complexity-scalable H.264/AVC in an IPP-based video encoder. In *IEEE International Conference on Image Processing (ICIP 2010)*, pages 2885 –2888, September 2010.

[35] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications*, 4:247–269, 1998.

[36] R. L. de Queiroz and T. D. Tran. *Lapped Transforms for Image Compression, Chapter 5 in The Handbook on Transforms and Data Compression*. San Diego, CA : Academic Press, 2001.

[37] A. J. Diaz-Honrubia, J. L. Martinez, and P. Cuenca. Hevc: A review, trends and challenges. In *Workshop on Multimedia Data Coding and Transmission (WMDCT 2012)*, September 2012.

[38] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2 edition, 2000.

[39] A. Eleftheriadis and D. Anastassiou. Constrained and general dynamic rate shaping of compressed digital video. In *IEEE International Conference on Image Processing (ICIP 1995)*, volume 3, pages 396–399, October 1995.

[40] N. Feamster and S. Wee. An MPEG-2 to H.263 transcoder. In *International Symposium in Voice, Video and Data Communications*. SPIE, September 1999.

[41] G. Fernandez-Escribano, P. Cuenca, L. O. Barbosa, and H. Kalva. Very low complexity MPEG-2 to H.264 transcoding using machine learning. In *ACM International Conference on Multimedia (ACM Multimedia 2006)*, pages 931–940. ACM, 2006.

[42] G. Fernandez-Escribano, H. Kalva, J.L. Martinez, P. Cuenca, L. Orozco-Barbosa, and A. Garrido. An MPEG-2 to H.264 video transcoder in the baseline profile. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(5):763 –768, May 2010.

[43] M. Flierl and B. Girod. Generalized B pictures and the draft H.264/AVC video-compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):587 – 597, July 2003.

[44] C.-M. Fu, C.-Y. Chen, Y.-W. Huang, and S. Lei. Sample adaptive offset for HEVC. In *IEEE International Workshop on Multimedia Signal Processing (MMSP 2011)*, pages 1 –5, October 2011.

[45] K.-T. Fung, Y.-L. Chan, and W.-C. Siu. New architecture for dynamic frame-skipping transcoder. *IEEE Transactions on Image Processing*, 11(8):886–900, August 2002.

[46] K.-T. Fung and W.-C. Siu. Low complexity H.263 to H.264 video transcoding using motion vector decomposition. In *IEEE International Symposium on Circuits and Systems (ISCAS 2005)*, pages 908 – 911 Vol. 2, May 2005.

[47] R. Garrido-Cantos, J. De Cock, J.L. Martinez, S. Van Leuven, and P. Cuenca. Motion-based temporal transcoding from H.264/AVC-to-SVC in baseline profile. *IEEE Transactions on Consumer Electronics*, 57(1):239 –246, February 2011.

[48] J. D. Gibson, T. Berger, T. Lookabaugh, R. Baker, and D. Lindbergh. *Digital Compression for Multimedia: Principles and Standards*. Morgan Kaufmann Publishers, 2006.

[49] B. Girod. Motion-compensating prediction with fractional-pel accuracy. *IEEE Transactions on Communications*, 41(4):604–612, April 1993.

[50] HEVC Test Sequences.
`ftp://hvc:US88Hula@ftp.tnt.uni-hannover.de/testsequences`.
Accessed: 19/10/2012.

[51] C. Holder and H. Kalva. H.263 to VP-6 video transcoder. In *Visual Communications and Image Processing (VCIP 2008)*, volume 6822, pages 68222B–68222B–4. SPIE, January 2008.

[52] C. Holder, T. Pin, and H. Kalva. Improved machine learning techniques for low complexity MPEG-2 to H.264 transcoding using optimized codecs. In *International Conference on Consumer Electronics (ICCE 2009)*, pages 1 –2, January 2009.

[53] S. T. Hsiang. *Highly scalable subband / wavelet image and video coding*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, January 2002.

[54] Y.-W. Huang, B.-Y. Hsieh, S.-Y. Chien, S.-Y. Ma, and L.-G. Chen. Analysis and complexity reduction of multiple reference frames motion estimation in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(4):507 – 522, April 2006.

[55] J.-H. Hur and Y.-L. Lee. H.264 to MPEG-4 transcoding using block type information. In *TENCON 2005 IEEE Region 10*, pages 1–6, November 2005.

[56] Q. Huynh-Thu and M. Ghanbari. Scope of validity of PSNR in Image/Video quality assessment. *Electronics Letters*, 44(13):800–801, June 2008.

[57] J. N. Hwang, T. D. Wu, and C. W. Lin. Dynamic frame-skipping in video transcoding. In *IEEE Workshop on Multimedia Signal Processing*, pages 616–621, December 1998.

[58] VCEG-AM91 ISO/ IEC JTC1/SC29 /WG11/ N11113, ITU-T Q6/16 document. Joint call for proposals on video compression technology. Technical Report VCEG-AM91, ITU-Telecommunications Standardization Sector, Video Coding Experts Group (VCEG), January 2010.

[59] ISO/IEC. ISO/IEC 13818 - Generic coding of moving pictures and associated audio information. Technical report, ISO/IEC, February 1996.

[60] S. Issa and O.O. Khalifa. Performance analysis of Dirac video codec with H.264/AVC. In *International Conference on Computer and Communication Engineering (ICCCE 2010)*, pages 1–6, May 2010.

[61] ITU-T. ITU-T Recommendation H.263, Video coding for low bit rate communication. Technical report, ITU-T, March 1996.

[62] ITU-T. ITU-T Recommendation H.264, Advanced video coding for generic audiovisual services. Technical report, ITU-T, May 2003.

[63] ITU-T. ITU-T Recommendation H.264, advanced video coding for generic audiovisual services. Technical report, ITU-T, March 2005.

[64] ITU-T. Joint model (JM), H.264/AVC reference software. Technical Report JM 14.2 KTA 1.0, Artech House Inc., 2008.

[65] ITU-T. ITU-T Recommendation BT.500-12 methodology for the subjective assessment of the quality of television pictures. Technical report, ITU-T, 2009.

[66] ITU-T and ISO/IEC. ITU-T and ISO/IEC Recommendation H.264 14496-10, Advanced video coding for generic audiovisual services. Technical report, ITU-T and ISO/IEC, November 2007.

[67] J. Jain and A. Jain. Displacement measurement and its application in interframe image coding. *IEEE Transactions on Communications*, 29(12):1799 – 1808, 1981.

[68] M. Karczewicz and R. Kurceren. The SP- and SI-frames design for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):637 – 644, July 2003.

[69] I.-K. Kim, W.-J. Han, J. H. Park, and X. Z. Zheng. CE2: Test results of asymmetric motion partition (AMP). Technical Report JCTVC-F379, Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, July 2011.

[70] P. Kunzelmann and H. Kalva. Reduced complexity H.264 to MPEG-2 transcoder. In *International Conference on Consumer Electronics (ICCE 2007)*, pages 1–2, January 2007.

[71] T.-K. Lee, C.-H. Fu, Y.-L. Chan, and W.-C. Siu. A new motion vector composition algorithm for fast-forward video playback in H.264. In *IEEE International Symposium on Circuits and Systems (ISCAS 2010)*, pages 3649–3652, June 2010.

[72] Y.-K. Lee, S.-S. Lee, and Y.-L. Lee. MPEG-4 to H.264 transcoding using macroblock statistics. In *IEEE International Conference on Multimedia and Expo (ICME 2006)*, pages 57 –60, July 2006.

[73] D. Lefol, D. Bull, N. Canagarajah, and F. Rovati. Performance evaluation of transcoding algorithms for H.264. In *International Conference on Consumer Electronics (ICCE 2006)*, pages 415 – 416, January 2006.

[74] D. Lelescu and D. Schonfeld. Statistical sequential analysis for real-time video scene change detection on compressed multimedia bitstream. *IEEE Transactions on Multimedia*, 5(1):106 – 117, March 2003.

[75] B. Li, G. J. Sullivan, and J. Xu. Comparison of compression performance of HEVC working draft 4 with AVC high profile. Technical Report Doc. JCTVCG399, Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, November 2011.

[76] M. Li and B. Wang. Hybrid video transcoder for bitrate reduction of H.264 bit streams. In *International Conference on Information Assurance and Security (IAS 2009)*, volume 1, pages 107 –110, August 2009.

[77] M. Li, N. Xiong, and C. Ma. A fast H.264 spatial downscaling transcoder for wireless communication. In *International Conference on Computational Science and Engineering (CSE 2009)*, volume 2, pages 932 –936, August 2009.

[78] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz. Adaptive deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology*, 13:614–619, July 2003.

[79] C.-S. Liu, W.-L. Wei, N.-C. Yang, and C.-M. Kuo. Motion vector re-estimation for video trans-coding with arbitrary downsizing. In *International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP 2008)*, pages 806 –809, August 2008.

[80] F. Lonetti and F. Martelli. Motion vector composition algorithm in H.264 transcoding. In *14th International Workshop on Systems, Signals and Image Processing, 2007 and 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services*, pages 401 –404, June 2007.

[81] X. Lu, A.M. Tourapis, P. Yin, and J. Boyce. Fast mode decision and motion estimation for H.264 with a focus on MPEG-2/H.264 transcoding. In *IEEE International Symposium on Circuits and Systems (ISCAS 2005)*, pages 1246 – 1249 Vol. 2, May 2005.

[82] S. Ma, W. Gao, and Y. Lu. Rate-distortion analysis for H.264/AVC video coding and its application to rate control. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(12):1533 – 1544, December 2005.

[83] H.S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky. Low-complexity transform and quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):598–603, July 2003.

[84] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, July 2003.

[85] J.L. Martinez, G. Fernandez-Escribano, H. Kalva, W.A.C. Fernando, and P. Cuenca. Wyner-Ziv to H.264 video transcoder for low cost video encoding. *IEEE Transactions on Consumer Electronics*, 55(3):1453 –1461, August 2009.

[86] K. McCann, B. Bross, S.-I. Sekiguchi, and W.-J. Han. HM4: High efficiency video coding (HEVC) test model 4 encoder description. Technical Report Doc. JCTVC-F802, Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, July 2011.

[87] N. Merhav. Multiplication-free approximate algorithms for compressed-domain linear operations on images. *IEEE Transactions on Image Processing*, 8(2):247–254, February 1999.

[88] MPEG. ISO/IEC 14496-2:2004 - Information technology - Coding of audio-visual objects - Part 2: Visual. Technical report, MPEG, 2004.

[89] M. Mrak. *Motion Scalability for Video Coding with Flexible Spatio-Temporal Decompositions*. PhD thesis, Queen Mary, University of London, London, January 2007.

[90] M. Mrak, N. Sprljan, and E. Izquierdo. An overview of basic techniques behind scalable video coding. In *International Symposium Electronics in Marine*, pages 597–602, June 2004.

[91] M. Mrak, N. Sprljan, and E. Izquierdo. Motion estimation in temporal subbands for quality scalable motion coding. *Electronics Letters*, 41(19):1050 – 1051, September 2005.

[92] M. Mrak, T. Zgaljic, and E. Izquierdo. Influence of downsampling filter characteristics on compression performance in wavelet-based scalable video coding. *Image Processing, IET*, 2(3):116 –129, June 2008.

[93] A.T. Naman and D. Taubman. Rate-distortion optimized delivery of JPEG2000 compressed video with hierarchical motion side information. In *IEEE International Conference on Image Processing (ICIP 2008)*, pages 2312–2315, October 2008.

[94] A.T. Naman and D. Taubman. Rate-distortion optimized JPEG2000-based scalable interactive video (JSIV) with motion and quantization bin side-information. In *IEEE International Conference on Image Processing (ICIP 2009)*, pages 3081 –3084, November 2009.

[95] A.T. Naman and D. Taubman. Predictor selection using quantization intervals in JPEG2000-based scalable interactive video (JSIV). In *IEEE International Conference on Image Processing (ICIP 2010)*, pages 2897 –2900, September 2010.

[96] T. Nguyen and G. Strang. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 2009.

[97] J.-R. Ohm. Three-dimensional subband coding with motion compensation. *IEEE Transactions on Image Processing*, 3(5):559–571, September 1994.

[98] A. Ortega and K. Ramchandran. Rate-distortion methods for image and video compression. *IEEE Signal Processing Magazine*, 15(6):23 –50, November 1998.

[99] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video coding with H.264/AVC: tools, performance, and complexity. *IEEE Circuits and Systems Magazine*, 4(1):7 – 28, Quarter 2004.

