Temporal Pixel Trajectories for Frame Denoising in a Hybrid Video Codec

vorgelegt von
Marko Esche, M.Sc.
geb. in Berlin

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
-Dr.-Ing.-

genehmigte Dissertation

Promotionsausschuss

Vorsitzender:	Prof. Dr.-Ing. T. Wiegand
1. Gutachter:	Prof. Dr.-Ing. T. Sikora
2. Gutachter:	Prof. Dr.-Ing. A. Kaup
3. Gutachter:	Prof. Dr.-Ing. P. Eisert

Tag der wissenschaftlichen Aussprache: 23.5.2014

Berlin 2014

Acknowledgments

First of all, I would like to thank Prof. Dr.-Ing. Thomas Sikora for the opportunity to write this thesis under his guidance. Numerous ideas and hints of his not only formed the basis of this thesis but also inspired the formulation of the underlying theory. I would also like to thank Michael Tok, Alexander Glantz, and Andreas Krutz for their cooperation on most of the scientific papers published in the context of this thesis as well as for their helpful professional feedback. Special thanks also goes to Lieven Lange whose bachelor and master thesis I had the pleasure to supervise. Most of the investigations mentioned in Chapter 6 were conducted by him.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die Dissertation selbstständig verfasst habe. Alle benutzten Hilfsmittel und Quellen sind aufgeführt.

Eine Anmeldung der Promotionsabsicht habe ich an keiner anderen Fakultät oder Hochschule beantragt.

Berlin, den .....

Abstract

In compressed video sequences artifacts frequently occur at low bit rates. One possible way to reduce these artifacts and to lower the required bit rate are in-loop filters. Among them are filters that work in the spatial domain only and those that utilize the temporal domain as well. In order to effectively perform temporal filtering, accurate motion information per pixel is required. In this thesis, one temporal filter, the Temporal Trajectory Filter (TTF), is investigated. Methods are described to reconstruct pixel motion paths from block-based motion vectors. In addition, a theoretical foundation for the filter is derived and predictions concerning the theoretically achievable gain are made. The thesis also covers several additions to the filter such as quadtree-based parameter signaling, side-information compression and dense motion vector field interpolation to improve the motion accuracy. Even for the latest version of the new video compression standard H.265/MPEG.H Part 2 the filter still produces an average additional bit rate reduction of 0.4% with much higher values for prior versions and for H.264/AVC. Finally, possible implementations of the TTF as a post-filter are presented and evaluated. These include a highly adaptive neural network approach and a true reference-free post-filter.

Zusammenfassung

In komprimierten Videosequenzen treten generell Artefakte bei niedrigen Bitraten auf. Sogenannte *in-loop*-Filter stellen eine Möglichkeit dar, diese Artefakte zu reduzieren. Zu *in-loop*-Filtern gehören sowohl solche, die ausschließlich räumlich arbeiten, als auch solche die zusätzlich noch die zeitliche Dimension beinhalten. Damit die temporale Filterung effektiv funktionieren kann, werden exakte Bewegungsinformationen für jedes Pixel benötigt. In dieser Arbeit wird ein zeitliches Filter, das Temporal Trajectory Filter (TTF), näher beleuchtet. Unter anderem werden Methoden vorgestellt, um die Bewegung eines einzelnen Pixels aus blockbasierten Bewegungsvektoren zu rekonstruieren. Zusätzlich wird ein theoretisches Fundament für das Filter aufgebaut und es werden Vorhersagen bezüglich der Filtereffektivität gemacht. In der Arbeit werden weiterhin Erweiterungen des Filters wie die quadtreebasierte Parametersignalisierung, Seiteninformationskompression und dichte Bewegungsvektorfeldinterpolation zur Verbesserung der Bewegungsrepräsentation vorgestellt. Selbst für die letzte Version des neuen Videokodierungsstandards H.265/MPEG.H Part 2 konnte das Filter noch mittlere Bitratenreduktionen von 0.4% erzielen. Für frühere Versionen des Testmodells und für H.264/AVC wurden sogar noch deutlich bessere Ergebnisse erzielt. Abschließend werden mögliche Implementierungen des TTFs als Postfilter vorgestellt und untersucht. Zu diesen gehört ein hochadaptives neuronales Netzwerk und ein echtes referenzfreies Postfilter.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

People often say that motivation doesn't last. Well, neither does bathing - that's why we recommend it daily. - Zig Ziglar

1.1 Motivation

Transmitting high-definition (HD) video files over today's communication networks still poses a problem, at least when resonable quality and a low delay at the receiver are required. With the introduction of super high-definition monitors ranging from resolutions of 2560×1600 pixels to 8000×4000 pixels, storage and transmission of suitable video files becomes an even more ambitious task. The raw bit rate for an HD video of 1920×1080 pixels alone at a framerate of 60 Hz in YUV420 format is $1920 \cdot 1080 \cdot 60\frac{1}{\mathrm{s}} \cdot 12\,\frac{\mathrm{bit}}{\mathrm{pixel}} = 1.49\frac{\mathrm{Gbit}}{\mathrm{s}}$. For an 8k × 4k-video the respective bit rate would be $23.04\frac{\mathrm{Gbit}}{\mathrm{s}}$. Even with the compression ratios provided by today's state-of-the-art video codecs like ITU-T H.264/ISO/IEC 14496-10 (*H.264/AVC*) the required bandwidth for transmitting such videos is still extremely high. Storing them on traditional media such as DVD or BluRay also poses a problem. In an attempt to essentially half the required bit rate for such videos, ITU and MPEG issued a joined Call for Proposals (CfP) in 2009 [45]. The now completed standardization activity *High Efficiency Video Coding* (HEVC) supervised by the *Joint Collaborative Team on Video Coding* (JCT-VC) saw the introduction of new tools that enable the new standard ISO/IEC 23008-2 MPEG-H Part 2 / ITU-T H.265 to provide compression ratios in the range of 50% compared to *H.264/AVC*. Nevertheless, as will be shown in this thesis, the achievements of the JCT-VC do not pose an upper limit. In fact, significant additional bit rate savings can be produced by further exploiting temporal dependencies within the video content. In an ideal case, where the individual motion of every image point is known throughout the duration of the video at both

encoder and decoder and where the illumination of the scene does not change, every new frame could be reconstructed by reusing the image content from the previously decoded frame. Additonal information would only be required when new content is introduced by newly appearing objects, rotations of objects that are already visible etc. In practice, perfect knowledge of individual pixel motion may, however, only occur under very special circumstances such as fixed cameras with stationary scenes. In every other case the overhead for transmitting individual pixel motion, so-called trajectories, becomes immense.

The traditional hybrid video codec, therefore, associates motion not with individual pixels but with blocks, that cluster neighboring pixels together. As soon as the video content can no longer be respresented by square blocks or if non-translational motion occurs, this motion model will to a certain degree fail. As a result, the implicit trajectories are no longer accurate and are only rough approximations of the true motion. Other motion representations such as global or affine motion models can overcome this deficiency. They are, however, only applicable to very large foreground objects or to the entire background. No matter where the motion information actually originates from and which model is used, hybrid video coding has two distinct features where such information may be used: The first is the generation of a prediction signal through motion compensation. This is a well established concept dating back to the very beginnings of video compression [24]. The second feature concerns the denoising of reconstructed frames either as an in-loop or a post-processing step. So-called temporal denoising filters have already been under investigation for several decades [41]. Mainly due to their computational complexity, they have, however, never yet been included in a video compression standard. A third possible application would be the use of motion information in applying a temporal filter as a preprocessing step. Here, denoising could improve the coding efficiency of a codec by increasing the temporal stability of the input sequence. However, since the usual evaluation criterion for compressed video sequences is the *mean squared error* (MSE) between input and reconstructed sequence, intentionally modifying the input sequence may in fact provide worse results in a rate-distortion sense. This is especially the case since a temporal filter can generally not distinguish between unwanted video artifacts such as camera noise and wanted artifacts such as heat haze. The removal of the latter will, of course, decrease the objective quality of the decoded sequence.

This thesis attempts to highlight certain properties of individual pixel trajectories that justify their use within a video encoder/decoder pair. There are a number of possible applications for pixelwise motion information within a hybrid video codec. However, covering all of them is beyond the scope of this thesis. Instead, a special focus will be placed on in-loop and post-filters for denoising since they share several

important aspects: The information available to such filters concering past encoded frames is identical and both can theoretically be modeled in the same manner. Moreover, the individual quality improvement for each treated frame will in both cases also be identical. Both temporal filtering concepts will be analysed in this thesis based on a theoretical description of their behavior. In addition, their optimality for some scenarios will also be demonstrated. Before highlighting the main contributions of this thesis the following literature survey will give an overview of other filtering concepts, their individual properties as well as their usage in today's video codecs. The chapter will be completed by a detailed description of the structure of the remaining thesis.

1.2 Literature Survey

The following section shall highlight some of the previous work done in the context of denoising and deblocking filters in order to put the contributions of this thesis into perspective.

1.2.1 General Denoising/Filtering Concepts

In-loop filters in video codecs serve the dual purpose of subjectively improving the quality of reconstructed frames while also reducing the required bit rate for transmitting a frame at a predefined quality [33]. The latter effect is due to the improved quality of frames used for the reconstruction of future images. Therefore, it cannot be reproduced in a post-processing scenario. An in-loop filter was first standardized as an optional codec component in ITU-T H.261 [25]. Details concering the utilized low-pass filter in H.261 and considerations concerning an optimal denoising-filter may be found in [42]. Other filters were later investigated during the standardization of ITU-T H.263 [27]. Best known, however, is the deblocking filter that was introduced with the standardization of H.264/AVC [28].

Deblocking Filter, List et al. In [33] List et al. give additional reasons for using filters within the local decoding loop at the encoder: They allow the encoder to guarantee a certain quality level at the decoder since disabling the filter at the decoder is no longer possible without losing synchronization with the encoder. Furthermore, no extra frame buffer is required at the decoder as no filtering is performed after a picture has been reconstructed. The H.264/AVC deblocking filter performs one-dimensional smoothing operations along block edges between neighboring 4×4-blocks using a *finite-impulse-response* (FIR) filter. The strength of the filter is

Block modes and conditions	$Bs = 1$
One of the blocks is Intra *and* the edge is a macroblock edge	4
One of the blocks is Intra	3
One of the blocks has coded residuals	2
Difference of block motion ≥ 1 luma sample distance	1
Motion compensation from different reference frames	1
Else	0

Table 1.1: Filter Strength parameter as a function of the coding mode. Source: [33].

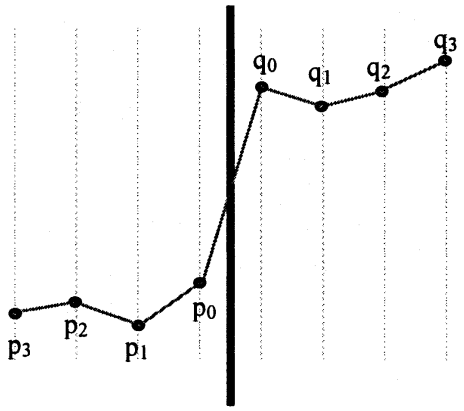


Figure 1.1: One-dimensional visualization of a block edge, where the y-axis signifies the luminance value. Source: [33].

chosen according to the boundary strength parameter Bs. Although the filter only works in the spatial domain by changing the content of only one frame at a time, a temporal component is added, too. For the case of $Bs = 1$, which corresponds to a very weak smoothing operation, the reference frames for neighboring blocks are also taken into account. Here the assumption is the following: The further apart the reference frames are, the more likely is the introduction of unwanted coding artifacts in the reconstructed frame, see Table 1.1 for reference. In addition to the adaptability introduced by the different boundary strength values, the filter depends on the actual luminance distribution at a block edge in order to distinguish between true edges in the image and block artifacts. For this evaluation, eight pixels on a line orthogonal to the block edge (p_0 to p_3 and q_0 to q_3) are examined. The naming convention as described in [33] can be seen in Figure 1.1. The subsequent filtering is only applied to a block if the following conditions hold.

$$\begin{aligned} |p_0 - q_0| &< \alpha(\text{Index}_A) \\ |p_1 - p_0| &< \beta(\text{Index}_B) \\ |q_1 - q_0| &< \beta(\text{Index}_B) \end{aligned} \tag{1.1}$$

Where Index_A and Index_B are directly calculated from the respective quantization parameter (QP) and the values for α and β have been derived empirically. The QP-dependency of α and β is motivated according to List et al. by the following statement: "Since the threshold values increase with QP, boundaries that contain higher content activity are filtered when QP is larger, since the coding error (size of artifacts) increases with QP. [sic]" This property will also be used in the design of the temporal filters discussed later in this thesis. The main advantage of the H.264/AVC deblocking filter is the omission of additional side-information. Since the filter is not optimized for any given picture at the encoder, no filter parameters need to be transmitted. All information required by the decoder is readily available from the bit stream.