[100] M. Pantoja, H. Kalva, and J.-B. Lee. P-frame transcoding in VC-1 to H.264 transcoders. In *IEEE International Conference on Image Processing (ICIP 2007)*, volume 5, pages V –297 –V –300, October 2007.

[101] E. Peixoto, R. L. de Queiroz, and D. Mukherjee. Mobile video communications using a Wyner-Ziv transcoder. In *Visual Communications and Image Processing (VCIP 2008)*, pages 1069–1079. SPIE, January 2008.

[102] E. Peixoto, R. L. de Queiroz, and D. Mukherjee. A Wyner-Ziv video transcoder. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(2):189 –200, February 2010.

[103] E. Peixoto, R.L. de Queiroz, and D. Mukherjee. Mapping motion vectors for a Wyner-Ziv video transcoder. In *IEEE International Conference on Image Processing (ICIP 2009)*, pages 3681 –3684, November 2009.

[104] K. R. Rao and J. J. Hwang. *Techniques and Standards for Image, Video and Audio Coding*. Prentice Hall PTR, 1996.

[105] K. R. Rao and P. Yip. *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Academic Press, 1 edition, 1990.

[106] I. E. Richardson. *The H.264 Advanced Video Compression Standard*. Wiley, 2 edition, 2010.

[107] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers, 2 edition, 2000.

[108] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103 –1120, September 2007.

[109] K.-D. Seo, S. J. Park, and S.-H. Jung. Wipe scene-change detector based on visual rhythm spectrum. In *International Conference on Consumer Electronics (ICCE 2009)*, pages 1 –2, January 2009.

[110] T. Shanableh. Matrix encoding for data hiding using multilayer video coding and transcoding solutions. *Signal Processing: Image Communication*, (0):–, 2012.

[111] T. Shanableh and K. Assaleh. Feature modeling using polynomial classifiers and stepwise regression. *Neurocomputing*, 73(10-12):1752–1759, June 2010.

[112] T. Shanableh and M. Ghanbari. Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats. *IEEE Transactions on Multimedia*, 2:101–110, June 2000.

[113] T. Shanableh, Tony M., and F. Ishtiaq. Error resiliency transcoding and decoding solutions using distributed video coding techniques. *Signal Processing: Image Communication*, 23(8):610 – 623, 2008.

[114] S. Sharma and K. R. Rao. Transcoding of H.264 bitstream to MPEG-2 bitstream. In *Asia-Pacific Conference on Communications (APCC 2007)*, pages 391–396, October 2007.

[115] D. Shen, I.E. Sethi, and V. Bhaskaran. Adaptive motion vector re-sampling for compressed video downscaling. In *IEEE International Conference on Image Processing (ICIP 1997)*, volume 1, pages 771–774, October 1997.

[116] K. Shen and E. J. Delp. Wavelet based rate scalable video compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(1):109–122, February 1999.

[117] I.-H. Shin, Y.-L. Lee, and H. W. Park. Motion estimation for frame-rate reduction in H.264 transcoding. In *IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, pages 63–67, May 2004.

[118] Y. Shin, N. Son, N. D. Toan, and G. Lee. Low-complexity heterogeneous video transcoding by motion vector clustering. In *International Conference on Information Science and Applications (ICISA 2010)*, pages 1–6, April 2010.

[119] H. Shu and L.-P. Chau. The realization of arbitrary downsizing video transcoding. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(4):540 – 546, April 2006.

[120] A. N. Skodras and C. A. Christopoulos. Down-sampling of compressed images in the DCT domain. In *Proceedings of European Signal Processing Conference (EUSIPCO 1998)*, pages 1713–1716, September 1998.

[121] SMPTE. Standard for television: VC-1 compressed video bitstream format and decoding process. Technical report, SMPTE 421M, 2006.

[122] N. Sprljan. *A Flexible Scalable Video Coding Framework with Adaptive Spatio-Temporal Decompositions*. PhD thesis, Queen Mary, University of London, London, August 2006.

[123] N. Sprljan, M. Mrak, and E. Izquierdo. Motion driven adaptive transform based on wavelet transform for enhanced video coding. In *Proceedings of the 2nd international conference on Mobile multimedia communications*, MobiMedia '06, pages 8:1–8:4, July 2006.

[124] N. Sprljan, M. Mrak, T. Zgaljic, and E. Izquierdo. Software proposal for wavelet video coding exploration group. Technical report, in ISO/IEC JTC1/SC29 /WG11/MPEG2005, M12941, 75-th MPEG Meeting, January 2006.

[125] G. J. Sullivan, P. Topiwala, and A. Luthra. The H.264/AVC advanced video coding standard:overview and introduction to the fidelity range extensions. In *SPIE conference on Applications of Digital Image Processing XXVII*, volume 5558, pages 454–474, August 2004.

[126] G.J. Sullivan and T. Wiegand. Rate-distortion optimization for video compression. *IEEE Signal Processing Magazine*, 15(6):74–90, November 1998.

[127] H. Sun, W. Kwok, and J.W. Zdepski. Architectures for MPEG compressed bitstream scaling. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(2):191–199, April 1996.

[128] H. Sun, A. Vetro, J. Bao, and T. Poon. A new approach for memory efficient ATV decoding. *IEEE Transactions on Consumer Electronics*, 43(3):517–525, August 1997.

[129] Y.-P. Tan and H. Sun. Fast motion re-estimation for arbitrary downsizing video transcoding using H.264/AVC standard. *IEEE Transactions on Consumer Electronics*, 50(3):887 – 894, August 2004.

[130] Q. Tang and P. Nasiopoulos. Efficient motion re-estimation with rate-distortion optimization for MPEG-2 to H.264/AVC transcoding. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(2):262 –274, February 2010.

[131] Test Media Collection. `http://media.xiph.org/video/derf/`.
Accessed: 19/10/2012.

[132] A. M. Tourapis. Enhanced predictive zonal search for single and multiple frame motion estimation. In *Visual Communications and Image Processing (VCIP 2002)*, pages 1069–1079. SPIE, 2002.

[133] A.M. Tourapis, O.C. Au, and M.L. Liou. Highly efficient predictive zonal algorithms for fast block-matching motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(10):934 – 947, October 2002.

[134] H.-Y.C. Tourapis and A.M. Tourapis. Fast motion estimation within the H.264 codec. In *International Conference on Multimedia and Expo (ICME 2003)*, volume 3, pages III – 517–20 vol.3, July 2003.

[135] A. Vetro, C. Christopoulos, and H. Sun. Video transcoding architectures and techniques: An overview. *IEEE Signal Processing Magazine*, 20(2):18–29, March

2003.

[136] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[137] X. H. Wei, S. Li, and S. Goto. A motion vector prediction scheme for MPEG-2 to H.264 transcoding based on smoothness of motion vector field. In *IEEE International Conference on Multimedia and Expo (ICME 2007)*, pages 424 –427, July 2007.

[138] S. Wenger. H.264/AVC over IP. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):645 – 656, July 2003.

[139] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003.

[140] J. Xin, J. Li, A. Vetro, H. Sun, and S.-I. Sekiguchi. Motion mapping for MPEG-2 to H.264/AVC transcoding. In *IEEE International Symposium on Circuits and Systems (ISCAS 2007)*, pages 1991 –1994, May 2007.

[141] J. Xin, C.-W. Lin, and M.-T. Sun. Digital video transcoding. *Proceedings of the IEEE*, 93(1):84–97, January 2005.

[142] J. Xin, A. Vetro, S. Sekiguchi, and K. Sugimoto. Motion and mode mapping for MPEG-2 to H.264/AVC transcoding. In *IEEE International Conference on Multimedia and Expo (ICME 2006)*, pages 313–316, July 2006.

[143] D. Xu and P. Nasiopoulos. Logo insertion transcoding for H.264/AVC compressed video. In *IEEE International Conference on Image Processing (ICIP 2009)*, pages 3693 –3696, November 2009.

[144] S. Yang, D. Kim, Y. Jeon, and J. Jeong. An efficient motion re-estimation algorithm for frame-skipping video transcoding. In *IEEE International Conference on Image Processing (ICIP 2005)*, volume 3, pages III – 668–71, September 2005.

[145] P. Yin, H.-Y.C. Tourapis, A.M. Tourapis, and J. Boyce. Fast mode decision and motion estimation for JVT/H.264. In *IEEE International Conference on Image Processing (ICIP 2003)*, volume 3, pages III – 853–6 vol.2, September 2003.

[146] P. Yin, A. Vetro, B. Liu, and H. Sun. Drift compensation for reduced spatial resolution transcoding. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(11):1009–1020, November 2002.

[147] J. Youn and M.-T. Sun. A fast motion vector composition method for temporal transcoding. In *IEEE International Symposium on Circuits and Systems (ISCAS 1999)*, volume 4, pages 243 –246 vol.4, July 1999.

[148] J. Youn, M.-T. Sun, and C.-W. Lin. Motion estimation for high performance transcoding. *IEEE Transactions on Consumer Electronics*, 44(3):649 –658, August 1998.

[149] J. Youn, M.-T. Sun, and C.-W. Lin. Motion vector refinement for high performance transcoding. *IEEE Transactions on Multimedia*, 1(1):30–40, March 1999.

[150] T. Zgaljic. *Entropy Coding Schemes for Scalable Video Coding Supporting Flexible Bit-stream Organisation and Adaptation.* PhD thesis, Queen Mary, University of London, London, June 2008.

[151] T. Zgaljic, N. Sprljan, and E. Izquierdo. Bit-stream allocation methods for scalable video coding supporting wireless communications. *Signal Processing: Image Communication, Special issue on Mobile Video*, 22(3):298 – 316, 2007.

[152] P. Zhang, Y. Lu, Q. Huang, and W. Gao. Mode mapping method for H.264/AVC spatial downscaling transcoding. In *IEEE International Conference on Image Processing (ICIP 2004)*, volume 4, pages 2781 – 2784 Vol. 4, October 2004.

[153] C. Zhu, X. Lin, L.-P. Chau, K.-P. Lim, H.-A. Ang, and C.-Y. Ong. A novel hexagon-based search algorithm for fast block motion estimation. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001)*, volume 3, pages 1593–1596, June 2001.

[154] S. Zhu and K.-K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing*, 9(2):287 –290, February 2000.

Appendix A.

Fast Motion Estimation Methods

This appendix describes two fast motion estimation algorithms used in the H.264/AVC to W-SVC transcoder. The goal of these algorithm is to return the motion vector that yields the lowest cost, in rate-distortion sense, for a particular partition. This motion vector is usually called the *best motion vector*. All algorithms described here are used in the same way regardless of the partition size. Two popular methods are considered: the Hexagon search [153] and the Enhanced Predictive Zonal Search (EPZS) [132]. Both methods are implemented in several reference codecs.

The Hexagon search is a simple method that provides a good performance, compared to full motion estimation, with a fraction of the complexity. On the other hand, there are several implementations of the EPZS algorithm, with different trade-offs of rate-distortion performance and complexity. The EPZS algorithm implemented here performs very close to the full search algorithm, with almost negligible PSNR loss, but its complexity is usually higher than the Hexagon search algorithm by about 50% (thus, it is still much faster than the full search algorithm, roughly by a factor of 100, with the typical search window of 32).

A.1. Hexagon Search

This method consists of two steps. The first step uses a hexagonal shape, testing motion vectors that are not immediate neighbours of the center motion vector, in an attempt to avoid local minima. The second step tests the immediate neighbours of the best motion vector found in the first iteration, in an attempt to refine the prediction. The search

starts on a motion vector candidate $mv_{center} = (x_C, y_C)$, which is the first one that is tested. Then, the first step works as follows:

(1) Tests six candidates around mv_{center}, namely: $(x_C + 2, y_C)$, $(x_C + 2, y_C + 1)$, $(x_C + 2, y_C - 1)$, $(x_C - 2, y_C)$, $(x_C - 2, y_C - 1)$, $(x_C - 2, y_C + 1)$.

(2) If mv_{center} is the one that yields the lowest cost, stop the first step. Otherwise, repeat the search (1) with mv_{center} as the one that yields the lowest cost.

Note that the first iteration of the first step tests six motion vector candidates, but subsequent iterations only need to test 3 candidates, as the other 3 were tested in the previous iteration. Once the first iteration stops, the second iteration tests four candidates around the last mv_{center}, namely: $(x_C + 1, y_C)$, $(x_C, y_C + 1)$, $(x_C - 1, y_C)$, $(x_C, y_C - 1)$. The two steps are shown in Fig. A.1.