Block-based Adaptive Loop Filter A first attempt to adapt the filter directly to the image content was described by Wittmann and Wedi in [57]. Although the filter detailed therein is described as a post-processing step, it may also be used in-loop. Based on the well-known Wiener-Hopf equation the optimal denoising of a signal s' with a noisy free original s can be achieved by applying an FIR-filter with the coefficients w to s':

$$w = R_{s's'}^{-1} \cdot R_{s's}, \tag{1.2}$$

where $R_{s's'}$ is the auto-correlation function of the noisy signal and $R_{s's}$ is the cross-correlation function between the noisy signal and the noise-free original. As described by Wittmann and Wedi, the required side-information for applying the filter at the decoder can either contain the coefficients w themselves or the correlation information. In the latter case the decoder has the choice to design a suitable filter itself. According to [57] their post-filter provided a mean bit rate reduction of 8.5% for the *joint video team*'s (JVT) test set as defined in [1]. Further adaptibility of the Wiener Filter method was introduced by a contribution to ITU's *Video Coding Experts Group* (VCEG) in [7] as a possible extension to H.264/AVC. The Block-based Adaptive Loop Filter (BALF) defines a certain block-size between 8×8 and 128×128 for each frame. On this block-level the Wiener-based Adaptive Loop Filter (ALF) is either switched on or off. In addition, different filter coefficients were, for the first time, applied to the chrominance samples of a decoded frame. When used as an in-loop filter, the BALF provided an average bit rate reduction of 7.20% for the VCEG common test conditions [1].

Quadtree-based Adaptive Loop Filter A fully flexible version of the ALF, that is no longer restricted to fixed block-sizes, was introduced in [6] by Chujoh et al. The Quadtree-based Adaptive Loop Filter (QALF) again switches the ALF on and off on the block level. However, no flag for a fixed block size is required

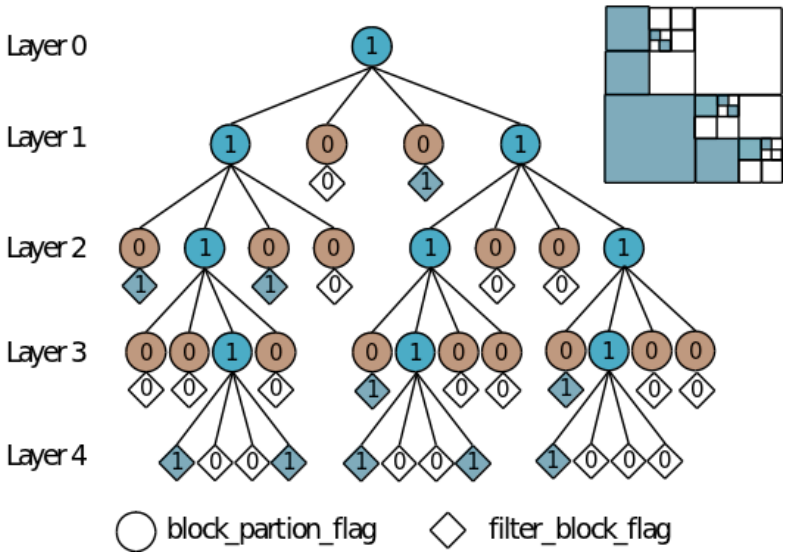


Figure 1.2: An exemplary quadtree structure with the associated block partitions. Source: [6].

anymore, instead the block size can be chosen adaptively. To this end Chujoh et al. proposed to use a quadtree structure to partition the frame into blocks that are filtered and those that are not. Their representation may be illustrated by Figure 1.2. Essentially, the quadtree always contains one flag for the current block or picture. Should the *block_partition_flag* be set to 1, then the current block is split once horizontally and once vertically yielding four subblocks of identical size, that are afterwards examined in turn. Should the current block not be split, then the *filter_block_flag* is used to either enable or disable the filter for the current block. The *block_partition_flag* is omitted, when a fourth layer of partitions has been reached. In which case no block is split further. Details on how an optimal quadtree in the *rate-distortion*-sense may be derived, can be found in [6] and will be reexamined later in Chapter 5. The QALF was tested on a different data set than the BALF and produced an average bit rate reduction of 7.27%. According to Chujoh et al. this corresponds to an improvement of 0.63% compared to the original BALF proposal.

Adaptive In-Loop Noise-Filtered Prediction for High Efficiency Video Coding Most in-loop filters do not cope well with noisy input sequences since they will introduce visible visual differences between original and decoded sequence through the suppression of the original noise. One filtering concept especially designed for such noisy sequences was presented by Wige et al. in [56]. There the authors defined the following description of the error residual after motion-

compensated prediction in a hybrid video codec

$$e[i] = s_g[i] - s_f[i] + n_g[i] - n_f[i], \tag{1.3}$$

where $\hat{f}$ is the prediction signal and g is the original frame. $s_g[i]$, $s_f[i]$ represent the useful signal parts and $n_g[i]$, $n_f[i]$ are the additive noise components. Assuming that the noise between adjacent frames is uncorellated, the authors arrive at the conclusion that the variance of the noise term will generally be increased by the prediction step. They conclude, that a noise suppression in the form of an in-loop filter is needed to counteract this effect. Since the noise is part of the original sequence their filter is only applied after the reconstructed signal has been passed on to the display device. In this manner the original noise is still visible in the reconstructed frame while it is at the same time removed from the reference frames used for prediction. The actual noise filter is a locally adaptive Wiener filter of size 3×3 with low-pass characteristics. The adaptivity is needed to reflect the estimated variance of the noise-free image so as not to introduce additional artifacts. The filter was tested within the HEVC test model HM 2.2 and produced bit rate reductions of up to 22% for the low delay low complexity setting when only P-frames are used and bit rate reductions of up to 4% for the low delay high efficiency setting. Such a gain can, however, only be realized for sequences suitable for the filter, i.e. where a certain amount of additive noise is already present in the original sequence. When B-frames are used instead, significantly lower gains are reported. According to the authors this reflects the noise reduction property of the B-frame prediction step.

1.2.2 Temporal Denoising/Filtering Concepts

From the field of microphone and antenna arrays, it is a well-known fact, that denoising of a signal may be achieved by averaging N noisy samples. As long as the noise is additive and uncorrelated from sample to sample, the noise variance may be reduced through the averaging by a factor N [30]. The underlying statistical signal theory will be revisited in detail later in Chapter 3. In video compression the idea of having several antennas or other capturing devices is instead replaced by the tracking of image content over two or more frames. Filters working in such a manner are generally summarized under the term *temporal denoising filters*. Some prominent examples that share certain properties with the filters to be developed in this thesis will now be examined in more detail.

Three-dimensional Subband Coding with Motion Compensation Among the first publications concerned with temporal filtering is [40], where a three-dimensional subband coding scheme was detailed. This does not only include the filtering

of noisy samples, but also combines this step with the prediction of future samples along a trajectory. Based on the individual trajectory of a single pixel, a decomposition of the color components along the trajectory with the help of a QMF filter bank is performed. Given a trajectory of length R this produces a total of R frequency components per pixel. Prediction and in-loop filtering are consequently done simultaneously, since the predicted reconstructed pixel-value is again derived through the combination of R frequency bands with an FIR filter. In [40] Ohm also discussed cases of uncovered new image details and occluded image regions, which lead to the termination of old or the creation of new trajectories. As described earlier, the main problem in this context is conveying pixel-wise exact motion information to the decoder. A special scheme to transmit quantized trajectory information without strong impairments on the filter quality was also described in [40].

Optimal Motion Compensated Spatio-Temporal Filter A possible implementation of a simple temporal post-filter for Motion JPEG sequences was detailed in [48]. The filter performs block-wise spatial alignment of consecutive frames with motion estimation at the decoder. The described filtering step consists of a simple averaging of three temporally adjacent pixels, i.e. the current and both a future and a past frame are used for filtering. In [49] this idea was extended to a general spatio-temporal filter for H.264/AVC coded sequences that incorporated spatially adjacent pixels from the current frame as well as from past and future frames. In [49] an average PSNR improvement of 0.56 dB was reported for the well-known *foreman* sequence at 172 kbps. Test results for other sequences are, however, missing. In the described implementation the need to transmit motion information was simply omitted by using a block-based motion estimation (ME) algorithm at the decoder. The main drawback of this approach is, of course, the possibility that ME errors will also influence the performance of the filter, thus possibly decreasing the reconstructed picture quality.

Global Motion Temporal Filtering, Glantz et al. As has been mentioned before, very efficient general motion representations can be used, if large parts of a frame move in the same manner. The simplest case would, of course, be a translational motion model applicable to the entire frame requiring only one motion vector (MV) with two components for the frame. In more complex cases, rotational, affine, and perspective motion models with 4, 6, or 8 parameters respectively can be used. In [20] filtering of the image background was achieved by consecutively warping frames into the current frame through the use of homographies transmitted in the bit stream. The construction of the fused image is controlled by an optimal filter length transmitted also as side-information. In addition, the algorithm requires the construction of a binary object mask at the encoder to prevent the filtering of fore-

ground objects. The mask, too, is conveyed via the bit stream. Despite the fact that the filter was applied as a post-filter only, a significant reduction of visual artifacts was achieved. For individual sequences, bit rate reductions of up to 18% were reported. As mentioned earlier, such a filter will only produce satisfactory results on the image background or on very large foreground objects. In case of faulty segmentation masks the filter will severely impair both the objective and subjective quality of the reconstructed frames.

Rate-Constrained Multihypothesis Prediction for Motion-Compensated Video Compression, Flierl et al. Even though it does not constitute a pure filtering approach the method introduced by Flierl et al. in [18] and expanded in [17] should be mentioned here. In the two publications, the authors introduced a new prediction mode for the then state-of-the-art codec H.263 and the experimental codec entitled H.26L, which later became part of H.264. The new mode referred to as *multihypothesis prediction* extended the earlier idea of bidirectionally predicted frames or B-frames to frames and blocks with an arbitrary number of reference frames and an equally arbitrary number of averaged predictors or *hypotheses* for generation of the next frame. The authors restricted these predictors to use only causal information from previous frames in display order, thus reducing the computational overhead. Even though, two and four motion vectors (*hypotheses*) per frame were implemented and tested the authors state that "combining two hypotheses already achieves most of the gain possible with multihypotheses [motion-compensated prediction (MCP)]." [17] The major advantage compared to the simple B-frames with fixed reference frames used in H.263 is the possibility of referencing arbitrary previously coded frames by means of a reference index per frame. The new prediction mode comes close to a temporal filtering mode over several frames since linearly combining several predictors can also be seen as averaging a number of noisy versions of the same pixels. The main difference compared to filtering along an individual trajectory per pixel is, that in both [18] and [17] the same filtering characteristics are applied for an entire motion-compensated block with a fixed number of motion vectors. Trajectory-based filtering will, instead, apply a different filter or at least a different filter length to each pixel individually. This difference will again be examined in Chapter 5.

1.3 Main Contributions

In this thesis, a novel temporal filter will be introduced that reconstructs motion information on pixel level directly from the motion vectors conveyed in the bit stream making it thus independent from additional motion information. Furthermore, due

to its accuracy, the filter can be applied equally to both foreground and background giving it a certain advantage over temporal filters that use parametric motion models. A detailed description of the main developments and findings in this thesis may be taken from the following list of contributions, where all authors mentioned after the first author provided helpful discussions, feedback, and corrections.

1. The original Temporal Trajectory Filter (TTF) introducing a lumiance and a spatial motion consistency threshold was implemented and tested in the environment of the H.264/AVC baseline profile. The algorithm description and the results were published in

   - Marko Esche, Andreas Krutz, Alexander Glantz, Thomas Sikora
     A Novel In-loop Filter for Video-Compression based on Temporal Pixel Trajectories
     Proceedings of the 28th IEEE Picture Coding Symposium (PCS 2010), Nagoya, Japan, 12|07|2010 - 12|10|2010, ISBN: 978-1-4244-7135-5

2. The idea of a one-dimensional motion trajectory for a simple IPPP coding structure was later extended to the more general case of hierarchical B-frames, essentially turning the trajectory path into an entire tree of possible pixel locations that can contribute to the filter efficiency. In addition, a third threshold describing the temporal motion consistency along the trajectory was described and further evaluated in

   - Marko Esche, Andreas Krutz, Alexander Glantz, Thomas Sikora
     Temporal Trajectory Filtering for Bi-directional Predicted Frames
     Proceedings of the 18th IEEE International Conference on Image Processing (IEEE ICIP2011), Brussels, Belgium, 09|11|2011 - 09|14|2011, pp. 1669-1672, IEEE catalog number: CFP11CIP-USB ISBN: 978-1-4577-1302-6
   - Marko Esche, Alexander Glantz, Andreas Krutz and Thomas Sikora
     Adaptive Temporal Trajectory Filtering for Video Compression
     IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), IEEE, vol. 22, no. 5, May 2012, pp. 659-670

3. A new weighting scheme for individual noisy samples contributing to the filtered average as well as a method for ensuring a greater length of individual trajectories was introduced and discussed in

   - Marko Esche, Alexander Glantz, Andreas Krutz, Michael Tok, and Thomas Sikora

Weighted Temporal Long Trajectory Filtering for Video Compression *Proceedings of the 29th IEEE Picture Coding Symposium (PCS 2012)*, Krakow, Poland, 05|07|2012 - 05|09|2012 ISBN: 978-1-4577-2048-2

4. Following the gerneral idea of the QALF a quadtree was added to the TTF making it thus more adaptable to differently moving regions within a frame. The algorithmic details and simulation results may be found in

   - Marko Esche, Alexander Glantz, Andreas Krutz, Michael Tok, and Thomas Sikora
     Quadtree-based Temporal Trajectory Filtering
     Proceedings of the 19th IEEE International Conference on Image Processing (ICIP), Orlando, Florida, 09|30|2012 - 10|03|2012

5. An additional scheme to effectively compress the side-information required for the quadtree-based TTF was presented in

   - Marko Esche, Alexander Glantz, Andreas Krutz, Michael Tok, and Thomas Sikora
     Efficient Quadtree Compression for Temporal Trajectory Filtering
     Proceedings of the 23rd Data Compression Conference (DCC), Snowbird, Utah, 03|20|2013 - 03|22|2013

6. In order to improve the motion vector accuracy, a novel scheme for motion vector interpolation was described in

   - Marko Esche, Michael Tok, and Thomas Sikora
     Adaptive Dense Vector Field Interpolation for Temporal Filtering
     Proceedings of the 20th IEEE International Conference on Image Processing (ICIP), Melbourne, Australia, 09|15|2013 - 09|18|2013

7. Finally, a theoretical justification of the TTF's functionality was given in

   - Marko Esche, Michael Tok, and Thomas Sikora
     Theoretical Considerations Concerning Pixelwise Temporal Filtering
     Proceedings of the 24rd Data Compression Conference (DCC), Snowbird, Utah, 03|26|2013 - 03|28|2013

1.4 Structure of the Thesis

The remainder of the thesis is structured as follows: In Chapter 2 a number of terms and concepts are defined, which will later be used in the description of filter criteria and algorithms. Furthermore, scenarios for using temporal trajectory filters are outlined and the ideas briefly introduced in Chapter 1 are expanded further. The Chapter concludes with a detailed description of the evaluation methods and quality metrics used in the remaining chapters. The slightly unrealistic case of a decoder with perfect knowledge of individual pixel motion is investigated in Chapter 3. The Human-Assisted Motion Annotation Tool as described by Liu et al. in [34] has been used on several MPEG test sequences to generate perfect motion vector fields. A temporal filter with access to the optimal motion data has then been applied to encoded versions of theses sequences at low bit rates. The basic theory of signal denoising through averaging of several motion-compensated pixel copies is examined in Chapter 4. There, achievable bit rate reductions in presence of both noisy samples and noisy motion information are also discussed. The chapter is continued by the prediction of certain filter characteristics which are later evaluated using real-world data. Practical realizations of Temporal Trajectory Filters are presented in Chapter 5. These include mainly the TTF mentioned before and its extended version the Quadtree-based Temporal Trajectory Filter (QTTF). Both were also tested as simple post-processing steps with the associated filter parameters being calculated at the encoder and conveyed as additional side-information in the bitstream. The results for the post-filters together with alternative approaches to derive the filter parameters will be detailed in Chapter 6. There, it will be shown, that despite its increased complexity the application of a temporal filter within the encoding loop is readily justifiable when compared to the post-filter implementation. In Chapter 7 the theoretical considerations from Chapter 4 will be evaluated using the results from Chapter 5. A detailed analysis of correspondences and mismatches between both parts will also be provided. The Chapter finishes with hints on how the theory can be better adapted to real-world scenarios. Chapter 8 summarizes the thesis and its main findings. Furthermore, an outlook concerning future developments and possible improvements of TTF and QTTF is also given.

Chapter 2

Pixel Trajectories: Definitions, Applications, Quality Metrics

Most controversies would soon be ended, if those engaged in them would first accurately define their terms, and then adhere to their definitions. - Tryon Edwards

2.1 Definitions

In order to effectively describe and evaluate both motion vectors, general motion information, and individual pixel trajectories, a number of basic definitions need to be made. In the following, it is implicitly assumed that the encoded video is spatially quantized, e.g. color information is only available at full pixel positions. Furthermore, the video should be encoded in YUV420 format with four luminance samples per chrominance sample U and V respectively. Based on these assumptions the luminance value of a pixel at position $(x, y)^T$ in frame i of a video sequence shall be denoted by $Y_i(x, y)$. The respective chrominance samples are $U_i(x, y)$ and $V_i(x, y)$. Should the video, for example, be represented in in the YUV420 as described above, then the U- and V-channels need to be upsampled and interpolated before being used in the filtering algorithms. For every pixel motion information describing its translation relative to a reference frame shall be available. The motion vector at location $(x, y)^T$ in frame i then consists of an x-component $dx_i(x, y)$ and a y-component $dy_i(x, y)$, see Figure 2.1. If motion information relative to several reference frames is available, a second subscript is used to indicate the reference frame index. The motion vector then contains the components $dx_{i,j}(x, y)$ and $dy_{i,j}(x, y)$ for the $j-th$ available reference frame of pixel $(x, y)^T$ in frame i of the video sequence. The exact ordering of the reference frames here depends, of course, on the encoding structure and the respective choices of the encoder. A temporal pixel trajectory

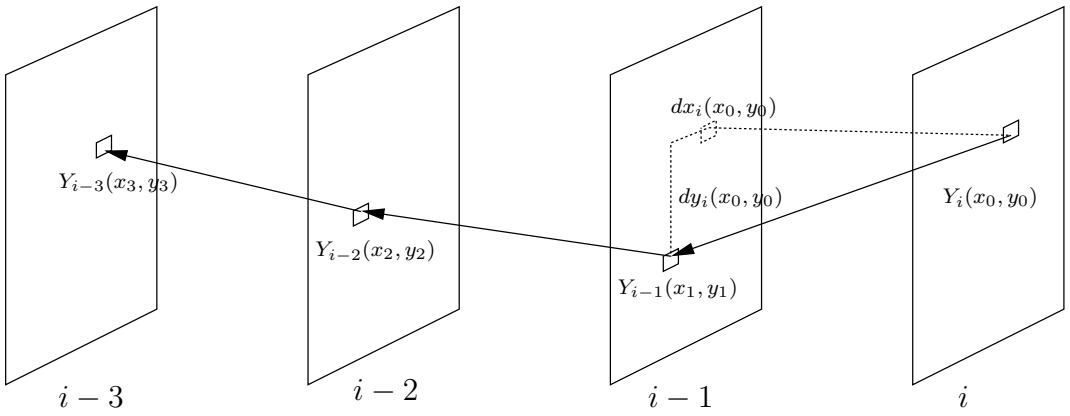


Figure 2.1: The trajectory starts at a point $(x_0, y_0)^T$ in frame i. The associated color components are $Y_i(x_0, y_0)$, $U_i(x_0, y_0)$ and $V_i(x_0, y_0)$. In a traditional IPPP coding structure, $dx_i(x, y)$ and $dy_i(x, y)$ are the motion vector components of frame i at position $(x, y)^T$ relative to frame $i - 1$.

now shall include all pixel positions in all past and/or future frames through which a certain image point moves. Starting with a given pixel $(x_0, y_0)^T$ in frame i its predecessors along the motion trajectory of said pixel shall be $(x_k, y_k), k \geq 1$. However, pixel and frame index are independent from one another. This differentiation is needed, since simply using the frame index does no longer suffice to identify a trajectory element when hierarchical B-frames or some other more complex *group of pictures* (GOP) structure is used. As motion vectors will play an essential role in later chapters some more details and definitions are required in this context.

Both in H.264/AVC and in HEVC motion vectors are encoded by transmitting a motion vector predictor and a residual describing the difference between the motion vector and its predictor [55]. In both codecs a motion vector is found by performing a block-matching algorithm and calculating the *sum of absolute differences* (SAD) between the original block and the motion-shifted one. The motion vector is, however, not simply chosen based on this single criterion. Instead, as in almost all aspects of modern video codecs, *rate-distortion* (RD) optimization is conducted. This combines the expected error residual with the number of bits required for transmitting the actual vector. More details concerning RD-optimization may be found in Section 5.2.2. Due to this optimization a motion vector does not necessarily describe the true translational motion of all pixels within the associated block. Errors occur, of course, at object boundaries with non-rectangular shape. In addition, a block may also be chosen as a reference purely because its SAD is small compared to the original block. In effect, there is no way to guarantee any suitability of transmitted motion vectors with respect to individual pixel trajectories. More details on this matter will

be provided in Chapter 5. Motion vectors in video compression algorithms suffer, unfortunately, from additional restrictions. They can, for instance, refer to a larger number of reference frames. Since the transmitted vectors can thus span a more or less arbitrary number of frames, they are not easily comparable with one another. The following notation shall therefore be used to indicate that a motion vector has been scaled according to the temporal distance it spans: If $(dx_i(x,y), dy_i(x,y))^T$ is a motion vector pointing from frame i to a reference frame $i-k$, its temporally scaled version shall be given by

$$\begin{pmatrix} dx_i(x,y) \\ dy_i(x,y) \end{pmatrix}' = \begin{pmatrix} \frac{dx_i(x,y)}{k} \\ \frac{dy_i(x,y)}{k} \end{pmatrix}. \tag{2.1}$$

It is to be noted here, that k can take on any number $k \in \mathbb{Z}\backslash\{0\}$. Vectors pointing to a future frame (with an associated value of $k < 0$) will thus be inverted and scaled to point to a past frame with a temporal distance of $k = 1$. In addition, encoded motion vectors are generally restricted to quarter-pel precision. During the stages of motion estimation and motion compensation (MC) interpolation filters are employed to calculate image samples at subpixel locations in reference frames. Details concerning the used interpolation filter may be found in [55] and [45].

2.2 Possible Applications

The filter to be developed in this thesis may, of course, be used in the manner already described in Chapter 1. It can, however, also be used in a number of other applications which will be detailed here. For the sake of completeness the in-loop filter is also included.