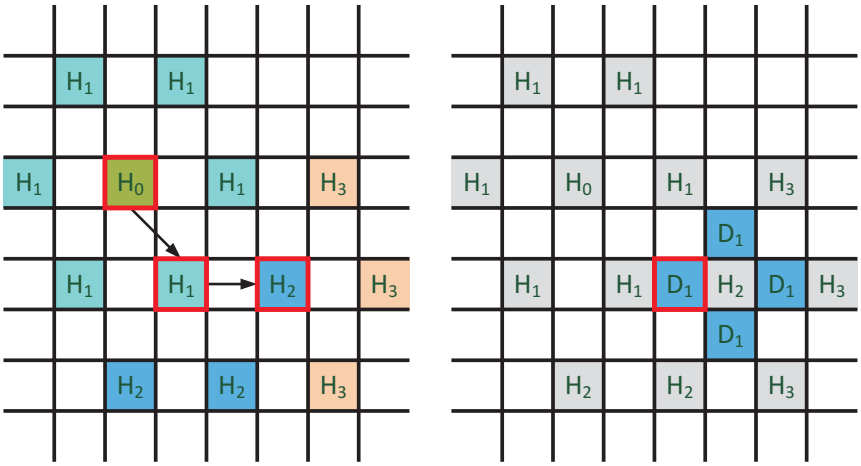


Figure A.1.: Example of Hexagon search: (a) first step; and (b) second step (diamond refinement). In the figure, each step is labeled as H_n, which represents the motion vectors tested in the $n - th$ iteration of the Hexagon search, or D_1, which represents the motion vectors tested in the diamond refinement. The highlighted motion vectors indicate the best motion vector for each iteration.

A.1.1. Sub-pixel search

The sub-pixel search used here is one of the most popular sub-pixel refinements. For each level of refinement (usually, only half-pixel and quarter-pixel are used), the algorithm tests the eight neighbours of the best motion vector on the previous level (the integer

level being the previous level for the half-pixel level). Then, it proceeds to the next level, or returns the best motion vector if the final level has been reached.

A.2. EPZS Implementation on W-SVC

The Enhanced Predictive Zonal Search (EPZS) algorithm is very popular, and it is implemented in several reference codecs, such as the H.264/AVC reference JM [64] and the HEVC reference software HM [86]. It is capable of achieving results very close to the full search algorithm, at a fraction of the cost. Due to its immense popularity, there are several versions of this algorithm [132, 133, 134, 145]. Also, the versions found in the reference softwares usually differ from these works, incorporating new advances in the algorithm to tackle the characteristics of that codec.

In order to fully evaluate the H.264/AVC to W-SVC transcoder, the EPZS algorithm was implemented in the W-SVC, to be used as a benchmark. The algorithm implemented is based on the algorithm found in the JM 14.2 [64]. The algorithm was implemented with all the default settings, regarding the predictors and the thresholds. Also, the algorithm had to be slightly modified, in order to account for the differences of the W-SVC codec. The implemented algorithm is described next.

The EPZS algorithm implemented is based on generating several motion vector candidates, also called predictors, in an attempt to sample the search window. Several motion vector prediction techniques are used to generate these predictors, and the results of each technique are called a *subset* of predictors. Then, early stop decision algorithm are used in an attempt to stop the search once a suitable candidate has been found. The following sections describes the subsets, and afterwards the algorithm is discussed.

A.2.1. Subset A: Median Motion Vector

The first subset consists only in the median motion vector. This is the same motion vector that is used in the H.264/AVC codec to encode the motion vectors (called the motion vector predictor, in that codec). To compute the median motion vector, three neighbours of the current block are considered: the "block to the left", "block to the top" and "block to the top-right" (namely, A, B and C in Fig. A.2). If block C is not available, a "block to the top-left" (namely D in the figure) is used instead. Note

that, despite the size of the neighbouring partitions, only the motion vector at these partitions are used to compute the median (for example, only the motion vector for the B neighbour is used for the "top" block, the remaining motion vectors are ignored). If any of the others are not available, the algorithm uses only the remaining blocks. The motion vector generated by this method is the median between the three blocks.

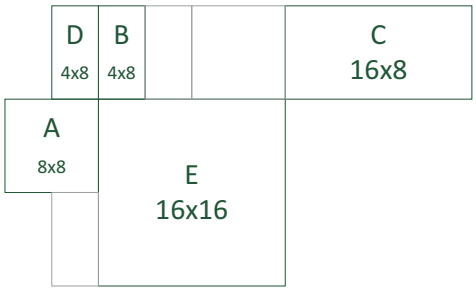


Figure A.2.: Example of the neighbouring partitions used for computing the Median approximation.

A.2.2. Subset B: Spatial Predictors

This subset always consists of five motion vectors and it is formed of motion vectors from the neighbouring blocks from the current block. It uses the motion vectors that were actually used to encode those blocks. Note that only the motion vectors used by the 4×4 blocks are used, regardless of the size of the current block or how the neighbouring blocks are partitioned.

1) The $(0, 0)$ motion vector.

2) The motion vector of the block immediately to the left (namely A in Fig. A.3), or $(0, 3)$, if A is not available.

3) The motion vector of the block immediately above (namely B in Fig. A.3), or $(3, 0)$, if B is not available.

4) The motion vector of the block immediately to the top-right (namely C in Fig. A.3), or $(0, -3)$, if C is not available.

5) The motion vector of the block immediately to the top-left (namely D in Fig. A.3), or $(-3, 0)$, if D is not available.

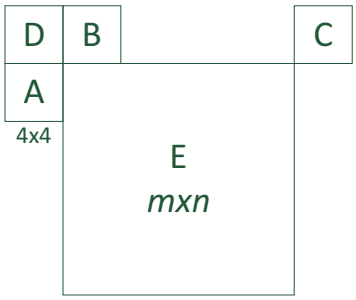


Figure A.3.: Location of the Spatial predictors for a given block.

A.2.3. Subset C: Spatial Memory Predictors

These predictors come from the the neighbouring blocks of *the same size* as the current block. The spatial memory predictors consists of up to three motion vectors:

1) The motion vector chosen for the partition to the left (namely A in Fig. A.4).

2) The motion vector chosen for the partition above (namely B in Fig. A.4).

3) The motion vector chosen for the partition to the top-right (namely C in Fig. A.4).

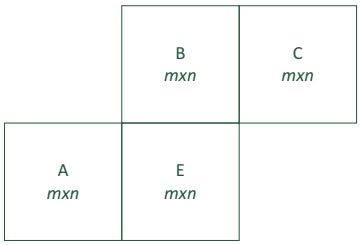


Figure A.4.: Location of the Spatial Memory predictors for a given block.

If one of these motion vectors are not available, no other motion vector is used in its place. Also, it is important to note that, in this case, the algorithm keeps the motion vector that was chosen to a partition that might not have been used. For example, suppose that the current partition is 16×16. Then, the algorithm will take the motion vector that was chosen for the 16×16 partition of the neighbouring macroblocks, even if these macroblocks were encoded using a different partitioning.

A.2.4. Subset D: Temporal Predictors

The temporal predictors come from the co-located blocks from the previous frame. This had to be adapted to work on the W-SVC codec, due to its characteristics. First, due

to the hierarchical coding configuration used in the W-SVC codec, it is possible that motion estimation has not been performed on the immediately previous frame at the time the current frame n is being encoded. To solve this issue, the closest previous frame to which motion estimation has been performed is used as the previous frame. Note that this is always either the frame $n-1$ (for the even frames) or the frame $n-2$ (for the odd frames), since motion estimation is always performed on the odd frames first (in the first level of temporal decomposition).

Another issue is that the used previous frame $n-\gamma$ may have a different distance to its reference frame than the current frame n. To solve this problem, the motion vectors are scaled, using the following equation:

$$mv^k_{n\to n-\beta} = \left(\frac{\beta}{\gamma-\alpha}\right) \cdot mv^k_{n-\gamma\to n-\alpha} \tag{A.1}$$

where $n-\beta$ is the reference frame for the current frame n and $n-\alpha$ is the reference frame for the previous frame $n-\gamma$. This solution is similar to the one used in the JM reference software when different coding configurations are used. However, in the JM software, several coding configurations can be used, while for the W-SVC a hierarchical structure is always used.

For the temporal predictors, the algorithm uses up to nine motion vectors from co-located blocks in the frame $n-\gamma$. Again, if one of these motion vectors is not available, no other motion vector is used in its place, and the motion vector used for the 4×4 block is considered, regardless of the actual partitioning for that block. These predictors are:

1) The motion vector of the co-located block (namely A in Fig. A.5).
2) The motion vector immediately to the left (namely B in Fig. A.5).
3) The motion vector immediately to the top-left (namely C in Fig. A.5).
4) The motion vector immediately above (namely D in Fig. A.5).
5) The motion vector immediately to the bottom-left (namely E in Fig. A.5).
6) The motion vector immediately below (namely F in Fig. A.5).
7) The motion vector immediately to the top-right (namely G in Fig. A.5).

8) The motion vector immediately to the right (namely H in Fig. A.5).

9) The motion vector immediately to the bottom-right (namely I in Fig. A.5).

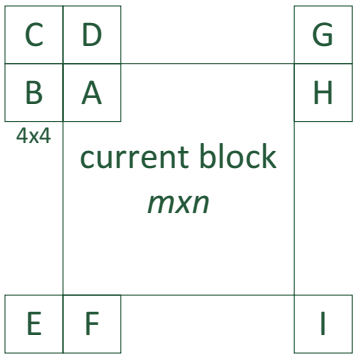


Figure A.5.: Location of the Temporal Predictors for a given block.

A.2.5. Subset E: Window Based Predictors

The Windows Based Predictors creates several motion vectors following a pattern around the median motion vector. The main purpose is to sample the search window, trying to avoid local minima. It generates predictors using the equation:

$$mv_{candidate} = mv_{median} + mv_{window} \tag{A.2}$$

In the JM implementation, and in the W-SVC implementation, the default window uses the following motion vectors, in integer pixel scale:

$$\begin{bmatrix} (0,4) & (0,8) & (0,16) & (0,32) \\ (4,4) & (8,8) & (16,16) & (32,32) \\ (4,0) & (8,0) & (16,0) & (32,0) \\ (4,-4) & (8,-8) & (16,-16) & (32,-32) \\ (0,-4) & (0,-8) & (0,-16) & (0,-32) \\ (-4,-4) & (-8,-8) & (-16,-16) & (-32,-32) \\ (-4,0) & (-8,0) & (-16,0) & (-32,0) \\ (-4,4) & (-8,8) & (-16,16) & (-32,32) \end{bmatrix} \tag{A.3}$$

A.2.6. EPZS Thresholds

The algorithm uses two thresholds, defined as:

$$T_1 = m \cdot n \cdot \left(\sqrt{2}\right)^l \tag{A.4}$$

where m and n are the partition dimensions and l denotes the temporal decomposition level of the current frame (starting at $l = 0$). The factor $\left(\sqrt{2}\right)^l$ accounts for the change in the dynamic range of the frame after l temporal decompositions. To compute the second threshold, first a minimum threshold is computed:

$$T_{min} = min\left(MSAD_A, MSAD_B, MSAD_C, 3 \cdot T_1\right) \tag{A.5}$$

where $MSAD_A$ means the minimum SAD for the partition of the same size immediately to the left, $MSAD_B$ means the minimum SAD for the partition of the same size immediately above, and $MSAD_C$ means the minimum SAD for the partition of the same size immediately to the top-right. Note that it uses the partition of the same size as the current partition, even if this partition was not chosen by the rate distortion engine for that partition. The factor $3 \cdot T_1$ is considered in order to avoid setting this threshold too high. Finally, the second threshold is computed as:

$$T_2 = \frac{9 \cdot max\left(T_{min}, T_1\right) + 2 \cdot T_1}{8} \tag{A.6}$$

A.2.7. Refinement on the EPZS: Diamond Search

The Diamond search [154] is one of the most popular fast motion estimation algorithms, mainly due to its simplicity. It is the precursor to the Hexagon search shown in Sec. A.1. The search starts on a motion vector candidate $mv_{center} = (x_C, y_C)$, which is the first one that is tested. Then, the search works as follows:

(1) Tests four candidates around mv_{center}, namely: $(x_C + 1, y_C)$, $(x_C, y_C + 1)$, $(x_C - 1, y_C)$, $(x_C, y_C - 1)$.