2.2.1 Temporal In-loop Filter for Video Compression

As has been mentioned in Chapter 1 an in-loop filter can (according to List et al. [33]), if applied correctly, increase the subjective quality of the decoded video. Since reconstructed frames are also used for the prediction of future frames the prediction signal will also be of better quality and the bit rate required for transmitting the residual is potentially reduced by an in-loop filter. The overall bit rate needed to transmit a video at a predefined quality level will thus also be decreased. An in-loop filter also ensures that all decoders generate output sequences with identical quality, since bypassing the in-loop filter will result in an asynchronous state between encoder and decoder and is thus not an option. A block diagram of an HEVC encoder with an added TTF may be found in Figure 2.2. The main drawback of an in-loop filter is

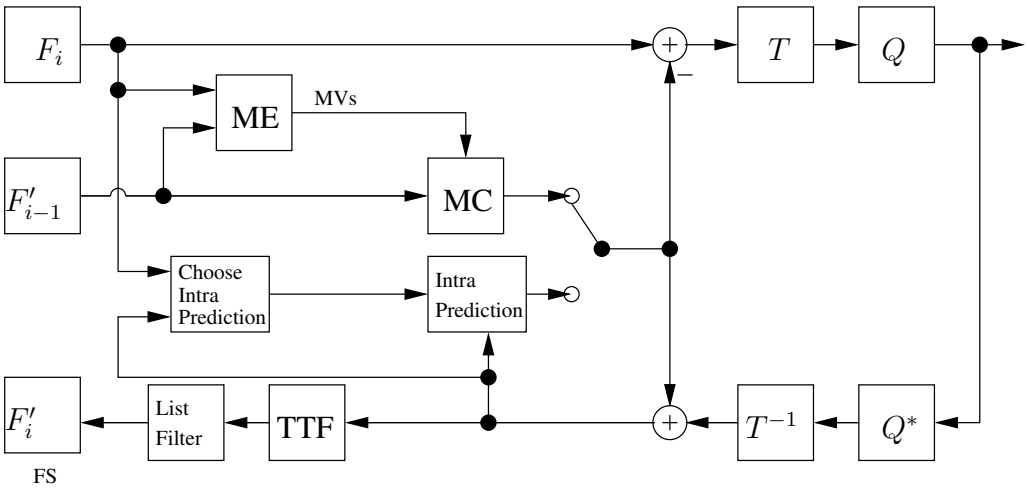


Figure 2.2: General encoder diagram based on H.264/AVC. The TTF is employed within the local decoding loop at the encoder.

the increased encoder and decoder complexity. Since a decoder is generally much less complex than an encoder in a broadcast scenario, the impact of an in-loop filter on its average runtime will be even more significant. This statement, however, only holds true if no encoder-side optimization requiring multipass optimization is necessary. In [33] for example it was stated that the deblocking filter contributed to one third of the total decoder runtime. For the QALF, however, an increased decoder runtime of 146% was reported [35]. With QALF included the encoder was also up to 90% more complex. As most video coding applications assume the existance of many more decoding devices than encoding devices, the main concern while designing an in-loop filter should actually be the complexity of the decoder.

2.2.2 Temporal Post-Filter for Video Denoising

A second option to use a temporal denoising filter, which will also be investigated in this thesis, is its employment as a post-processing step. In this case, the usage of the filter is non-mandatory, thus giving the decoder the choice to switch it either on or off. Apart from the obvious advantage of a generally less complex decoder, this scheme has several drawbacks:

- As will be shown later in this thesis, unsupervised temporal filtering without input from the encoder bears the risk of introducing new artifacts. This is especially the case, since the encoder will in general not be able to distinguish

between parts of the frame suitable for filtering and unsuitable ones.

- If an encoder-side optimization is carried out instead, the side information will increase the size of the bitstream without the guaranteed subjective and objective improvements of the decoded picture quality that are present in the in-loop case.

- The subjective or objective quality improvements at the decoder, if any, have no impact on the required bit rate. The gain produced in a RD-sense will also be a lot smaller since the transmitted residual is unaffected by the filter thus keeping the bit rate constant.

2.2.3 Temporally Denoised Prediction Mode for Video Compression

As has been described in [40], it is possible to predict and encode a video sequence along individual pixel trajectories. Such schemes are, however, only under certain conditions able to outperform the traditional hybrid video coding approach. One way to combine both temporal filtering and prediction is constituted by the B-frames that were first introduced in H.262/MPEG 2 Part 2 [26]. Since in a B-frame two motion vectors can exist per block and the reference blocks are combined, the predicted signal is a temporally filtered version with a trajectory length of 2. The same could of course be done with multiple reference frames per block. The pixel could be filtered and denoised prior to the construction of the predicted frame. In this case the transmitted residual for the current frame will also be reduced. It is exactly this feature that distinguishes the prediction mode from the simpler in-loop filter since the latter only affects the bit rate of future frames. Despite not using a temporal filter, the approach by Wige et al. [56] comes very close to the concept of a denoised prediction mode.

2.3 Quality Metrics

When discussing the quality of pixel trajectories two aspects have to be distinguished:

- In how far is the pixel trajectory identical with the true motion of the displayed object? Since only two-dimensional image data is generally available the reprojection of a three-dimensional trajectory is actually computed. The error can then be determined between the reprojection of the true motion and

the reprojection of the predicted motion. The error will correspond to the euclidean distance between two alternative or coinciding pixel locations in every frame.

- In which way does the calculated trajectory contribute to an improvement of the encoded video sequence in the RD-sense? This corresponds in essence to an evaluation of the filter performance of the TTF. The main difference compared to the first point is the fact, that a trajectory does not need to be correct in order to produce good results. It will be shown in the experimental evaluation that even faulty trajectories may contribute to the efficient removal of coding artifacts.

2.3.1 Evaluating the Correctness of a Trajectory

Since the available data in a video sequence is inherently two-dimensional, the difference between the true trajectory and an approximate one can be measured in the $\mathbb{R}^2$, too. Given a set of correct pixel locations $(x_i, y_i)^T$, $i = 0, ..., N-1$ in N consecutive frames and a corresponding set of N approximate pixel locations $(x'_i, y'_i)^T$ the average Euclidean distance between both trajectories is given by

$$\overline{e} = \frac{1}{N} \sum_{i=0}^{N-1} \sqrt{(x'_i - x_i)^2 + (y'_i - y_i)^2}. \tag{2.2}$$

The resulting error can directly be expressed in pixel distances. The main drawback of this method is twofold:

- The measure is not scale invariant. Even though the relative error in an HD sequence (compared for instance to the width of a frame) may be small, the measure will be a lot bigger compared to a similar trajectory in a frame of smaller size.

- In order to evaluate the approximated locations $(x'_i, y'_i)^T$, the original correct locations $(x_i, y_i)^T$ need to be known first. This kind of ground truth data is, however, not generally available for real-world sequences. Using synthetic sequences with known camera motion instead, solves this problem, although such sequences often differ significantly from realistic ones in terms of the achievable compression ratios. This is mainly due to the textures used in such sequence and also due to the absence of camera noise.

The error measure $\overline{e}$ is commonly also referred to as the average endpoint error (AEE). A way to determine ground truth data for arbitrary real-world sequences

has been detailed in [34]. The human-assisted method described therein will be used in Chapter 3 to derive pixel-precise optical flow fields for the sequences used in the experimental evaluation. Another commonly used performance measure for the evaluation of motion accuracy in optical flow algorithms is the average angular error (AAE) [3]. A two-dimensional motion vector $(dx, dy)^T$ can be represented by a normalized vector in 3D space

$$\vec{r} = \frac{1}{\sqrt{dx^2 + dy^2 + 1}} \begin{pmatrix} dx \\ dy \\ 1 \end{pmatrix}. \tag{2.3}$$

The angular error (AE) is then given by

$$\mathrm{AE} = \arccos\left(\vec{r}_t^{\,T}\, \vec{r}_e\right), \tag{2.4}$$

where $\vec{r}_e$ is the estimated and $\vec{r}_t$ the true motion vector. The added advantage of this measure is that it simultaneously combines direction and magnitude of the motion information. However, identical positive and negative deviations from the true value yield different AE values. Moreover, the magnitude of the motion vector will also influence the interpreted angle. The AAE is here used as a quality measure, instead, since for temporal filtering motion vectors are selected if they point to correct image locations inspite of their angle.

2.3.2 Evaluating the Image Quality

Similar to the mean quadratic error for a trajectory an MSE between a reconstructed filtered image $Y'(x, y)$ and an original image $Y(x, y)$ can be defined as:

$$\mathrm{MSE} = \frac{1}{H \cdot W} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} \left(Y'(x, y) - Y(x, y)\right)^2 \tag{2.5}$$

for an image of height H and width W. A common measure derived from the MSE is the PSNR expressing the MSE in dB instead

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{MSE} \tag{2.6}$$

for an image with luminance components in the range of 0 to 255. Corresponding measures for the two chrominance components can easily be defined. Over an entire video sequence the PSNR can of course be averaged resulting in a single overall PSNR value for the sequence. However, in order to compare encoded sequences with one another, the PSNR alone is not suitable. A PSNR difference is only meaningful if

both sequence and reference sequence have been encoded at the same bit rate. Due to the generally non-linear relationship between bit rate and picture quality, a measure is needed which combines both over a wide range of quality levels. Such a scheme was detailed by Bjøntegaard in [4]. According to Bjøntegaard simply averaging the PSNR for different quantization parameters (QP) will always favor those values at higher bit rates. He proposed to compute the integral between two curves instead and to divide the integral by the integrated interval. To this end both the original and the reference curve need to be mathematically approximated. As was shown in [4] this is most easily achieved when a double-logarithmic scale (PSNR in dB over logarithmic bit rate) is used. The resulting measure can either be expressed as an average bit rate reduction at identical quality or as a quality improvement at identical bit rates. Both terms known as $BD-\text{PSNR}$ and $BD-\text{rate}$ are commonly used in the video coding community. They are the main optimization criteria in both H.264/AVC and in HEVC. On the other hand, some researchers argue [50] that MSE and PSNR poorly represent image quality as it is perceived by the human visual system (HVS). One metric that is supposed to close this gap is the *structural similarity* (SSIM) index introduced in [51], which is based on the finding, that "the human visual system is highly adapted to extract structural information from visual scenes." This is achieved by analyzing mean, variance, and covariance between two images I_1 and I_2 per pixel yielding the local SSIM [51]:

$$
\begin{aligned}
(2.7)\quad S(I_1, I_2) &= l(I_1, I_2) \cdot c(I_1, I_2) \cdot s(I_1, I_2) \\
&= \left(\frac{2\mu_{I1}\mu_{I2} + C_1}{\mu_{I1}^2 + \mu_{I2}^2 + C_1} \right) \cdot \left(\frac{2\sigma_{I_1}\sigma_{I_2} + C_2}{\sigma_{I1}^2 + \sigma_{I2}^2 + C_2} \right) \cdot \left(\frac{\sigma_{I1I2} + C_3}{\sigma_{I1}\sigma_{I2} + C_3} \right)
\end{aligned}
$$

Here μ_{I1} and μ_{I2} are the local sample means. σ_{I1} and σ_{I2} are their standard deviations and σ_{I1I2} is the local sample covariance. The SSIM value for the entire image is computed by averaging the local values over one frame, where the local values are computed within a sliding window. Figure 2.3 taken from [50] illustrates the superiority of SSIM over MSE in presence of various different kinds of noise.

The MSE for images (a) to (g) is nearly identical, whereas the SSIM approximates the human perception of the images much better. Of particular interest is image (f) depicting JPEG compression artifacts, which strongly impair the visual quality despite a relatively low MSE. The given value of the CW-SSIM originates from a wavelet-based extension of SSIM that is more robust towards translations, rotations, and scalings of images. However, during the standardizations of both H.264/AVC and HEVC the reference measure has always been PSNR. Moreover, a temporal equivalent of the SSIM does not yet exist. The visual quality measure of choice in this thesis will therefore also be the standard PSNR.

Figure 2.3: "Comparison of image fidelity measures for "Einstein" image altered with different types of distortions. (a) Reference image. (b) Mean contrast stretch. (c) Luminance shift. (d) Gaussian noise contamination. (e) Impulsive noise contamination. (f) JPEG compression. (g) Blurring. (h) Spatial scaling (zooming out). (i) Spatial shift (to the right). (j) Spatial shift (to the left). (k) Rotation (counter-clockwise). (l) Rotation (clockwise)." Source: [51].

2.4 Chapter Summary

This Chapter defined several mathematical terms to describe pixel trajectories over a given set of frames. These will be used in later Chapters to derive criteria for testing and evaluating motion information to be added to a trajectory. Furthermore, different quality measures for inspecting both the pixel trajectories and the filtered images have been discussed.

Chapter 3

Filtering in Presence of Perfect Motion Knowledge

Perfection is not attainable, but if we chase perfection we can catch excellence.
- Vince Lombardi

Before investigating real-world implementations of a temporal trajectory filtering approach, the theoretical potential of such an algorithm shall be highlighted. To this end, optimal pixel-wise motion information is generated and used for filtering both at encoder and decoder of a video transmission system. The challenges encountered during this investigation as well as the achieved results are presented here.

3.1 Human-Assisted Motion Annotation

In order to obtain precise pixel-wise motion information for non-synthetic sequences, optical-flow algorithms can be used [3]. However, these cannot produce satisfactory results when occlusions occur or new image content is introduced. One possibility to improve the generated motion field is to provide the optical flow-algorithm with object masks. Unfortunately, automatically generated object masks are also usually imperfect, especially when objects are partly occluded or do not move for several frames [34]. Subsequently, human input is unavoidable to generate high-quality object masks for arbitrary video sequences. One tool that combines both human a priori knowledge about moving objects and state-of-the-art optical flow estimation was presented in [34]. The tool described by Liu et al. provides motion-fields of such good quality that it is commonly used as a bench-mark to evaluate other (automatic) optical-flow algorithms. The following description of the "Human-Assisted Motion

Annotation“ (HAMA) algorithm is based on [34]. The algorithm consists of four major steps.

1. **Semi-automatic layer segmentation:** The user manually specifies boundaries for all foreground objects of a video sequence in one or more frames. The system then provides preliminary object masks for all other frames. Occlusion handling in this case is aided by additional user input specifying the object's depth in certain keyframes.

2. **Automatic layer-wise flow estimation:** Based on the seperate masks for foreground and background individual flow fields per mask (also referred to as layers) are generated. The tool offers the user the possibility to manually select certain parameters of the underlying optical flow algorithm. By backprojecting the pixels from one frame to the next, the user can evaluate the accuracy of the motion field.

3. **Semi-automatic motion labeling:** Should the user not be satisfied with the matches produced during the last step, he or she can specify sparse correspondences between neighboring frames. These are then used to either interpolate a dense motion field or to generate a parametric global motion model.

4. **Automatic Compositing:** The final step of the Human-Assisted Motion Annotation consists of merging all seperate object-specific motion fields into one optical-flow field per frame. As a byproduct highly accurate object masks are also generated.

The authors of [34] state, that their method generates motion fields that are much closer to the ground truth than the output of any other available tool based on a number of sequences publicly available. Moreover, the output of the method appears to be consistent over a number of different test subjects. For these reasons, the above-mentioned tool is applied here to generate ground truth motion data for the test sequences used in Chapter 5. An exemplary object shape generated by the HAMA tool for the *BlowingBubbles* sequence is shown in Figure 3.1(a). The corresponding motion vector field (showing the x-component in grayscale) may be found in Figure 3.1(b). To first demonstrate the general properties of temporal noise induced by video compression, the following two sections analyze the noise in a simple synthetic test sequence and analyze some publicly available real-world sequences.

(a) Labeled contours for frame 2 of the *BlowingBubbles* sequence.

(b) Ground truth motion vector field for frame 2 of the *BlowingBubbles* sequence (x-component in grayscale).

Figure 3.1: Output of the HAMA algorithm for the *BlowingBubbles* sequence consisting of a set of object masks (*left*) and motion vector fields (*right*).

3.2 Theoretical Foundations

In statistical signal theory it is a well-known fact that, if several noisy versions of a signal are available, noise reduction can be achieved by averaging all versions of the signal. Important applications of such noise reduction concepts include microphone arrays and antenna array systems. In the case of time-dependent signals such as audio, speech or video, temporal or spatial alignment respectively need to be performed prior to filtering. $Y_i(x, y)$, $U_i(x, y)$, and $V_i(x, y)$ shall denote the luminance and chrominance components of the i-th frame of a video sequence in display order. Initially, it is assumed that for a given pixel $(x_0, y_0)^T$ in frame j, $N-1$ previous locations $((x_1, y_1)^T, ..., (x_{N-1}, y_{N-1}))$ of the image point in $N-1$ past frames are known and the pixel amplitudes remain identical, i.e. the illumination of the scene does not change $Y_i(x_0, y_0) = Y_{j-i}(x_i, y_i)$. Each of these shall be distorted by i.i.d. additive white noise n_i with variance σ_N^2

$$\hat{Y}_{j-i}(x_i, y_i) = Y_{j-i}(x_i, y_i) + n_i. \tag{3.1}$$

The autocorrelation function of such a noise term is given by

$$R_{NN}(k, l) = E\left[n_k n_l\right] = \begin{cases} \sigma_N^2, & \text{if } k = l \\ 0, & \text{else.} \end{cases} \tag{3.2}$$

3.2.1 Noise Reduction Through Temporal Filtering

A filtered version of pixel $\hat{Y}_j(x_0, y_0)$ can then be computed:

$$Y_{\text{opt},j}(x_0, y_0) = \frac{1}{N}\sum_{i=0}^{N-1} \hat{Y}_{j-i}(x_i, y_i) \tag{3.3}$$

$$= \frac{1}{N}\sum_{i=0}^{N-1} Y_{j-i}(x_i, y_i) + \frac{1}{N}\sum_{i=0}^{N-1} n_i \tag{3.4}$$

$$= \frac{1}{N}\sum_{i=0}^{N-1} Y_j(x_0, y_0) + \frac{1}{N}\sum_{i=0}^{N-1} n_i \tag{3.5}$$

The noise variance σ^2_{opt} of the filtered pixel is reduced by a factor of N:

$$\sigma^2_{\text{opt}} = \frac{1}{N^2} E\left[\sum_{i=0}^{N-1} n_i \sum_{k=0}^{N-1} n_k\right] \tag{3.6}$$

$$= \frac{1}{N^2}\sum_{i=0}^{N-1} \sigma^2_N = \frac{\sigma^2_N}{N}. \tag{3.7}$$

In this context, it is assumed that pixel amplitudes and noise are not correlated. Even if such correlation exists, filtering is still successful but with reduced filter gain. This simple example illustrates how noise introduced by block-artifacts can be reduced if the motion of every image point in a video sequence is known. It remains to be shown that the noise term does indeed have the qualities mentioned above.

3.2.2 Image and Noise Model

For simplification it is assumed that the two-dimensional integer-step image content of a video sequence only has translational motion. In this ideal case, every image point from the first frame is also present in all other frames and can thus be assigned a unique motion trajectory. This model, however, does not account for occlusions, non-translational motion or the introduction of new image content. In real-world sequences, therefore, trajectories for individual pixels can only be calculated over a smaller number of frames. Nevertheless, the following example shall illustrate the properties of block-artifacts and the potential of applying a temporal filter to every pixel in a decoded sequence. A synthetic sequence is used for this purpose. In the sequence, a gray circle moves from left to right in front of a black and white checkered pattern. Figure 3.2 shows frame 64 of the original sequence. Even in the absence of camera noise, block-artifacts are expected to occur at the object

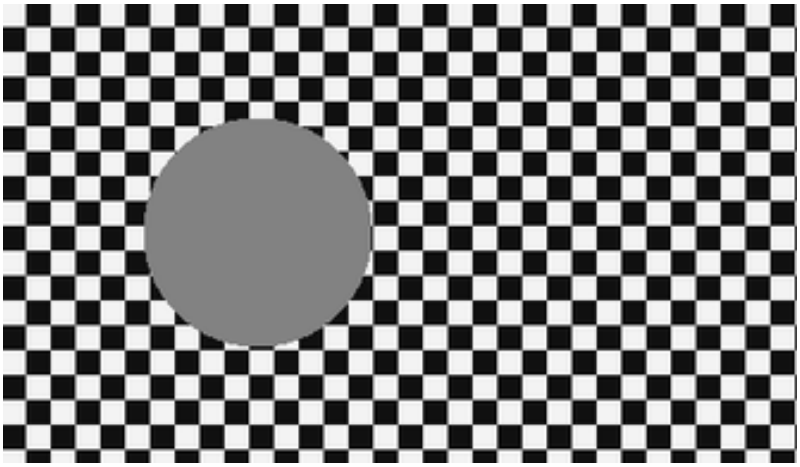


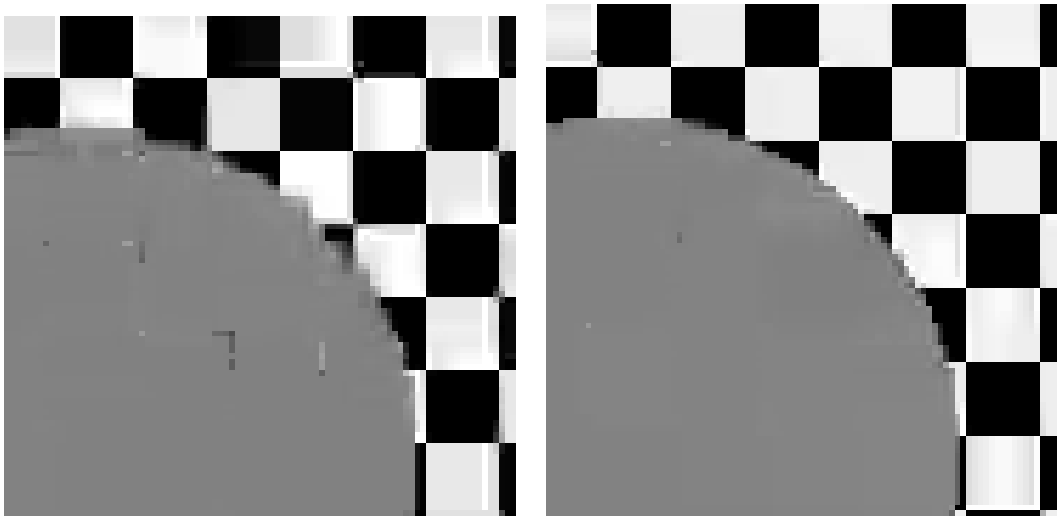
Figure 3.2: The synthetic test sequence shows a gray circle moving from left to right in front of a black and white checkered background.

boundaries if the sequence is encoded with the H.264/AVC baseline profile (IPPP GOP structure). For comparison see Figure 3.3(a), where an enlarged part of frame 64 of the decoded sequence for QP 47 is shown. As expected, noise is especially present at the edge of the circle. Since in this case the true motion of all pixels is known, a motion-compensated noise per pixel can be calculated. The autocorrelation function of the noise superimposed over one exemplary pixel from the circle boundary is displayed in Figure 3.4. Similar autocorrelation functions can be observed for almost all other pixels at the object boundary, which is a strong indication, that the temporal noise introduced by block-artifacts does indeed show properties of white noise. This property is also supported by the fact that extensive block flickering is apparent at the edge of the circle. Denoising can now be achieved by replacing every pixel in the decoded sequence with its motion-compensated average over the previous frames. The result of this operation, taking eight past frames into account, is depicted in Figure 3.3(b). Although the analysis undertaken on the synthetic sequence does not provide conclusive proof of the initial assumptions, it at least justifies further investigation of the temporal trajectory filtering approach. As will be shown in Chapter 7 the image and noise models used here are also applicable to real-world scenarios. The mean noise variance of the pixels close to the boundary of the depicted circle has also been measured:

$$\sigma_N^2 = 572.21 \tag{3.8}$$

$$\hat{\sigma}_N^2 = 77.30 \tag{3.9}$$

Here σ_N^2 denotes the variance of added noise in the temporal domain of the raw, decoded sequence. $\hat{\sigma}_N^2$ is the variance of the noise present in the filtered sequence. The ratio of both values $\frac{\sigma_N^2}{\hat{\sigma}_N^2} = 7.39$ is close to the theoretically assumed value of



(a) Decoded picture (b) Filtered picture

Figure 3.3: On the *left* an enlarged part of frame 64 of the decoded sequence is shown. The image on the *right* shows the same part after the application of a motion-compensated temporal filter.

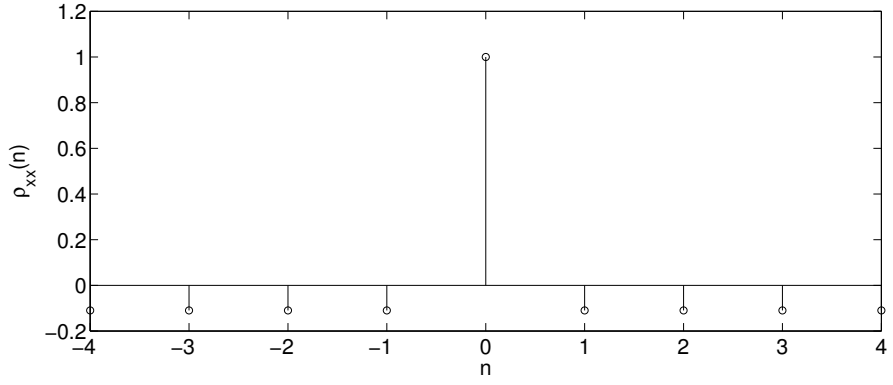


Figure 3.4: The autocorrelation function for the trajectory of one exemplary pixel of the circle boundary.