(2) If mv_{center} is the one that yields the lowest cost, stop the search. Otherwise, repeat the search (1) with mv_{center} as the one that yields the lowest cost. Note that each subsequent step needs only to test up to three motion vector candidates.

An example of the Diamond search algorithm is shown in Fig. A.6.

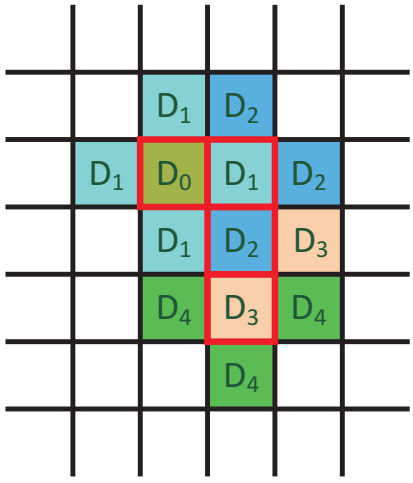


Figure A.6.: Example of Diamond search. In the figure, each step is labeled as D_n, which represents the motion vectors tested in the $n - th$ iteration of the Diamond search. The highlighted motion vectors indicate the best motion vector for each iteration. Note that, for the third iteration, only two motion vectors need to be tested.

A.2.8. The Algorithm

All motion vectors from Subsets A to E are rounded to integer precision before they are used in the algorithm, which works as follows. The SAD_{best} is the lowest SAD computed up to that point in the algorithm.

1) Test the motion vectors from Subset A (i.e., the median motion vector).
2) If ($SAD_{best} < T_1$), then terminates the algorithm.
3) Otherwise, test all motion vectors from Subsets B to E.
4) If ($SAD_{best} < T_2$), then terminates the algorithm.
5) Otherwise, perform a small Diamond search (see Sec. A.2.7) starting on the best candidate tested so far.

A.2.9. Sub-pixel search

The sub-pixel search also makes use of a threshold, defined as:

$$T_S = 2 \cdot m \cdot n \cdot \left(\sqrt{2}\right)^l \tag{A.7}$$

The sub-pixel search algorithm is the same for each level (half-pixel, then quarter-pixel, and so on). At each level, the nine motion vector candidates (i.e., the center, which is the best motion vector from the previous level, and its eight neighbours at this sub-pixel level) are labeled as shown in Fig. A.7.

First, the algorithm tests five motion vectors: A, B, C, D and E, and it keeps track of both the lowest SAD and the second lowest SAD. After testing the first five motion vectors, the algorithm checks its early stop criteria only if the best motion vector is the same as the predicted motion vector (the one that will be used to encode the motion vectors). This early stop criteria is if the lowest SAD is lower than T_S. If this criteria is met, the algorithm stops completely and does not test any remaining level, returning the best motion vector.

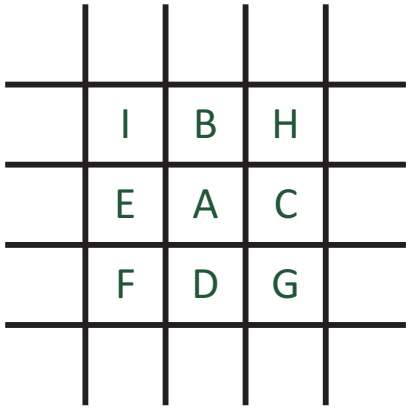


Figure A.7.: Labels for EPZS sub-pixel search.

Table A.1.: Sub-pixel positions tested according to the two lowest SAD.

Position of the two lowest SAD	Positions to test
$\{B, C\}$	$\{H\}$
$\{C, D\}$	$\{G\}$
$\{D, E\}$	$\{F\}$
$\{E, B\}$	$\{I\}$
$\{A, B\}$	$\{I, H\}$
$\{A, C\}$	$\{H, G\}$
$\{A, D\}$	$\{G, F\}$
$\{A, E\}$	$\{F, I\}$
otherwise	$\{F, G, H, I\}$

If the algorithm continues, it uses the positions of the two lowest SADs to decide which positions it will test next. Note that the order is irrelevant. The idea is to avoid testing unlikely positions, based on the positions of the two lowest SADs.

After the best motion vector for this sub-pixel level is found, the algorithm repeats itself for the next level, or returns the best motion vector, if the final level has been reached.

Appendix B.

Coding of a Macroblock in the H.264/AVC Codec

This appendix contains the mathematical representation of the coding sequence of a macroblock in the H.264/AVC codec. In this appendix, just the lossy part of the coding algorithm is explained. Also, just the encoding sequence is shown, as the main part of the decoding sequence is contained in the encoding sequence.

B.1. Encoding Sequence

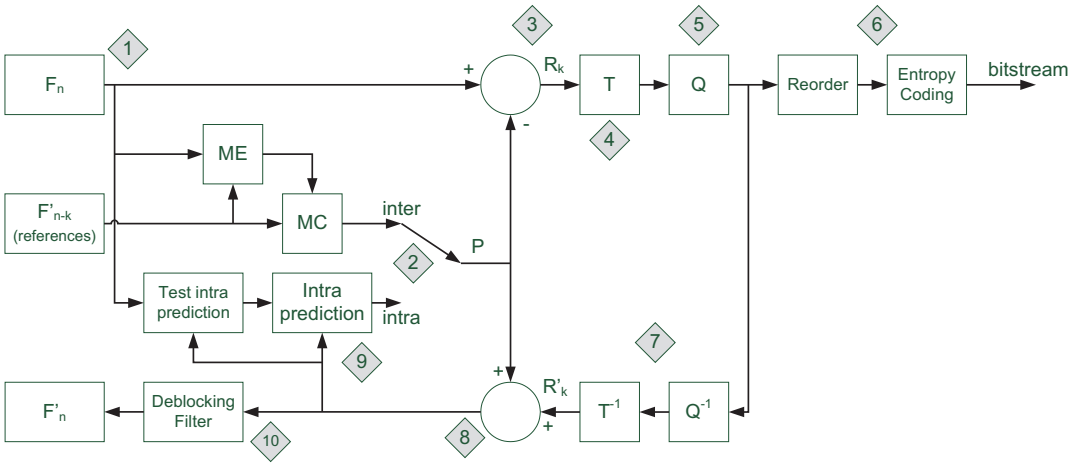


Figure B.1.: H.264/AVC encoder diagram.

The encoding flow of the H.264/AVC is shown in Fig. B.1. Consider the coding of a particular macroblock B_n^k (the k-th macroblock of the n-th frame f_n) of size 16×16. This step is shown as the step 1 in Fig. B.1.

$$B_n^k = \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,15} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,15} \\ \vdots & \vdots & \ddots & \vdots \\ b_{15,0} & b_{15,1} & \cdots & b_{15,15} \end{bmatrix} \tag{B.1}$$

where $b_{i,j}$, $0 \leq i < 16$ and $0 \leq j < 16$ is the luminance sample at position (i, j) within the macroblock. In typical video encoders, including the H.264/AVC, the encoding of the chroma components follows a similar fashion as the luma components, and therefore the analysis here is restricted to the luma component.

The next step in coding the block B_n^k is to decide which kind of prediction will be used to encode this block, shown as the step 2 in Fig. B.1. This prediction may be formed by samples of a previously encoded frame (in case of inter prediction) or by already encoded samples in the current frame (in case of intra prediction). Either way, the prediction for B_n^k is a matrix of prediction samples P_n^k of the same size as B_n^k:

$$P_n^k = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,15} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,15} \\ \vdots & \vdots & \ddots & \vdots \\ p_{15,0} & p_{15,1} & \cdots & p_{15,15} \end{bmatrix} \tag{B.2}$$

After deciding on the prediction, the residual, shown as step 3 in Fig. B.1, is computed as:

$$R_n^k = B_n^k - P_n^k = \begin{bmatrix} b_{0,0} - p_{0,0} & b_{0,1} - p_{0,1} & \cdots & b_{0,15} - p_{0,15} \\ b_{1,0} - p_{1,0} & b_{1,1} - p_{1,1} & \cdots & b_{1,15} - p_{1,15} \\ \vdots & \vdots & \ddots & \vdots \\ b_{15,0} - p_{15,0} & b_{15,1} - p_{15,1} & \cdots & b_{15,15} - p_{15,15} \end{bmatrix} \tag{B.3}$$

Note that, for the special case of B_0^0 (i.e., the first block of the first frame, when neither inter nor intra prediction are possible), $P_0^0 = \mathbf{0}$, and thus $R_0^0 = B_0^0$.

The next step is the transform. The H.264/AVC codec defines two transforms, of sizes 4×4 [83] and 8×8 [125]. Both are modified integer DCT transforms. Here, the computations for the 4×4 transform are given. At this point, consider $\mathbf{X}$ to be one of the 4×4 sub-matrices of R_n^k. The computations are the same for the remaining sub-matrices.

In the H.264/AVC, the operations of transform, scaling and quantization are re-arranged in two steps: the first one being the transform, and the second being the scaling and quantization merged as one step. The complete derivation of this process has been extensively detailed elsewhere [106] and it is out of the scope of this analysis. In the first of these steps, shown as the step 4 in Fig. B.1, the transform matrix used is given as:

$$\mathbf{C_{f4}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \tag{B.4}$$

The transform is then applied as:

$$T(\mathbf{X}) = \mathbf{C_{f4}} \cdot \mathbf{X} \cdot \mathbf{C_{f4}}^T \tag{B.5}$$

where $T(\mathbf{X})$ are the transformed coefficients. In order to perform the quantization step, shown as the step 5 in Fig. B.1, the standard defines a matrix $\mathbf{M_{f4}}$, with values that depend on the quantization parameter QP chosen to encode this macroblock. The matrix $\mathbf{M_{f4}}$ is defined as:

Table B.1.: Values of $m(r,n)$, as defined by the standard.

r	$m(r,0)$	$m(r,1)$	$m(r,2)$
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

$$\mathbf{M_{f4}} = \begin{bmatrix} m(QP\%6,0) & m(QP\%6,2) & m(QP\%6,0) & m(QP\%6,2) \\ m(QP\%6,2) & m(QP\%6,1) & m(QP\%6,2) & m(QP\%6,1) \\ m(QP\%6,0) & m(QP\%6,2) & m(QP\%6,0) & m(QP\%6,2) \\ m(QP\%6,2) & m(QP\%6,1) & m(QP\%6,2) & m(QP\%6,1) \end{bmatrix} \tag{B.6}$$

where QP is the quantization parameter, % denotes the remainder after division and the value of $m(r,n)$ is given in Table B.1. Naturally, the value of $QP\%6$ ranges from 0 to 5, which are the values of r shown in the table.

Finally, the complete forward transform, scaling and quantization process for the 4×4 block $\mathbf{X}$ is given as:

$$\mathbf{Y} = \left\lfloor [\mathbf{C_{f4}}] \cdot [\mathbf{X}] \cdot \left[\mathbf{C_{f4}}^T\right] \bullet [\mathbf{M_{f4}}] \cdot \frac{1}{2^{15+\lfloor QP/6 \rfloor}} \right\rceil \tag{B.7}$$

where $\bullet$ denotes element by element multiplication, $\lfloor x \rfloor$ denotes the largest integer not greater than x (the *floor* operator) and $\lfloor x \rceil$ denotes the integer closest to x (i.e., the *round* operator). Naturally, since this operation involves quantization, the block $\mathbf{X}$ cannot generally be perfectly reconstructed from $\mathbf{Y}$, due to the quantization error.

After applying Eq. B.7 to all 4×4 blocks within R_n^k, the quantized coefficients (i.e., the collection of matrices $\mathbf{Y}$ for each 4×4 block) are reordered and entropy encoded, shown as step 6 in Fig. B.1. The H.264/AVC defines two entropy encoders, the Context

Table B.2.: Values of $v\,(r, n)$, as defined by the standard.

r	$v\,(r, 0)$	$v\,(r, 1)$	$v\,(r, 2)$
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Adaptive Variable Length Coding (CAVLC) and the Context Adaptive Binary Arithmetic Coding (CABAC). Since these operations are lossless, they will not be detailed here, but can be found elsewhere [106].

Before proceeding to the next block, B_n^{k+1}, the encoder runs the so-called reconstruction loop, in which the quantized coefficients $\mathbf{Y}$ are decoded, in order to generate the reconstructed macroblock, which can then be used for intra and inter predictions.