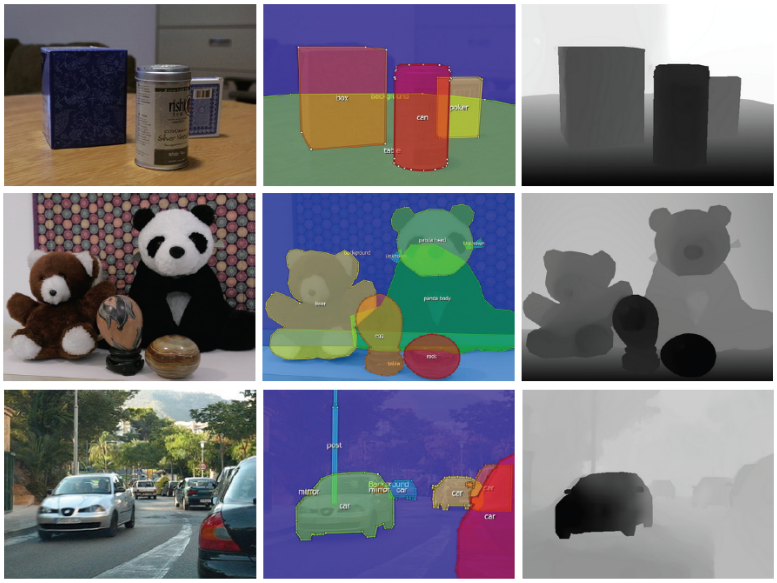


Figure 3.5: Original frame, layer segmentation, and color-coded x-component of the optical flow for the sequences *Sample1*, *Sample3*, and *CameraMotion*. Source: [34].

$N = 8$ corresponding to the number of frames used for averaging. This finding again indicates, that the noise introduced by the codec has the properties of white noise.

3.2.3 Investigation of Real-World Sequences

In order to investigate the properties of temporal noise in real-world video sequences, it becomes necessary to determine the motion trajectory for every pixel in a given video sequence. To this end, the dataset provided by Liu et al. in [34] was used consisting of a number of real-world sequences together with accurate object masks and optical flow information. All of these sequences were segmented and annotated by the user-aided algorithm described in [34], which is frequently used as a benchmark for fully automatic optical-flow algorithms. In the context of this thesis, only segmentation masks and the optical flow field with subpel accuracy were used. Exemplary frames together with the layer segmentation and the color-coded x-component of the flow field are shown in Figure 3.5. Each sequence has been encoded with the H.264/AVC baseline profile (IPPP GOP structure) at QP 47. Afterwards the trajectory for all boundary pixels of each foreground object was calculated using the provided motion information. Based on the per-pixel difference between original sequence and encoded sequence the spectral flatness measure γ_X^2 as defined in [41], [30] was determined. After calculating the power density spectrum $S_{\mathrm{XX}}(k\Delta\Omega)$ as

the Fourier transform of the autocorrelation function, the spectral flatness measure is given by

$$\gamma_X^2 \approx \frac{\left[\prod_{k=1}^{N} S_{\mathrm{XX}}(k\Delta\Omega)\right]^{\frac{1}{N}}}{\frac{1}{N}\sum_{k=1}^{N} S_{\mathrm{XX}}(k\Delta\Omega)},\ \Delta\Omega = \frac{2\pi}{N}. \tag{3.10}$$

In theory γ_X^2 lies between 0 and 1, where $\gamma_X^2 = 0$ describes a fully predictable data source, while a value of $\gamma_X^2 = 1$ indicates that the source has the properties of white noise. For the sequences *Sample1* and *Sample3* an average γ_X^2 of 0.93 and 0.95 were observed. For the *CameraMotion* sequence the value lay at 0.89. These values, however, only apply to object boundaries. Within a foreground object itself or for background pixels the observed correlation between temporally adjacent error samples is significantly higher with an average γ_X^2 of 0.6. For the synthetic sequence described above a γ_X^2 of 0.95 was observed for all image points within a 3-pixel radius around the object boundary.

3.3 Achievable Bit Rate Reductions with Perfect Motion Vector Fields

In presence of perfect motion knowledge, the performance of a temporal denoising or deblocking filter will of course be improved. However, in order to compare the observed results with tests based on real datasets, certain restrictions have to be applied. In this section, consequences from using an ideal motion vector field are considered. Moreover, the term "ideal" needs to be examined and precisely defined in this context, since perfect motion does not necessarily provide the best results, as will be shown. Nevertheless, it is possible to quantitavely compare the performance of the implemented QTTF algorithm [10] with the theoretically achievable maximum gain and thus to underline the effectiveness of the current implementation.

3.3.1 Preliminary Considerations

In order to evaluate the QTTF's ability to construct pixel trajectories from block-based motion vectors, the HM implementation will be compared with an identical version where only the motion vector data has been replaced. However, since the HAMA tool only provides one motion vector per pixel, the reference implementation of QTTF also needs to be modified so that only on motion vector per block is used. In addition, the HAMA motion vectors have floating point precision. Quantizing

these back to quarterpel would seriously corrupt the motion data. Thus a new interpolator had to be integrated into HM 3.0. The new spline-based interpolator then had to be used to generate new reference data, since the observed gain might otherwise have been due to the change of the interpolation scheme. For all subsequent experiments a third order spline interpolator was used.

Another restriction, that had to be placed on the encoder, concerns the range of available reference frames. Again, as HAMA only performs frame-wise optical flow calculations the reference frame, for comparability, also needs to be the previous frame. One additional problem is posed by the actual image content. Within the *BlowingBubbles* sequence, for example, specularities and reflections regularly occur. Even if one of the soap bubbles (see Figure 3.6 for details) is properly segmented and tracked, individual pixel trajectories originating from the background might still pass through the bubble. As a consequence it becomes necessary to distinguish

Figure 3.6: Since the soap bubbles visible in the sequence *BlowingBubbles* are mostly transparent, the motion vectors in these areas are unreliable, even if the bubbles are properly segmented.

between a semantically correct object trajectory and a visually correct pixel trajectory. One implication of this is, that all frames of the *BlowingBubbles* sequence featuring large soap bubbles as foreground objects, are essentially unsuitable for the evaluation of the proposed methods.

Sequence	QTTF		QTTF perfect	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BlowingBubbles	0.01	−0.35	0.03	−0.70
BQSquare	0.12	−2.62	0.14	−3.01
RaceHorses	0.00	−0.04	0.00	−0.05
Waterfall	0.16	−6.13	0.19	−7.22

Table 3.1: Comparison between QTTF and QTTF with perfect motion vectors.

3.3.2 Performance Comparison Between QTTF and QTTF with Perfect Motion Vector Fields

The results obtained with QTTF utilizing perfect motion vector fields compared to QTTF utilizing the HEVC-generated motion field are shown in Table 3.1. As can ben seen the QTTF with perfect motion fields always performs better than the original QTTF implementation. However, for *RaceHorses* both versions only achieve a very minor gain. In this sequence a lot of motion blur is present making the exact definition of object boundaries difficult even for humans. Moreover, even if objects are correctly tracked, the filtering may reduce the motion blur that is present in the original sequence by including frames without motion in the filtering process and thus introducing details not present in the blurred frame. The filtered video will thus have of course a greater MSE compared to the original. For *BlowingBubbles* the QTTF achieves around 50% of the theoretically possible bit rate reduction. In the case of the *BQSquare* sequence the achieved gain reflects around 87% of the gain that could ideally be created. These values correspond nicely with the average motion estimation errors that are given in Appendix C and further discussed in Chapter 7. From these it can be seen that for *BQSquare*, for instance, the encoder conveyes much more accurate motion information to the decoder which accounts for the larger gain achieved for this sequence. An identical scenario can be observed for the *Waterfall* sequence. Here, a much larger portion of the frame is filtered (see Chapter 7 and a much larger average bit rate reduction is produced. Again the QTTF realizes about 85% of the theoretically achievable gain. Both in *BQSquare* and *Waterfall* sequences the QTTF profits from the absence of large foreground objects, which tend to introduce bigger motion estimation errors.

3.4 Chapter Summary

In this Chapter the Human-Assissted Motion Annotation Tool [34] was described in detail and later utilized to provide pixel-based accurate motion vectors for four

seperate test sequences. On these both a modified version of the TTF and a theoretically optimal temporal filter were tested. The modifications of the TTF were necessary to objectively compare both filters under identical testing conditions. For this reason the usage of B-frames was prohibited and HM 3.0 was essentially configured to work in the same manner as the H.264/AVC baseline profile. For these settings the TTF provided around 50% of the gain produced by the optimal temporal filter. Of course, the perfect motion vector field does not include control instructions to terminate trajectories in case of occlusions or when new content is introduced in a frame. For this reason, both filters use side-information to restrict the trajectory length where necessary.

Chapter 4

Theoretical Considerations

Ponder Stibbons was one of those unfortunate people cursed with the belief that if only he found out enough things about the universe it would all, somehow, make sense. The goal is the Theory of Everything, but Ponder would settle for the Theory of Something and, late at night, [...] he despaired even a Theory of Anything. - Terry Pratchett

A paper describing the one-dimensional case of the theory presented in this section was first published in [16].
Temporal motion compensated filtering is based on the assumption that the visible content of a video sequence only changes slowly over time and that multiple instances of the same image point in consecutive frames can be seen as identical noisy copies of one another. This idea also finds application in the areas of noise reduction in audio sequences with microphone arrays [43], 2D/3D conversion with structure-from-motion [32], and in video coding with the introduction of B-frames [55]. Denoising can then be achieved by averaging the color components of such an image point along its spatiotemporal path, also referred to as a trajectory. The TTF can efficiently reconstruct such trajectories from the quantized motion data transferred in the bit stream. To this end, the TTF adaptively concatenates available motion vectors. However, a general theoretical justification of the filtering concept has yet to be presented. This chapter aims to provide a theoretical analysis of the TTF's performance based on simple video characteristics. This will result in the prediction of optimal filter lengths per quantization parameter and sequence. A comparable study for global motion parameters was presented in [8]. Similar analytical error models and models for the accuracy of motion estimation may be found in [29] and [2]. An approach that tries to apply statistical signal theory to the similar problem of motion estimation may be found in [38].

4.1 Maximum Theoretically Achievable Gain in Case of Translational Motion

The luminance values per pixel location in an arbitrary frame shall again be given by $Y(x, y)$. These shall be subject to an isotropic 2D-AR(1) correlation model [5]:

$$E\left[Y(x,y)Y(x+\Delta x, y+\Delta y)\right] = \sigma_Y^2 \alpha_x^{|\Delta x|} \cdot \alpha_y^{|\Delta y|}, 0 < \alpha_x < 1, 0 < \alpha_y < 1. \tag{4.1}$$

Here, σ_Y^2 is the luminance variance of the current frame. $|\Delta x|$ and $|\Delta y|$ are the horizontal and vertical offsets between two pixels from said frame. In the publication by Chen and Pang [5] it is assumed that the two correlation coefficients are identical $\alpha_x = \alpha_y$. The experimental verification in Chapter 7 will demonstrate that this is indeed the case for almost all tested sequences. The authors of [5] also produced evidence for the accuracy of the separable correlation model. They additionally showed that the same correlation model is accurate for the motion compensated frame difference image where the model is superimposed with spatial white noise. The filtering along a temporal trajectory will now be examined analytically for three special cases:

1. All trajectories are perfectly known but the reconstructed images are distorted by noise.

2. All images have been reconstructed perfectly but the trajectories are distorted due to motion estimation errors. This is a case in which the decoder will decrease the image quality due to filtering.

3. The combined case where both trajectory locations and images are noisy.

4.1.1 Perfect Knowledge of all Trajectories and Noisy Images

The location $(x_0, y_0)^T$, at which a trajectory starts, can without loss of generality be set to $(0, 0)^T$. In addition, if motion compensation has already been performed, all locations $(x_n, y_n)^T$ in subsequent frames are identical to the starting point

$$x_n = x_0 = 0, y_n = y_0 = 0. \tag{4.2}$$

Noise is added to all pixels the trajectory passes through

$$Y'(x_n, y_n) = Y(x_n, y_n) + \eta_n = Y(x_0, y_0) + \eta_n. \tag{4.3}$$

The noise term η_n can be assumed to be i.i.d. white noise with zero mean [11], see Chapter 7 for further details. If a non-zero mean occurs, the mean of the error cannot be corrected through temporal filtering. Otherwise, averaging of the noisy samples yields

$$\hat{Y} = \frac{1}{m}\sum_{i=1}^{m} Y'(x_i, y_i) \tag{4.4}$$

$$= \frac{1}{m}\left(\sum_{i=1}^{m} Y(x_i, y_i) + \sum_{i=1}^{m} \eta_i\right) \tag{4.5}$$

$$= \frac{1}{m}\left(\sum_{i=1}^{m} Y(x_0, y_0) + \sum_{i=1}^{m} \eta_i\right). \tag{4.6}$$

Subsequently, the error variance after filtering can be calculated. It is safe to assume that the mean of the error is zero, since $\frac{1}{m}\sum_{i=1}^{m} E\left[\eta_i\right] = 0$ for $E\left[\eta_i\right] = 0\ \forall i$.

$$\begin{aligned} E\left[(\hat{Y} - Y)^2\right] &= E\left[\left(\frac{1}{m}\sum_{i=1}^{m} Y(x_0, y_0) + \frac{1}{m}\sum_{i=1}^{m} \eta_i - Y(x_0, y_0)\right)^2\right] \\ &= E\left[\left(\frac{m}{m} Y(x_0, y_0) - Y(x_0, y_0) + \frac{1}{m}\sum_{i=1}^{m} \eta_i\right)^2\right] \\ &= \frac{1}{m^2} E\left[\eta_1 \cdot \eta_1 + \eta_1 \cdot \eta_2 + ... + \eta_1 \cdot \eta_m + \eta_2 \cdot \eta_1 + \eta_2 \cdot \eta_2 + ...\right. \\ &\quad \left. ... + \eta_2 \cdot \eta_m + ... + \eta_m \cdot \eta_1 + \eta_m \cdot \eta_1 + \eta_m \cdot \eta_2 + ... + \eta_m \cdot \eta_m\right] \\ &= \frac{1}{m^2}\left(E\left[\eta_1^2\right] + E\left[\eta_2^2\right] + ... + E\left[\eta_m^2\right]\right) \end{aligned} \tag{4.7}$$

As all noise terms are uncorrelated with identical variances σ_η^2 the term can be simplified as follows:

$$E\left[(\hat{Y} - Y)^2\right] = \frac{m}{m^2}\sigma_\eta^2 = \frac{\sigma_\eta^2}{m}, \quad \lim_{m\to\infty} E\left[\left(\hat{Y} - Y\right)^2\right] = 0. \tag{4.8}$$

In the case described above the luminance error variance approaches zero if the length of the trajectory goes to infinity. In practice, due to the non-linear dependency between the filtered error variance and the number of frames m, adding further frames does not always produce significant improvements. This relationship will be illustrated with real data in Chapter 5.

4.1.2 Motion Error and Noise-Free Images

In the scenario described in the following section, a noise term is added to the trajectory path. Equivalently, the motion vector referring from one frame to the next can be assumed to have additive, uniformely distributed white noise components [5]. This does, of course, not place restricitons on temporal correlations of the motion since only one frame is examined at a time.

$$\begin{aligned} \Delta x' &= \Delta x + q_x, p_{Qx}(q_x) = \frac{1}{q_{x,\max}} \sqcap_{q_{x,\max}} (q_x) \\ \Delta y' &= \Delta y + q_y, p_{Qy}(q_y) = \frac{1}{q_{y,\max}} \sqcap_{q_{y,\max}} (q_y) \end{aligned} \tag{4.9}$$

In [5] the authors set the maximum ME errors to $q_{x,\max} = q_{y,\max} = 0.5$ pel in order to reflect the half-pel resolution of their H.261 test codec. They also added a delta-pulse to the distribution to describe the stationary background of their sequences. Since the considerations here will result in a worst-case scenario description the zero-errors with perfect motion accuracy will be disregarded. Such zero errors quickly become leveraged when several motion vectors are concatenated and motion estimation errors accumulate. However, the uniformly distributed error model will still be used. Here it is extended to account for arbitrarily formed, independently moving image regions as well.
Whenever a motion-compensated block includes pixels from two differently moving regions, its single motion vector will be highly inaccurate for some of the pixels. All pixels from at least one of these regions will then have a motion vector error with a magnitude significantly higher than 0.5. Due to this observation the values for $q_{x,\max}$ and $q_{y,\max}$ will be measured separately based on ground truth data in Chapter 7.

With the established model its effects on the estimated motion trajectory can now be examined: After motion compensation has been performed the remaining motion vector consists of the error only.

$$\Delta x = 0 \Rightarrow \Delta x' = q_x, \Delta y = 0 \Rightarrow \Delta y' = q_y \tag{4.10}$$

In general, the motion at position $(x, y)^T$ shall be given by the two random functions $v_x(x, y)$ and $v_y(x, y)$. Starting with position $(x_0, y_0)^T$ in a given frame the subsequent

trajectory locations are given as follows:

$$
\begin{aligned}
x_0 &= 0 \\
x_1 &= x_0 + \Delta x_0' = x_0 + \underbrace{v_x(x_0, y_0)}_{=0} + q_{x0} = \underbrace{x_0}_{=0} + q_{x0} = q_{x0} \\
x_2 &= x_1 + \Delta x_1' = x_1 + v(x_1) + q_{x1} = x_1 \\
&\quad + \underbrace{v_x(q_{x0}, q_{y0})}_{=:p_{x1}} + q_{x1} = q_{x0} + q_{x1} + p_{x1}
\end{aligned} \tag{4.11}
$$

and similarly

$$
\begin{aligned}
y_0 &= 0 \\
y_1 &= q_{y0} \\
y_2 &= q_{y0} + q_{y1} + p_{y1}.
\end{aligned} \tag{4.12}
$$

Without loss of generality it can be assumed that the original motion vector errors $(q_{xi}, q_{yi})^T$ and the newly defined error terms $p_{xi} = v_x(q_{xi}, q_{yi}), p_{yi} = v_y(q_{xi}, q_{yi}), \forall i$ are i.i.d. random variables since they only reflect the quantization of motion vectors and mismatches. The next location of the trajectory is consequently given by

$$
\begin{aligned}
x_3 &= x_2 + \Delta x_2' = x_2 + v_x(x_2, y_2) + q_{x2} \\
&= q_{x0} + p_{x1} + q_{x1} + v_x(q_{x0} + q_{x1} + p_{x1}, q_{y0} + q_{y1} + p_{y1}) + q_{x2} \\
&= q_{x0} + q_{x1} + q_{x2} + p_{x1} + p_{x2} \\
y_3 &= q_{y0} + q_{y1} + q_{y2} + p_{y1} + p_{y2}.
\end{aligned} \tag{4.13}
$$

From these preliminary calculations a general description of the motion compensated trajectory location can be derived:

$$
x_n = \sum_{i=0}^{n-1} q_{xi} + \sum_{i=1}^{n-1} p_{xi}, \;\; y_n = \sum_{i=0}^{n-1} q_{yi} + \sum_{i=1}^{n-1} p_{yi}. \tag{4.14}
$$

It is now possible to calculate the resulting similarity between the n-th pixel along the trajectory and the original pixel with luminance $Y = Y(x_0, y_0)$. For this the filtered average over m samples is again calculated.

$$
\hat{Y} = \frac{1}{m} \sum_{i=1}^{m} Y(x_i, y_i) = \frac{1}{m} \sum_{i=1}^{m} Y \left(\sum_{j=0}^{i-1} q_{xj} + \sum_{j=1}^{i-1} p_{xj}, \sum_{j=0}^{i-1} q_{yj} + \sum_{j=1}^{i-1} p_{yj} \right) \tag{4.15}
$$

If the mean of the introduced error is again assumed to be zero, the error variance is given by the following term.

$$
\begin{aligned}
E\left[\left(\hat{Y}-Y\right)^2\right] &= E\left[\left(\frac{1}{m}\sum_{i=1}^{m} Y\left(\sum_{j=0}^{i-1} q_{xj}+\sum_{j=1}^{i-1} p_{xj}, \sum_{j=0}^{i-1} q_{yj}+\sum_{j=1}^{i-1} p_{yj}\right) - Y(x_0,y_0)\right)^2\right] \\
(4.16) \qquad &= \underbrace{E\left[\left(\frac{1}{m^2}\left(\sum_{i=1}^{m} Y\left(\sum_{j=0}^{i-1} q_{xj}+\sum_{j=1}^{i-1} p_{xj}, \sum_{j=0}^{i-1} q_{yj}+\sum_{j=1}^{i-1} p_{yj}\right)\right)^2\right]}_{=:\mathrm{I}} \\
&\quad \underbrace{-2\,E\left[\frac{1}{m}\sum_{i=1}^{m} Y\left(\sum_{j=0}^{i-1} q_{xj}+\sum_{j=1}^{i-1} p_{xj}, \sum_{j=0}^{i-1} q_{yj}+\sum_{j=1}^{i-1} p_{yj}\right)\cdot Y(x_0,y_0)\right]}_{=:\mathrm{II}} \\
&\quad + \underbrace{E\left[Y^2(x_0,y_0)\right]}_{=:\mathrm{III}}
\end{aligned}
$$

All three terms introduced in Equation 4.16 are now examined seperately and evaluated with respect to the worst-case scenario. The error variance will increase if the term I becomes maximal.

$$
\begin{aligned}
\mathrm{I} &= E\left[\left(\frac{1}{m^2}\left(\sum_{i=1}^{m} Y\left(\sum_{j=0}^{i-1} q_{xj}+\sum_{j=1}^{i-1} p_{xj}, \sum_{j=0}^{i-1} q_{yj}+\sum_{j=1}^{i-1} p_{yj}\right)\right)^2\right] \\
(4.17) \qquad &= \frac{1}{m^2} E\left[\left(\sum_{i=1}^{m} Y\left(\sum_{j=0}^{i-1} q_{xj}+\sum_{j=1}^{i-1} p_{xj}, \sum_{j=0}^{i-1} q_{yj}+\sum_{j=1}^{i-1} p_{yj}\right)\right)^2\right]
\end{aligned}
$$

In order to make the following equations more easily readable, the sum of motion vector errors will now be represented by the trajectory locations $(x_i, y_i)^T$ as defined in Equation 4.14. A lower bound for term I can then be derived.

$$
\begin{aligned}
\mathrm{I} &= \frac{1}{m^2} E\left[\left(\sum_{i=1}^{m} Y\left(x_i, y_i\right)\right)^2\right] \\
(4.18) \qquad &= \frac{1}{m^2} E\left[\sum_{i=1}^{m} Y\left(x_i, y_i\right)^2\right] + \frac{1}{m^2}\sum_{i=1}^{m} E\left[Y\left(x_i, y_i\right)\sum_{\substack{j=1 \\ j\neq i}}^{m} Y\left(x_j, y_j\right)\right] \\
&= \frac{1}{m^2} E\left[\sum_{i=1}^{m} Y(x_i, y_i)^2\right] + \frac{1}{m^2}\sum_{i=1}^{m}\sum_{\substack{j=1 \\ j\neq i}}^{m} E\left[Y\left(x_i, y_i\right) Y\left(x_j, y_j\right)\right]
\end{aligned}
$$

With the aim of making the following calculations mathematically feasible a linear Taylor approximation of the autocorrelation model is used.

$$E\left[Y(x,y)Y(x+\Delta x, y+\Delta y)\right] = (1 + |\Delta x| \cdot \log \alpha_x + |\Delta y| \cdot \log \alpha_y) \cdot \sigma_Y^2 \tag{4.19}$$

The correspondence between both models may be seen in Figure 4.1. However, the linear model is only applicable for very small values of Δx and Δy.

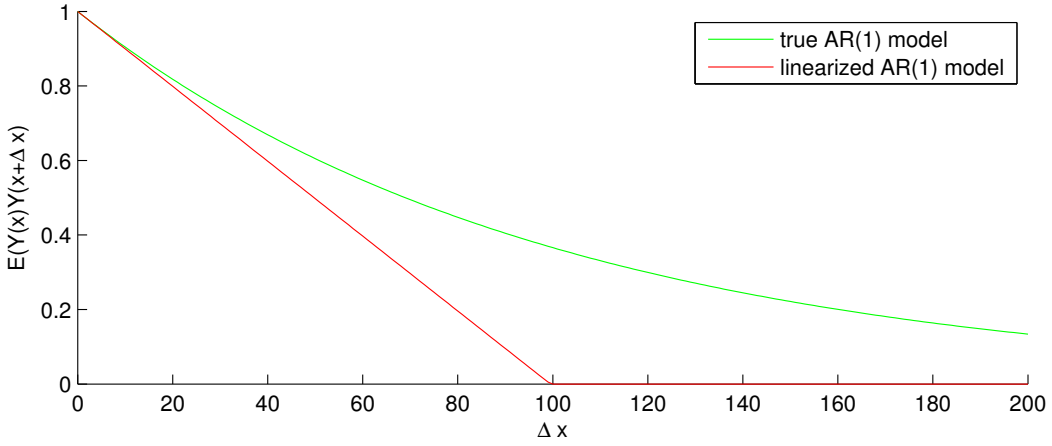


Figure 4.1: Graphical comparison between the true and the linearized $AR(1)$ correlation model. A close match between both functions can be observed for $|\Delta x| < 20$.