Similar to the forward transform and quantization, the inverse quantization and transform operations are computed in one step, shown as step 7 in Fig. B.1. In order to perform the inverse quantization, the standard defines a matrix $\mathbf{V_{i4}}$, with values that depend on the quantization parameter QP. The matrix $\mathbf{V_{i4}}$ is defined as:

$$\mathbf{V_{i4}} = \begin{bmatrix} v\,(QP\%6, 0) & v\,(QP\%6, 2) & v\,(QP\%6, 0) & v\,(QP\%6, 2) \\ v\,(QP\%6, 2) & v\,(QP\%6, 1) & v\,(QP\%6, 2) & v\,(QP\%6, 1) \\ v\,(QP\%6, 0) & v\,(QP\%6, 2) & v\,(QP\%6, 0) & v\,(QP\%6, 2) \\ v\,(QP\%6, 2) & v\,(QP\%6, 1) & v\,(QP\%6, 2) & v\,(QP\%6, 1) \end{bmatrix} \tag{B.8}$$

where the value of $v\,(r, n)$ is defined by the standard and it is given in Table B.2.

The inverse transform matrix $\mathbf{C_{i4}}$ is given as:

$$\mathbf{C_{i4}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \tag{B.9}$$

and the complete inverse transform and scaling process for the 4×4 block $\mathbf{Y}$ is given as:

$$\mathbf{Z} = \left\lfloor \left[\mathbf{C_{i4}}^T\right] \cdot \left[\mathbf{Y} \bullet \mathbf{V_{i4}} \cdot 2^{\lfloor QP/6 \rfloor}\right] \cdot \left[\mathbf{C_{i4}}\right] \cdot \frac{1}{2^6} \right\rceil \tag{B.10}$$

where $\mathbf{Z}$ is the 4×4 block for the reconstructed residual. After applying Eq. B.10 to all 4×4 blocks in the macroblock, the reconstructed residual $R_n^{'k}$ is formed.

The reconstructed block $B_n^{'k}$ is then formed as:

$$B_n^{'k} = R_n^{'k} + P_n^k \tag{B.11}$$

which is shown as step 8 in Fig. B.1. Note that the prediction P_n^k is the same both in the forward and the reconstruction loops. At the same time, since, in general $R_n^{'k} \neq R_n^k$, then it follows that $B_n^{'k} \neq B_n^k$, due to the quantization errors.

After forming the block $B_n^{'k}$, this block can be used for intra prediction for the subsequent macroblocks, shown in step 9 in Fig. B.1. After all macroblocks are encoded, a deblocking filter [78] is applied to the reconstructed frame in an attempt to reduce the blocking artifacts caused by the encoding. This is shown as step 10 in Fig. B.1. The output of the deblocking filter is the reconstructed frame $f_n^{'}$.

The decoding sequence is shown in Fig. B.2. Note that the decoding sequence is contained in the encoding sequence, as the reconstruction loop (steps 7, 8 and 10, shown in Fig B.1).

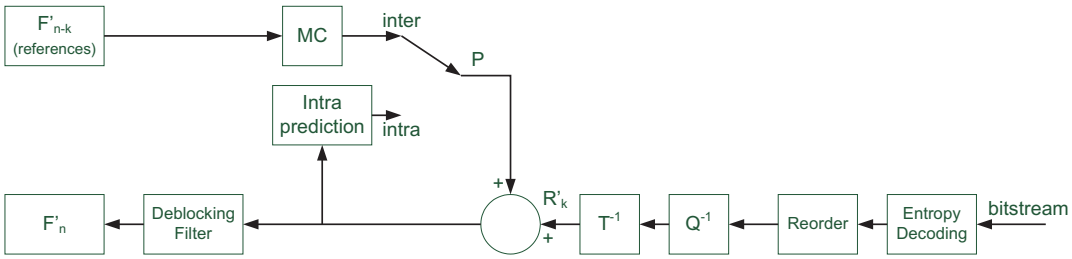


Figure B.2.: H.264/AVC decoder diagram.

Appendix C.

Video Sequences

Throughout this thesis, different test sequences have been used. All of them are popular sequences widely found in the literature. This appendix shows a sample frame of all sequences, along with a small description for each sequence, and it is divided in two sections. All sequences are uncompressed YUV files in 4:2:0 format, where each chrominance component have half of the horizontal and vertical resolution of the luma component [106].

C.1. Sequences used in the W-SVC Transcoder

The sequences used to evaluate the W-SVC transcoder, in Chapters 4 and 5, are part of the database of the Test Media Collection, in particular of the Derf's Collection [131]. The first frame of each sequence is shown in Fig. C.1 and a brief summary of the content of each sequence is given here.

- *City*: aerial view of a city, with a skyscraper in the foreground and other buildings in the background. It contains significant camera pan movement, along with the camera moving in the horizontal axis.
- *Crew*: shot of a crew of astronauts. The camera zooms out of the subjects in the first part of the sequence, and then moves horizontally in the second half. It is also known for the particularly strong photographic flashes throughout the sequence, and there is significant movement between the subjects (walking, waving, speaking).
- *Harbour*: shot of a harbour. The camera is fixed, and there are many masts in the foreground that are still for the whole sequence, thus consisting of high frequency

Figure C.1.: Sample frames for the sequences used in the W-SVC Transcoder: (a) City; (b) Crew; (c) Harbour; (d) Soccer; (e) Foreman; and (f) Coastguard.

content, with several edges. In the background, there is movement in the water, and also some ships moving horizontally, coming in and out of the scene.

- *Soccer*: shot of a soccer practice. The camera pans and zooms out significantly through the scene, following the ball. There is also movement from the players.

- *Foreman*: shot of a foreman at a construction site. In the first part of the sequence, the camera is fixed in a close of the foreman's face, who is speaking and gesticulating. In the second part, the camera pans, showing the construction site.
- *Coastguard*: shot of boats in a river. The camera starts with a zooming to the left side, but then starts panning to the right side, following one of the boats. In the background, there is some movement in the water.

C.2. Sequences used in the HEVC Transcoder

(a) (b) (c) (d) (e)

Figure C.2.: Sample frames for the sequences used in the HEVC Transcoder: (a) BasketballDrill; (b) BQMall; (c) PartyScene; (d) RaceHorses; and (e) Vidyo1.

The sequences used to evaluate the HEVC transcoder, in Chapter 6, are part of the HEVC database [50], and are shown in Fig. C.2. A brief summary of the content of each sequence is given here.

- *BasketballDrill*: shot of a basketball practice. The camera is fixed, but there is significant movement throughout the scene, both from the players and the ball.
- *BQMall*: shot of a shopping mall. There is a slow camera pan movement in the horizontal axis, and there is also movement from a large number of people walking in different directions.
- *PartyScene*: shot of a children's party. The camera zooms in slowly, but there is movement from the children playing and also erratic movement from soap bubbles all around the scene.
- *RaceHorses*: shot of horses and horse riders preparing for a race. There is a slow camera movement, but significant movement both from the horses and riders in the scene.
- *Vidyo1*: view from a tele-conference talk. The camera is fixed, and there is very little movement through the whole scene.

Appendix D.

Quality Metrics

Throughout this thesis different quality metrics have been used in order to provide a better evaluation of the algorithms. This appendix describes three quality metrics used to evaluate a single image or video sequence, and then it describes two quality metrics used to evaluate rate distortion curves.

D.1. Quality metrics to evaluate a single image/video

The best method of evaluating the visual quality of an image or video sequence is through subjective evaluation, since usually the ultimate goal of encoding the content is to be seen by end users. However, in practice, subjective evaluation is too inconvenient, time-consuming and expensive [136]. Therefore, several objective metrics have been proposed in order to predict the perceived image quality of an image or video sequence. An objective metric is easier to compute and it has the advantage that the experiment can be easily repeatable. An objective metric is said to be reliable if its output is correlated to the output of a subjective analysis. In this thesis, two objective metrics are used: the *peak signal-to-noise ratio* (PSNR) and the *structural similarity* (SSIM), which are described in the following sections. Also, subjective experiments have been carried out, and they are also described in this section.

D.1.1. Peak Signal-to-Noise Ratio - PSNR

The assumption behind the PSNR is that the difference in perceptual quality between an image I and a reference image R (usually the original, uncompressed image) is related to the strength of the error signal between these two images. The PSNR is defined through the mean square error:

$$MSE = \frac{1}{m \cdot n} \cdot \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left(I\left(i,j\right) - R\left(i,j\right)\right)^2 \tag{D.1}$$

where m and n are the dimensions of the image, $I\left(i,j\right)$ is the pixel at position (i,j) of the image I and $R\left(i,j\right)$ is the pixel at position (i,j) of the reference image R. The PSNR is then defined as:

$$PSNR = 10 \cdot log_{10}\left(\frac{L^2}{MSE}\right) \tag{D.2}$$

where L is the maximum possible value for a pixel in the image. For instance, for the usual 8 bit images, $L = 255$. For videos, the PSNR is computed as the average PSNR among all frames of the sequence. Also, it is common to compute the PSNR only for the luma component, which is the way it is used in this thesis.

The PSNR is the most well known metric to measure the distortion between images (or videos). It is very simple to compute (given that a reference image or video is available), and it has a very clear physical meaning. However, a problem of using the PSNR as the quality metric is that two distorted images with the same PSNR may have very different types of errors, and therefore may have very different perceptual qualities. This has been extensively studied in the literature [136]. However, it has been shown [56] that, as long as the video content and the codec type are fixed across the test conditions, the correlation between the PSNR and a subjective evaluation is very high. The same study, however, also concludes that the PSNR is not a reliable quality metric when comparing different contents or codecs. In this thesis, the PSNR is only used to compare the results for the same sequence, encoded with the same coding tools (for instance, the proposed transcoder compared to the trivial transcoder, which both

use the same transform, quantization, and other tools), and therefore within the scope of validity of the PSNR as a distortion metric.

D.1.2. Structural Similarity - SSIM

Opposed to the PSNR, which computes the actual distortion in the content, the SSIM, structural similarity, is designed to measured the *perceived* change in the content, measured as the structural information [136]. The metric can be computed in various windows of an image. The measure between two windows x and y of size $m \times n$ is:

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

where μ_x and σ_x^2 are the average and the variance of the window x, μ_y and σ_y^2 are the average and the variance of the window y, σ_{xy} is the covariance between x and y, and c_1 and c_2 are just variables used to stabilize the division with weak denominators (in case either $\left(\mu_x^2 + \mu_y^2\right)$ or $\left(\sigma_x^2 + \sigma_y^2\right)$ are close to zero). It is proposed by the authors to use $c_1 = (0.01 \cdot L)^2$ and $c_2 = (0.03 \cdot L)^2$, where L is the maximum possible value for the image.

In order to compute a value for the SSIM, the default settings, as proposed by the authors, were used. The window size is fixed as 11×11, and a gaussian filter is applied to the windows x and y. The window is displaced pixel by pixel through the images, computing a value for each displacement using Eq. D.3 and producing a structural similarity map, with a SSIM value for each displacement. However, it is desired to have a single overall quality measure for the entire image, and thus the *mean structural similarity* index (MSSIM) is defined as the average of the SSIM map for all pixel displacements. The MSSIM value ranges from -1 to 1, and it is only equal to 1 if both sets of data are identical. Similarly to the PSNR, the MSSIM is only computed for the luma component and the MSSIM for a video sequence is the average of the MSSIM among all frames. It has been shown that the MSSIM is highly correlated with the output of subjective experiments [136].

D.1.3. Subjective Experiments

There are several different types of experiments to subjectively evaluate the quality of video sequences. In this thesis, a simple double stimulus subjective test is used [65]. This method is especially useful when it is not possible to produce test conditions that exhibit the full range of quality.

Let N be the number of test methods used in a given experiment and M the number of different video sequences used. In this test, the subjects are exposed to two versions of a given sequence: one of them (in random order) is always the original, uncompressed video, and the other is the video to be tested. This is known as the hidden reference method, because the user does not know which sequence is the original video (in fact, the user is not told that one of them is the original video). Each session lasts up to thirty minutes, and in every session, the user is shown, for all M sequences, $N+1$ pairs (composed of the original sequence paired with all N test methods plus a pair composed of two original sequences).

After seeing each pair of videos, the user is asked to rate both videos on a vertical scale, from 0 (worst quality) to 100 (best quality). Each of these ratings is called an *opinion score*. The test is conducted with K subjects, and the average score of all users, called *mean opinion score* (MOS) is computed and used as the quality metric.

D.2. Quality metrics to evaluate rate distortion curves

In the previous section, different quality metrics to evaluate the quality of a single image or video sequence were explained. Here, the problem is on how to measure the quality of a collection of sequences, encoded with different methods. Two ways of evaluating this quality are described in the following sections: the Average PSNR Loss and the Bjøntegaard Delta Bitrate [20].

D.2.1. Average PSNR Loss

This metric can only be used if the codec has a constant bitrate output, which is the case on the W-SVC codec used in this thesis. This way, the output of the codec can be

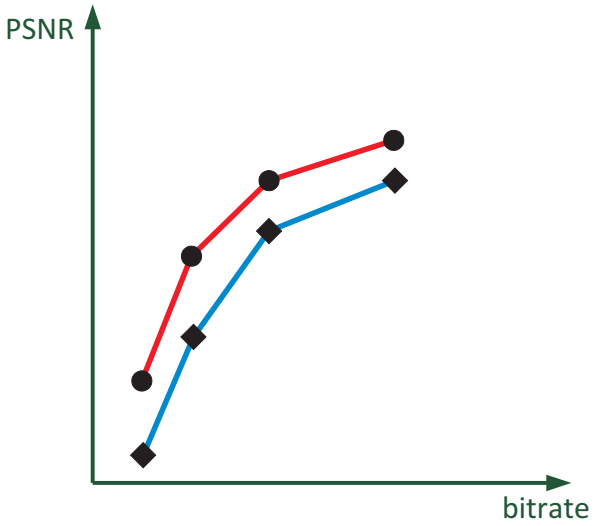


Figure D.1.: Example Average PSNR Loss. Note that the data set for both curves have the same bitrate.

evaluated by just the PSNR difference among the data points. Consider two data sets consisting of N pairs (b, p) (where b is the bitrate and p is the PSNR), with the added constraint that $b_i^1 = b_i^2$ (i.e., for the i-th pair, the bitrate is the same for the two data sets). An example is shown in Fig. D.1. The first data set is used as the anchor. The Average PSNR loss is then defined as:

$$APL = \frac{1}{N} \cdot \sum_{i=0}^{N-1} \left(p_i^2 - p_i^1\right) \tag{D.4}$$

D.2.2. Bjøntegaard Delta Bitrate

The Bjøntegaard Delta bitrate and PSNR metrics [20] have found great acceptance in the literature. The main reason is because it provides a very good and simple way to understand how close two rate distortion curves are. This metric is more robust than the Average PSNR Loss, since it does not require the bitrate among the data pairs to be the same. In this thesis, the BD bitrate is used, and it is computed as follows.

First, it is assumed that two sets of data, each consisting of four pairs (b, p) (again, where b is the bitrate and p is the PSNR), are available. Then, a logarithm is applied to the bitrate, so that each pair becomes $(log_{10}(b), p)$. A third order polynomial is fitted

through this new set of data, so that the area between the two curves can be computed. This area is set in order to avoid extrapolation of the data, as it is shown in Fig. D.2.

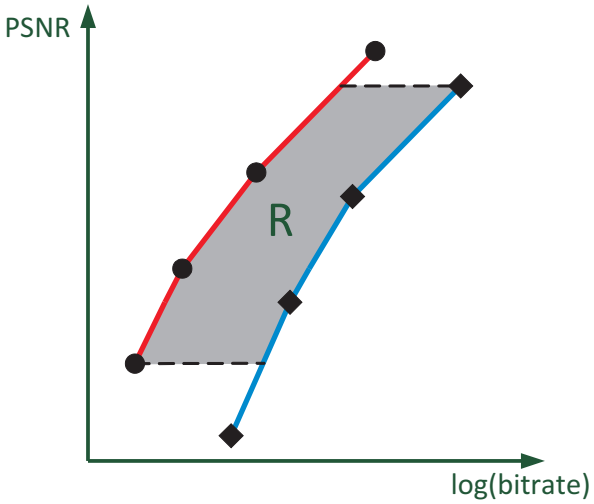


Figure D.2.: Example of BD bitrate integration limits. The BD bitrate is defined as the area R between the two curves.

The BD bitrate value is given as a percentage of average bitrate loss (if negative) or gain (if positive).

Appendix E.

Results for the H.264/AVC to W-SVC Transcoder

This appendix presents the complete results for the experiments made to evaluate the proposed transcoder from H.264/AVC to W-SVC referred in Sec. 5.4, both in terms of rate-distortion performance and complexity savings.

It presents the results for the two options for the proposed transcoder, PT (proposed transcoder) and PT-RC (proposed transcoder with the Reduced Complexity module), compared to three reference trivial transcoders: (i) using full motion estimation (RT-FS); (ii) using Hexagon Search [153] (RT-HS); and (iii) using EPZS [132] (RT-EPZS).

E.1. Average PSNR Loss Results

For all cases, the PSNR shown is the average among all frames of the luma component, and it is always computed using the original sequence as the reference, while the bit-rate always includes both the luminance and the chrominance components.

In the W-SVC codec, independently on the coding configuration and the QP used in the H.264/AVC, the following bitrates were used: $\{384, 480, 576, 720, 1152, 1536, 2304\}$ kbps, for CIF resolution; and $\{1280, 1536, 1792, 2048, 2304, 2688, 3072\}$ kbps, for 4CIF resolution. The PSNR loss presented is the average, for each of these bitrates, of the PSNR loss for each transcoder (RT-HS, RT-EPZS, PT and PT-RC) compared to RT-FS. The PSNR loss for each coding configuration used is shown in Tables E.1 to E.4.

Table E.1.: Average PSNR Loss for *IPP1* configuration, compared to RT-FS.

			16×16				32×32				64×64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	−0.69	−0.19	+0.02	−0.02	−0.40	−0.16	+0.05	−0.10	Not Available			
		Crew	−0.26	−0.07	+0.03	+0.00	−0.17	−0.04	+0.05	+0.00				
		Harbour	−0.24	−0.06	+0.02	+0.00	−0.13	−0.05	+0.03	−0.01				
		Soccer	−0.85	−0.07	+0.08	+0.04	−0.58	+0.02	+0.13	+0.02				
	QP 28	City	−0.27	−0.07	−0.01	−0.03	−0.13	−0.06	+0.01	−0.04	Not Available			
		Crew	−0.19	−0.04	+0.02	−0.01	−0.10	−0.02	+0.04	−0.01				
		Harbour	−0.14	−0.05	+0.02	+0.01	−0.10	−0.04	+0.02	+0.00				
		Soccer	−0.44	−0.04	+0.04	+0.01	−0.30	+0.01	+0.06	+0.02				
4CIF Resolution	QP 26	City	−0.49	−0.19	+0.08	+0.03	−0.20	−0.10	+0.07	+0.05	−0.14	−0.08	+0.10	+0.07
		Crew	−0.32	−0.13	+0.04	−0.13	−0.21	−0.06	+0.01	−0.02	−0.13	−0.04	+0.03	+0.00
		Harbour	−0.22	−0.07	+0.02	−0.01	−0.13	−0.04	+0.00	−0.02	−0.10	−0.03	+0.03	+0.01
		Soccer	−0.93	−0.05	+0.16	+0.04	−0.63	+0.08	+0.17	+0.14	−0.42	+0.12	+0.22	+0.13
	QP 34	City	−0.18	−0.07	+0.02	−0.01	−0.06	−0.03	+0.02	+0.01	−0.04	−0.03	+0.02	+0.01
		Crew	−0.18	−0.08	+0.03	−0.09	−0.10	−0.03	+0.01	−0.02	−0.06	−0.02	+0.01	−0.02
		Harbour	−0.13	−0.03	+0.01	−0.03	−0.09	−0.02	+0.00	+0.02	−0.06	−0.01	−0.01	+0.02
		Soccer	−0.37	−0.02	+0.08	+0.00	−0.19	+0.04	+0.07	+0.06	−0.13	+0.06	+0.08	+0.07

Table E.2.: Average PSNR Loss for *IPP5* configuration, compared to RT-FS.

			16×16				32×32				64×64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	−0.69	−0.18	+0.02	−0.01	−0.44	−0.16	+0.05	+0.01	Not Available			
		Crew	−0.31	−0.06	+0.03	+0.00	−0.16	−0.04	+0.05	+0.01				
		Harbour	−0.23	−0.06	+0.03	+0.02	−0.13	−0.04	+0.05	+0.03				
		Soccer	−0.88	−0.05	+0.09	+0.07	−0.54	+0.01	+0.13	+0.05				
	QP 28	City	−0.24	−0.07	+0.00	−0.02	−0.15	−0.06	+0.01	−0.02	Not Available			
		Crew	−0.18	−0.04	+0.03	−0.02	−0.11	−0.03	+0.03	−0.01				
		Harbour	−0.20	−0.04	+0.02	+0.01	−0.10	−0.03	+0.03	+0.01				
		Soccer	−0.44	−0.04	+0.03	+0.01	−0.28	+0.01	+0.07	+0.02				
4CIF Resolution	QP 26	City	−0.50	−0.19	+0.06	+0.02	−0.22	−0.10	+0.07	+0.05	−0.14	−0.08	+0.10	+0.08
		Crew	−0.34	−0.14	+0.05	−0.11	−0.21	−0.05	+0.02	−0.02	−0.13	−0.05	+0.03	+0.00
		Harbour	−0.22	−0.07	+0.02	−0.01	−0.13	−0.03	+0.01	+0.00	−0.10	−0.03	+0.03	+0.02
		Soccer	−0.92	−0.05	+0.15	+0.10	−0.63	+0.08	+0.18	+0.14	−0.42	+0.12	+0.22	+0.15
	QP 34	City	−0.18	−0.07	+0.02	−0.01	−0.07	−0.03	+0.02	+0.01	−0.04	−0.03	+0.02	+0.01
		Crew	−0.20	−0.09	+0.04	−0.08	−0.10	−0.03	+0.01	−0.02	−0.06	−0.02	+0.01	−0.01
		Harbour	−0.13	−0.04	+0.01	−0.04	−0.08	−0.02	+0.00	+0.01	−0.05	−0.01	+0.00	+0.01
		Soccer	−0.39	−0.02	+0.06	+0.02	−0.18	+0.05	+0.08	+0.06	−0.11	+0.06	+0.08	+0.06

Table E.3.: Average PSNR Loss for *IBBP* configuration, compared to RT-FS.

			16×16				32×32				64×64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	−0.67	−0.19	−0.01	−0.06	−0.38	−0.16	+0.03	−0.07	Not Available			
		Crew	−0.27	−0.06	+0.02	+0.00	−0.17	−0.05	+0.03	+0.00				
		Harbour	−0.23	−0.06	+0.02	+0.01	−0.14	−0.04	+0.04	+0.01				
		Soccer	−0.80	−0.07	+0.02	+0.00	−0.59	+0.01	+0.05	+0.01				
	QP 28	City	−0.26	−0.07	−0.03	−0.06	−0.13	−0.07	−0.01	−0.06	Not Available			
		Crew	−0.16	−0.04	+0.02	−0.01	−0.11	−0.03	+0.02	−0.01				
		Harbour	−0.15	−0.04	+0.01	+0.00	−0.09	−0.03	+0.02	+0.01				
		Soccer	−0.41	−0.04	+0.01	−0.01	−0.27	+0.01	+0.03	+0.00				
4CIF Resolution	QP 26	City	−0.50	−0.19	+0.07	+0.03	−0.22	−0.11	+0.06	+0.03	−0.14	−0.09	+0.09	+0.06
		Crew	−0.32	−0.14	+0.03	−0.04	−0.21	−0.06	+0.01	−0.02	−0.12	−0.04	+0.02	+0.00
		Harbour	−0.21	−0.07	+0.02	+0.01	−0.15	−0.04	+0.00	−0.01	−0.11	−0.03	+0.03	+0.02
		Soccer	−0.90	−0.05	+0.12	+0.13	−0.61	+0.09	+0.16	+0.14	−0.43	+0.12	+0.21	+0.17
	QP 34	City	−0.19	−0.08	+0.01	−0.02	−0.07	−0.04	+0.01	−0.01	−0.04	−0.03	+0.02	−0.01
		Crew	−0.18	−0.08	+0.01	−0.02	−0.09	−0.04	+0.00	−0.03	−0.06	−0.02	+0.01	−0.02
		Harbour	−0.11	−0.03	+0.01	+0.00	−0.08	−0.02	+0.00	+0.00	−0.06	−0.01	+0.02	+0.01
		Soccer	−0.39	−0.03	+0.05	+0.03	−0.18	+0.03	+0.07	+0.05	−0.11	+0.05	+0.08	+0.06

Table E.4.: Average PSNR Loss for *Hierarchical* configuration, compared to RT-FS.

			16×16				32×32				64×64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	−0.69	−0.22	+0.04	−0.02	−0.45	−0.18	+0.08	−0.01	Not Available			
		Crew	−0.26	−0.06	+0.04	+0.00	−0.17	−0.04	+0.05	+0.04				
		Harbour	−0.23	−0.06	+0.02	+0.02	−0.13	−0.05	+0.04	+0.02				
		Soccer	−0.82	−0.07	+0.01	−0.01	−0.59	+0.01	+0.02	−0.01				
	QP 28	City	−0.33	−0.11	+0.01	−0.02	−0.20	−0.09	+0.02	−0.03	Not Available			
		Crew	−0.17	−0.05	+0.04	+0.01	−0.10	−0.03	+0.06	+0.03				
		Harbour	−0.16	−0.04	+0.02	+0.03	−0.09	−0.03	+0.04	+0.02				
		Soccer	−0.43	−0.04	+0.02	+0.02	−0.27	−0.01	+0.01	+0.00				
4CIF Resolution	QP 26	City	−0.54	−0.19	+0.03	−0.08	−0.23	−0.11	+0.06	+0.02	−0.14	−0.10	+0.09	+0.06
		Crew	−0.31	−0.14	+0.06	−0.23	−0.19	−0.05	+0.01	−0.04	−0.12	−0.04	+0.02	+0.00
		Harbour	−0.21	−0.06	−0.01	−0.10	−0.13	−0.03	−0.01	−0.02	−0.10	−0.03	+0.01	+0.01
		Soccer	−0.92	−0.03	+0.20	+0.11	−0.60	+0.08	+0.09	+0.06	−0.40	+0.11	+0.15	+0.12
	QP 34	City	−0.21	−0.09	+0.03	−0.01	−0.08	−0.05	+0.02	−0.02	−0.05	−0.04	+0.03	+0.00
		Crew	−0.19	−0.09	+0.04	−0.07	−0.10	−0.04	+0.01	−0.03	−0.06	−0.03	+0.02	−0.02
		Harbour	−0.15	−0.03	+0.00	+0.04	−0.08	−0.02	+0.00	+0.00	−0.06	−0.02	+0.02	+0.00
		Soccer	−0.41	−0.02	+0.10	+0.09	−0.21	+0.03	+0.06	+0.01	−0.14	+0.04	+0.07	+0.04

E.2. Complexity Results as the Number of SAD Operations

In this section, the complexity of the transcoder is measured in terms of the average number of SAD operations. In the W-SVC codec, the cost of each motion vector is computed as $J = D + \lambda \cdot R$. The complexity of estimating the rate R is fairly low, since R is calculated for the whole block by simple prediction of the neighbouring motion information. However, calculating the distortion D, which is measured as the SAD, involves many more operations, and it is the most complex operation in the motion

estimation module, even if fast motion estimation methods are used. The SAD of a block B can be expressed as $SAD_B = \sum_{i=0}^{M-1} |p_i - \hat{p_i}|$, where M is the number of considered pixels in the block, p_i is the pixel with the index i in the block B and $\hat{p_i}$ is the pixel with the index i in the reference block. Following the above equation, a SAD operation is defined as calculating a difference between two pixels, calculating the absolute value of that difference and adding the absolute difference to the sum of previously calculated absolute differences. Thus, calculating the SAD for the block B consists of M SAD operations. Usually, in the motion estimation implementation, once the SAD for the first motion vector tested is computed and it already has a value for the current lowest cost (J_{best}), the SAD for the other motion vectors tested is only computed until this limit is reached (as this motion vector would never be chosen, as the cost for this motion vector, $J_{current}$, would already be higher than J_{best}). Therefore, the number of operations for non-optimal motion vectors is usually lower than M. This implementation optimisation is used in these tests, for all three reference transcoders and the two proposed transcoders.

Using the number of SAD operations as an indication of complexity has the advantage of being an objective measure, independent of the software optimizations and the hardware used for experiments. However, it does not account for the complexity of the MV approximation techniques, or computing the motion vector similarity in the RC module, for instance, and it is therefore a lower bound of the transcoder complexity. This is useful to better understand the maximum speed-up that can be achieved for each transcoding option, and also to compare the transcoder to the trivial transcoder using full motion estimation, RT-FS. The results are shown in Tables E.5 through E.8.

E.3. Complexity Results

To measure the running time, a PC with an Intel Core i5-2400 running at 3.10 GHz with 8 Gb of RAM and Windows 7 64-bit was used. Again, since the transcoder is mainly based on motion estimation and mode decision module, only the time spent on these modules is considered here. Each sequence was encoded 3 times, and the average time spent on the motion estimation and mode decision for all frames was measured. These results are shown as the Speed-Up relative to the RT-HS, which is used as a benchmark for speed. In this case, all modules of the transcoder are accounted for, including the MV Approximation methods, computing MV Similarity, and deciding the partition using the H.264/AVC motion information. Therefore, this is the complexity of the implemented transcoder. The results are shown in Tables E.9 through E.12.

Table E.5.: Number of SAD operations for *IPP1* configuration, comparing to RT-HS.

			16 × 16				32 × 32				64 × 64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	0.71	0.92	0.42	0.21	0.65	0.87	0.59	0.29	Not Available			
		Crew	0.88	1.20	0.78	0.59	0.96	1.28	1.11	0.89				
		Harbour	0.49	0.66	0.41	0.28	0.47	0.64	0.58	0.36				
		Soccer	0.94	1.21	0.68	0.41	1.04	1.34	1.02	0.62				
	QP 28	City	0.73	0.91	0.37	0.18	0.69	0.88	0.53	0.26	Not Available			
		Crew	0.91	1.20	0.71	0.51	1.01	1.31	1.02	0.79				
		Harbour	0.49	0.66	0.37	0.26	0.49	0.66	0.54	0.34				
		Soccer	0.98	1.18	0.60	0.35	1.10	1.35	0.93	0.56				
4CIF Resolution	QP 26	City	0.62	0.78	0.33	0.16	0.74	0.91	0.66	0.30	0.70	0.89	0.86	0.43
		Crew	0.59	0.79	0.42	0.29	0.73	0.91	0.77	0.54	0.78	0.97	1.01	0.77
		Harbour	0.51	0.61	0.36	0.25	0.49	0.60	0.57	0.36	0.45	0.56	0.72	0.48
		Soccer	0.63	0.76	0.38	0.20	0.77	0.89	0.73	0.38	0.81	0.96	0.97	0.55
	QP 34	City	0.64	0.75	0.28	0.10	0.78	0.90	0.56	0.21	0.76	0.91	0.74	0.32
		Crew	0.60	0.76	0.38	0.23	0.76	0.91	0.71	0.45	0.82	0.99	0.93	0.65
		Harbour	0.52	0.60	0.28	0.17	0.52	0.61	0.46	0.27	0.50	0.60	0.58	0.36
		Soccer	0.63	0.70	0.34	0.15	0.81	0.87	0.67	0.32	0.88	0.97	0.90	0.48

Table E.6.: Number of SAD operations for *IPP5* configuration, comparing to RT-HS.

			16 × 16				32 × 32				64 × 64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	0.71	0.92	0.43	0.25	0.65	0.87	0.59	0.38	Not Available			
		Crew	0.88	1.20	0.78	0.59	0.96	1.29	1.10	0.89				
		Harbour	0.49	0.66	0.42	0.31	0.47	0.64	0.58	0.44				
		Soccer	0.94	1.21	0.68	0.41	1.04	1.34	1.01	0.66				
	QP 28	City	0.73	0.91	0.37	0.19	0.69	0.89	0.52	0.29	Not Available			
		Crew	0.91	1.20	0.69	0.50	1.01	1.32	1.01	0.78				
		Harbour	0.49	0.66	0.38	0.28	0.49	0.66	0.53	0.40				
		Soccer	0.98	1.18	0.59	0.35	1.11	1.35	0.92	0.57				
4CIF Resolution	QP 26	City	0.63	0.79	0.34	0.18	0.76	0.93	0.67	0.39	0.71	0.90	0.87	0.55
		Crew	0.59	0.79	0.42	0.28	0.73	0.91	0.76	0.54	0.78	0.97	1.00	0.76
		Harbour	0.51	0.61	0.36	0.27	0.49	0.60	0.58	0.43	0.45	0.56	0.72	0.57
		Soccer	0.63	0.77	0.38	0.21	0.78	0.90	0.73	0.41	0.82	0.96	0.96	0.59
	QP 34	City	0.65	0.75	0.28	0.10	0.79	0.91	0.56	0.22	0.77	0.92	0.74	0.34
		Crew	0.61	0.77	0.38	0.22	0.76	0.91	0.71	0.45	0.83	0.99	0.92	0.64
		Harbour	0.52	0.60	0.28	0.17	0.52	0.62	0.45	0.29	0.50	0.60	0.58	0.40
		Soccer	0.63	0.70	0.34	0.15	0.81	0.88	0.67	0.32	0.88	0.98	0.90	0.49

Table E.7.: Number of SAD operations for *IBBP* configuration, comparing to RT-HS.

			16×16				32×32				64×64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	0.71	0.92	0.40	0.22	0.65	0.86	0.56	0.33	Not Available			
		Crew	0.88	1.20	0.73	0.59	0.96	1.28	1.05	0.88				
		Harbour	0.49	0.66	0.37	0.31	0.47	0.64	0.54	0.43				
		Soccer	0.95	1.21	0.66	0.43	1.05	1.34	1.01	0.68				
	QP 28	City	0.75	0.92	0.36	0.20	0.70	0.89	0.53	0.29	Not Available			
		Crew	0.92	1.20	0.66	0.51	1.02	1.32	0.98	0.78				
		Harbour	0.49	0.66	0.33	0.28	0.49	0.66	0.49	0.39				
		Soccer	0.98	1.19	0.60	0.38	1.12	1.37	0.95	0.61				
4CIF Resolution	QP 26	City	0.63	0.79	0.31	0.17	0.77	0.93	0.64	0.32	0.72	0.91	0.84	0.46
		Crew	0.60	0.79	0.40	0.28	0.74	0.91	0.74	0.53	0.78	0.97	0.97	0.74
		Harbour	0.52	0.61	0.32	0.25	0.50	0.59	0.53	0.40	0.46	0.56	0.66	0.53
		Soccer	0.63	0.77	0.37	0.21	0.79	0.90	0.72	0.40	0.83	0.97	0.95	0.57
	QP 34	City	0.66	0.77	0.30	0.13	0.83	0.95	0.62	0.25	0.80	0.95	0.83	0.37
		Crew	0.61	0.77	0.37	0.24	0.77	0.92	0.70	0.45	0.83	1.00	0.92	0.65
		Harbour	0.53	0.59	0.27	0.19	0.51	0.60	0.44	0.30	0.48	0.58	0.55	0.39
		Soccer	0.63	0.72	0.34	0.17	0.82	0.90	0.69	0.35	0.89	1.00	0.94	0.51

Table E.8.: Number of SAD operations for *Hierarchical* configuration, comparing to RT-HS.

			16×16				32×32				64×64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	0.70	0.90	0.37	0.21	0.65	0.86	0.53	0.31	Not Available			
		Crew	0.88	1.19	0.66	0.51	0.97	1.28	0.99	0.82				
		Harbour	0.49	0.66	0.33	0.27	0.47	0.64	0.50	0.38				
		Soccer	0.94	1.19	0.60	0.38	1.04	1.33	0.95	0.65				
	QP 28	City	0.71	0.88	0.32	0.13	0.68	0.86	0.47	0.23	Not Available			
		Crew	0.92	1.19	0.60	0.37	1.01	1.31	0.91	0.66				
		Harbour	0.49	0.65	0.29	0.22	0.48	0.65	0.44	0.33				
		Soccer	0.98	1.17	0.54	0.27	1.10	1.34	0.87	0.51				
4CIF Resolution	QP 26	City	0.63	0.78	0.30	0.12	0.75	0.91	0.59	0.28	0.70	0.89	0.79	0.43
		Crew	0.59	0.78	0.39	0.22	0.73	0.90	0.72	0.47	0.78	0.96	0.98	0.72
		Harbour	0.51	0.60	0.28	0.21	0.49	0.59	0.48	0.34	0.45	0.56	0.65	0.49
		Soccer	0.63	0.76	0.35	0.16	0.78	0.89	0.68	0.35	0.81	0.96	0.94	0.56
	QP 34	City	0.65	0.76	0.28	0.06	0.79	0.90	0.56	0.17	0.77	0.91	0.76	0.29
		Crew	0.61	0.76	0.36	0.14	0.76	0.90	0.68	0.34	0.82	0.98	0.91	0.55
		Harbour	0.52	0.57	0.25	0.11	0.49	0.56	0.40	0.21	0.46	0.54	0.52	0.33
		Soccer	0.63	0.71	0.33	0.09	0.82	0.88	0.67	0.25	0.88	0.98	0.92	0.43

Table E.9.: Implemented transcoder speed-up for *IPP1* configuration, comparing to RT-HS.

			16 × 16				32 × 32				64 × 64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	1.00	0.58	1.26	2.30	1.00	0.63	0.87	1.71	Not Available			
		Crew	1.00	0.57	0.93	1.21	1.00	0.63	0.74	0.93				
		Harbour	1.00	0.56	0.90	1.37	1.00	0.63	0.62	1.02				
		Soccer	1.00	0.59	1.13	1.83	1.00	0.64	0.86	1.41				
	QP 28	City	1.00	0.60	1.56	2.83	1.00	0.65	1.09	2.10	Not Available			
		Crew	1.00	0.59	1.11	1.48	1.00	0.65	0.87	1.11				
		Harbour	1.00	0.56	1.02	1.50	1.00	0.63	0.72	1.16				
		Soccer	1.00	0.63	1.41	2.21	1.00	0.67	1.06	1.69				
4CIF Resolution	QP 26	City	1.00	0.61	1.57	2.83	1.00	0.73	0.94	1.88	1.00	0.76	0.70	1.33
		Crew	1.00	0.60	1.22	1.68	1.00	0.72	0.84	1.14	1.00	0.76	0.69	0.89
		Harbour	1.00	0.62	1.19	1.70	1.00	0.73	0.71	1.12	1.00	0.77	0.53	0.80
		Soccer	1.00	0.65	1.44	2.46	1.00	0.77	0.93	1.68	1.00	0.79	0.75	1.28
	QP 34	City	1.00	0.67	2.02	4.44	1.00	0.77	1.22	2.76	1.00	0.79	0.92	1.97
		Crew	1.00	0.64	1.40	2.12	1.00	0.75	0.96	1.43	1.00	0.78	0.81	1.13
		Harbour	1.00	0.65	1.62	2.46	1.00	0.73	0.98	1.62	1.00	0.77	0.76	1.19
		Soccer	1.00	0.71	1.69	3.23	1.00	0.82	1.10	2.12	1.00	0.84	0.90	1.60

Table E.10.: Implemented transcoder speed-up for *IPP5* configuration, comparing to RT-HS.

			16 × 16				32 × 32				64 × 64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	1.00	0.58	1.25	1.96	1.00	0.64	0.89	1.33	Not Available			
		Crew	1.00	0.57	0.94	1.22	1.00	0.63	0.74	0.91				
		Harbour	1.00	0.56	0.81	1.07	1.00	0.63	0.57	0.77				
		Soccer	1.00	0.59	1.13	1.77	1.00	0.63	0.87	1.30				
	QP 28	City	1.00	0.60	1.56	2.70	1.00	0.65	1.10	1.87	Not Available			
		Crew	1.00	0.59	1.14	1.52	1.00	0.65	0.89	1.13				
		Harbour	1.00	0.56	0.99	1.33	1.00	0.63	0.71	0.97				
		Soccer	1.00	0.63	1.42	2.21	1.00	0.67	1.08	1.65				
4CIF Resolution	QP 26	City	1.00	0.62	1.57	2.59	1.00	0.73	0.95	1.53	1.00	0.76	0.71	1.08
		Crew	1.00	0.60	1.25	1.73	1.00	0.72	0.85	1.16	1.00	0.76	0.71	0.91
		Harbour	1.00	0.62	1.14	1.50	1.00	0.73	0.70	0.93	1.00	0.77	0.53	0.67
		Soccer	1.00	0.65	1.45	2.41	1.00	0.77	0.94	1.58	1.00	0.79	0.77	1.21
	QP 34	City	1.00	0.67	2.03	4.47	1.00	0.78	1.23	2.73	1.00	0.79	0.93	1.89
		Crew	1.00	0.64	1.44	2.21	1.00	0.75	0.98	1.46	1.00	0.78	0.82	1.15
		Harbour	1.00	0.66	1.63	2.44	1.00	0.73	1.00	1.53	1.00	0.77	0.78	1.11
		Soccer	1.00	0.72	1.70	3.25	1.00	0.82	1.11	2.12	1.00	0.84	0.91	1.61

Table E.11.: Implemented transcoder speed-up for *IBBP* configuration, comparing to RT-HS.

			16 × 16				32 × 32				64 × 64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	1.00	0.58	1.41	2.32	1.00	0.64	0.95	1.55	Not Available			
		Crew	1.00	0.56	1.01	1.23	1.00	0.63	0.79	0.94				
		Harbour	1.00	0.56	1.05	1.29	1.00	0.63	0.68	0.85				
		Soccer	1.00	0.59	1.18	1.71	1.00	0.65	0.89	1.30				
	QP 28	City	1.00	0.60	1.69	2.80	1.00	0.65	1.13	1.94	Not Available			
		Crew	1.00	0.59	1.18	1.49	1.00	0.65	0.92	1.14				
		Harbour	1.00	0.56	1.20	1.43	1.00	0.62	0.79	0.98				
		Soccer	1.00	0.62	1.41	2.05	1.00	0.67	1.05	1.55				
4CIF Resolution	QP 26	City	1.00	0.62	1.75	2.89	1.00	0.74	1.03	1.87	1.00	0.76	0.75	1.30
		Crew	1.00	0.61	1.33	1.75	1.00	0.72	0.89	1.19	1.00	0.76	0.74	0.95
		Harbour	1.00	0.63	1.39	1.70	1.00	0.73	0.78	1.00	1.00	0.77	0.55	0.66
		Soccer	1.00	0.65	1.53	2.41	1.00	0.78	0.98	1.64	1.00	0.80	0.79	1.27
	QP 34	City	1.00	0.67	2.02	3.83	1.00	0.78	1.20	2.65	1.00	0.80	0.88	1.84
		Crew	1.00	0.64	1.51	2.16	1.00	0.75	1.00	1.48	1.00	0.79	0.84	1.17
		Harbour	1.00	0.67	1.75	2.33	1.00	0.74	1.02	1.45	1.00	0.77	0.73	0.99
		Soccer	1.00	0.70	1.72	2.91	1.00	0.81	1.09	2.01	1.00	0.83	0.88	1.55

Table E.12.: Implemented transcoder speed-up for *Hierarchical* configuration, comparing to RT-HS.

			16 × 16				32 × 32				64 × 64			
		Sequence	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC	RT-HS	RT-EPZS	PT	PT-RC
CIF Resolution	QP 20	City	1.00	0.58	1.55	2.55	1.00	0.63	1.07	1.79	Not Available			
		Crew	1.00	0.57	1.18	1.46	1.00	0.64	0.89	1.06				
		Harbour	1.00	0.55	1.28	1.55	1.00	0.63	0.82	1.10				
		Soccer	1.00	0.60	1.37	2.09	1.00	0.65	1.01	1.47				
	QP 28	City	1.00	0.60	1.95	3.93	1.00	0.65	1.31	2.50	Not Available			
		Crew	1.00	0.59	1.40	2.10	1.00	0.65	1.04	1.41				
		Harbour	1.00	0.56	1.43	1.86	1.00	0.62	0.97	1.30				
		Soccer	1.00	0.63	1.65	2.90	1.00	0.67	1.19	1.95				
4CIF Resolution	QP 26	City	1.00	0.63	1.95	3.90	1.00	0.74	1.15	2.27	1.00	0.76	0.83	1.47
		Crew	1.00	0.61	1.39	2.17	1.00	0.73	0.93	1.37	1.00	0.76	0.74	1.00
		Harbour	1.00	0.63	1.64	2.12	1.00	0.73	0.92	1.27	1.00	0.77	0.65	0.85
		Soccer	1.00	0.65	1.67	3.20	1.00	0.78	1.07	1.93	1.00	0.79	0.82	1.34
	QP 34	City	1.00	0.67	2.21	7.20	1.00	0.78	1.34	3.89	1.00	0.80	0.97	2.38
		Crew	1.00	0.65	1.56	3.35	1.00	0.75	1.05	1.95	1.00	0.79	0.85	1.37
		Harbour	1.00	0.68	1.91	3.76	1.00	0.75	1.15	2.02	1.00	0.79	0.84	1.31
		Soccer	1.00	0.70	1.84	5.05	1.00	0.82	1.18	2.83	1.00	0.84	0.92	1.89

Appendix F.

Linear Discriminant Functions

This appendix presents a brief overview on Linear Discriminant Functions [13, 111], which are a simple and robust polynomial classifier, with the advantage of having a very fast and non-iterative training. These classifiers have been successfully used in many applications of pattern recognition, specifically speech and speaker recognition [25] and biomedical signal separation [12].

F.1. Training Procedure

Consider a K-class pattern recognition problem whose feature vectors have dimension M. Denote by

$$\mathbf{X}_i = [\mathbf{x}_{i,1} \;\; \mathbf{x}_{i,2} \;\; ... \;\; \mathbf{x}_{i,N_i}]^T \tag{F.1}$$

the sequence of feature vectors corresponding to a particular class i, where $\mathbf{X}_i$ is a $N_i \times M$ matrix, and N_i is the number of feature vectors that belong to class i. Concatenating the $\mathbf{X}_i$ matrices for all K classes results in:

$$\mathbf{X} = [\mathbf{X}_1 \;\; \mathbf{X}_2 \;\; ... \;\; \mathbf{X}_K]^T \tag{F.2}$$

Let $\mathbf{y}_i$ be the ideal output vector for class i, which is a column vector comprised of zeros and ones, such as: $\mathbf{y}_i = \left[\mathbf{0}_{N_1}\ ,\ \mathbf{0}_{N_2}\ ,\ \ldots\ ,\ \mathbf{0}_{N_{i-1}}\ ,\ \mathbf{1}_{N_i}\ ,\ \mathbf{0}_{N_{i+1}}\ ,\ \ldots\ ,\ \mathbf{0}_{N_K}\right]^T$. Naturally, $\mathbf{y}_i$ is one for those feature vectors that belong to class i.

The training procedure consists of computing the optimum weight vector $\mathbf{w}_i^{opt}$ that minimises the distance between $\mathbf{y}_i$ and a linear combination of the training feature vectors $\mathbf{X}\mathbf{w}_i$ such that

$$\mathbf{w}_i^{opt} = \underset{\mathbf{w}_i}{\arg\ min}\ \left\|\mathbf{X}\mathbf{w}_i - \mathbf{y}_i\right\|_p \tag{F.3}$$

Eq. F.3 indicates that the optimal weight vector $\mathbf{w}_i^{opt}$ could be obtained by minimising the L^p-norm of the error vector $\mathbf{e}_i = \mathbf{X}\mathbf{w}_i - \mathbf{y}_i$. While it has been shown that using the L^1-norm can lead to a more robust classifier [13], the L^2-norm is used because it has a non-iterative solution, making it more suitable to be used inside the loop of the transcoder. Minimising the function given in Eq. F.3 using the L^2-norm leads to:

$$\mathbf{w}_i^{opt} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}_i \tag{F.4}$$

which can be simply solved.

F.2. Using the Classifier

Once the optimal weights are computed, using the classifier is a simple operation. Let X_C be the feature vector that needs to be classified as one of the K classes. This can be done by evaluating the output of this feature vector against all K models, computing a set of *scores* s_i, such as:

$$\mathbf{s}_i^{opt} = \mathbf{X}_C\mathbf{w}_i^{opt} \tag{F.5}$$

Then, the class of the sequence X_C is determined by choosing the class with the highest score:

$$c = \arg\limits_{i} max \ \ (\mathbf{s}_i) \tag{F.6}$$

Note that, since the classification is performed solving Eq F.5, which is a linear combination of the feature vector $\mathbf{X}_C$ and the optimal weights $\mathbf{w}_i^{opt}$, and then the decision is made by choosing the highest output, this classifier is often called *linear discriminant functions*.