The cross-correlation term $E\left[Y\left(x_i, y_i\right) Y\left(x_j, y_j\right)\right]$ in Equation 4.17 compares two arbitrary motion compensated locations along the trajectory. Since both are always influenced by the accumulated noisy motion differences over several frames, it is highly likely that the linearized correlation model is no longer applicable but has gone into saturation at a value of zero. To better approximate the original exponential model, the correlation is here set to zero. In how far this simplification is justified will be shown in Chapter 7.

$$\begin{aligned} \mathrm{I} &= \frac{1}{m^2} E\left[\sum_{i=1}^{m} Y(x_i, y_i)^2\right] + \frac{1}{m^2} \sum_{i=1}^{m} \sum_{\substack{j=1 \\ j \neq i}}^{m} E\left[Y\left(x_i, y_i\right) Y\left(x_j, y_j\right)\right] \\ &= \frac{\sigma_Y^2}{m} + \frac{1}{m^2} \sum_{i=1}^{m} \sum_{\substack{j=1 \\ j \neq i}}^{m} \underbrace{E\left[Y\left(x_i, y_i\right) Y\left(x_j, y_j\right)\right]}_{\sigma_Y^2 \alpha_x^{|x_i - x_j|} \alpha_y^{|y_i - y_j|}} \\ &\approx \frac{\sigma_Y^2}{m} \end{aligned} \tag{4.20}$$

Since the term II is subtracted from the total error variance, the variance is biggest if this term becomes minimal.

$$\begin{aligned} \text{II} &= E\left[\frac{1}{m}\sum_{i=1}^{m} Y\left(\sum_{j=0}^{i-1} q_{xj} + \sum_{j=1}^{i-1} p_{xj}, \sum_{j=0}^{i-1} q_{yj} + \sum_{j=1}^{i-1} p_{yj}\right)\cdot Y(x_0, y_0)\right] \\ &= \frac{1}{m}\sum_{i=1}^{m} E\left[Y\left(\sum_{j=0}^{i-1} q_{xj} + \sum_{j=1}^{i-1} p_{xj}, \sum_{j=0}^{i-1} q_{yi} + \sum_{j=1}^{i-1} p_{yj}\right)\cdot Y(x_0, y_0)\right] \end{aligned} \tag{4.21}$$

After the application of the linearized correlation model term II is transformed into

$$\begin{aligned} \text{II} = \frac{1}{m}\sum_{i=1}^{m}\Bigg(1 &+ |\sum_{j=0}^{i-1} q_{xj} + \sum_{j=1}^{i-1} p_{xj}|\cdot\log\alpha_x \\ &+ |\sum_{j=0}^{i-1} q_{yj} + \sum_{j=1}^{i-1} p_{yj}|\cdot\log\alpha_y\Bigg)\cdot\sigma_Y^2. \end{aligned} \tag{4.22}$$

Since $\log\alpha_x < 0$ and $\log\alpha_y < 0$, the sum of motion vector errors needs to be maximized ($p_{xj} = q_{xj} = q_{x,\max}, p_{yj} = q_{yj} = q_{y,\max}, \forall j$).

$$\begin{aligned} \text{II} &= \frac{1}{m}\sum_{i=1}^{m}\left(1 + |\sum_{j=0}^{i-1} q_{xj} + \sum_{j=1}^{i-1} p_{xj}|\cdot\log\alpha_x + |\sum_{j=0}^{i-1} q_{yj} + \sum_{j=1}^{i-1} p_{yj}|\cdot\log\alpha_y\right)\cdot\sigma_Y^2 \\ &\geq \frac{1}{m}\sum_{i=1}^{m}\left(1 + (2i-1)\cdot q_{x,\max}\cdot\log\alpha_x + (2i-1)\cdot q_{y,\max}\cdot\log\alpha_y\right)\cdot\sigma_Y^2 \\ &= \left(1 + \frac{1}{m}\underbrace{\sum_{i=1}^{m}(2i-1)}_{=m^2}\cdot q_{x,\max}\cdot\log\alpha_x + \frac{1}{m}\underbrace{\sum_{i=1}^{m}(2i-1)}_{=m^2}\cdot q_{y,\max}\cdot\log\alpha_y\right)\cdot\sigma_Y^2 \\ &= \left(1 + m\cdot(q_{x_{\max}}\cdot\log\alpha_x + q_{y_{\max}}\cdot\log\alpha_y)\right)\cdot\sigma_Y^2 \end{aligned} \tag{4.23}$$

The whole filtered error variance is then given by the following equation:

$$\begin{aligned} E\left[\left(\hat{Y} - Y\right)^2\right] &= \text{I} - 2\text{II} + \text{III} \\ &\leq \underbrace{\frac{\sigma_Y^2}{m}}_{=\text{I}} - 2\underbrace{\left(1 + m\left(q_{x,\max}\cdot\log\alpha_x + q_{y,\max}\cdot\log\alpha_y\right)\right)\cdot\sigma_Y^2}_{=\text{II}} + \underbrace{\sigma_Y^2}_{=\text{III}} \end{aligned} \tag{4.24}$$

4.1.3 Motion Error and Noisy Images

The most general case with motion estimation errors and noisy images shares many properties with the previously discussed one with noise-free images. The derivation of the trajectory locations $(x_n, y_n)^T$ can again be taken from Equation 4.14. In addition, the luminance values are now superimposed with coding noise η. This will initially be assumed to have the properties of white noise with zero mean and a variance of σ_η^2.

$$Y'(x,y) = Y(x,y) + \eta \tag{4.25}$$

The noise term is interpreted as being white both in spatial and temporal dimensions. Supporting evidence for these assumptions may be found in [9]. Since only a single trajectory path with one pixel per frame is investigated the noise term η can without loss of generality be seen as a constant value per frame.

$$Y'(x_n,y_n) = Y(x_n,y_n) + \eta_n = Y\left(\sum_{j=0}^{n-1} q_{xj} + \sum_{j=1}^{n-1} p_{xj}, \sum_{j=0}^{n-1} q_{yj} + \sum_{j=1}^{n-1} p_{yj}\right) + \eta_n \tag{4.26}$$

A filtered pixel now consists of a sum of luminance values taken from possibly inaccurate pixel locations and additive noise.

$$\hat{Y} = \frac{1}{m}\sum_{i=1}^{m} Y'(x_m,y_m) = \frac{1}{m}\left(\sum_{i=1}^{m} Y(x_m,y_m) + \sum_{i=1}^{m}\eta_i\right) \tag{4.27}$$

If the noise has zero mean, the error variance after filtering can be calculated in the following manner.

$$\begin{aligned} E\left[\left(\hat{Y}-Y\right)^2\right] &= E\left[\left(\frac{1}{m}\sum_{i=1}^{m} Y(x_i,y_i) + \frac{1}{m}\sum_{i=1}^{m}\eta_i - Y(x_0,y_0)\right)^2\right] \\ &= \underbrace{E\left[\frac{1}{m^2}\left(\sum_{i=1}^{m} Y(x_i,y_i)\right)^2\right]}_{=\mathrm{I}} \underbrace{-2\,E\left[\frac{1}{m}\sum_{i=1}^{m} Y(x_i,y_i)\cdot Y(x_0,y_0)\right]}_{=\mathrm{II}} \\ &\quad + E\left[\frac{1}{m^2}\sum_{i=1}^{m}\eta_i \cdot \sum_{i=1}^{m}\eta_i\right] + 2E\left[\frac{1}{m^2}\cdot\sum_{i=1}^{m} Y(x_i,y_i)\sum_{i=1}^{m}\eta_i\right] \\ &\quad -2E\left[\frac{1}{m^2}\sum_{i=1}^{m}\eta_i\cdot Y(x_0,y_0)\right] + \underbrace{E\left[Y(x_0,y_0)Y(x_0,y_0)\right]}_{=\mathrm{III}} \end{aligned} \tag{4.28}$$

Terms I, II and III have already been calculated during the description of the simpler case of noise-free images. The noise terms $\eta_i \ \forall i$ can safely be assumed to be uncorrelated [9].

$$\begin{aligned} E\left[\left(\hat{Y}-Y\right)^2\right] &= E\left[\frac{1}{m}\left(\sum_{i=1}^{m} Y\left(x_i,y_i\right)\right)^2\right] - 2E\left[\frac{1}{m}\sum_{i=1}^{m} Y(x_i,y_i)\cdot Y(x_0,y_0)\right] \\ &\quad + E\left[Y(x_0,y_0)Y(x_0,y_0)\right] + E\left[\frac{1}{m^2}\sum_{i=1}^{m}\eta_i\cdot\sum_{i=1}^{m}\eta_i\right] \\ &= \frac{\sigma_Y^2}{m} - 2(1+m\left(q_{x,\max}\cdot\log\alpha_x + q_{y,\max}\cdot\log\alpha_y\right))\cdot\sigma_Y^2 \\ &\quad + \sigma_Y^2 + E\left[\frac{1}{m^2}\sum_{i=1}^{m}\eta_i\cdot\sum_{i=1}^{m}\eta_i\right] \\ &= \frac{\sigma_Y^2}{m} - 2(1+m\left(q_{x,\max}\cdot\log\alpha_x + q_{y,\max}\cdot\log\alpha_y\right))\cdot\sigma_Y^2 + \sigma_Y^2 + \frac{\sigma_\eta^2}{m} \end{aligned} \tag{4.29}$$

4.2 Optimal Filter Length and Possible Quality Improvements

If perfect motion knowledge is available and only the images are distorted by i.i.d. noise the optimum filter length will of course always be infinite. Based on the calculations done in Section 4.1.2 concepts are now derived that can be tested in real-world scenarios. These will be used in Chapter 7 in order to validate the theoretical considerations. This is done for simplification, since the results presented here can easily be extended to the case of both noisy motion vectors and noisy images. As given in Equation 4.24 the variance after filtering is reduced to

$$E\left[\left(\hat{Y}-Y\right)^2\right] = \frac{\sigma_Y^2}{m} - 2(1+m(q_{x,\max}\cdot\log\alpha_x + q_{y,\max}\cdot\log\alpha_y))\cdot\sigma_Y^2 + \sigma_Y^2. \tag{4.30}$$

The optimum filter length can now be calculated by finding the minimum of the equation above. Theoretically, there should exist a global optimum for the filter length, which intuitively is induced by the trade-off between filtering gain and motion error accumulation. However, for noise-free images the optimal trajectory length should be one. Since $E\left[\left(\hat{Y}-Y\right)^2\right]$ is a convex function for $m \in \mathbb{R}_0^+$, finding the

optimum corresponds to setting the first derivative of $E\left[\left(\hat{Y}-Y\right)^2\right]$ to zero

$$\frac{\partial E\left[\left(\hat{Y}-Y\right)^2\right]}{\partial m} = -\frac{\sigma_Y^2}{m^2} - 2(q_{x,\max}\log\alpha_x + q_{y,\max}\log\alpha_y)\cdot\sigma_Y^2 = 0. \tag{4.31}$$

Solving the equation for m yields

$$\begin{aligned} \frac{\sigma_Y^2}{m_{\text{opt}}^2} &= -2(q_{x,\max}\log\alpha_x + q_{y,\max}\log\alpha_y)\cdot\sigma_Y^2 \\ m_{\text{opt}}^2 &= -\frac{1}{2(q_{\text{x,max}}\log\alpha_x + q_{y,\max}\log\alpha_y)} \\ m_{\text{opt}} &= \sqrt{-\frac{1}{2(q_{x,\max}\log\alpha_x + q_{y,\max}\log\alpha_y)}} \end{aligned} \tag{4.32}$$

For typical values $q_{x,\max}, q_{y,\max} \geq 3$, and $\alpha_x, \alpha_y \approx 0.98$ this yields a trajectory length of $m_{\text{opt}} \leq 1.65$ simply corresponding to the original pixel with no additional motion-compensated samples. The filter is thus disabled for this scenario. Of course, the actual predicted filter length strongly depends on the values for α_x and α_y. For some parameter combinations a filter length greater than 2 might thus be predicted. The ideal case is, however, always quickly approached, when $q_{x,\max}$ and $q_{y,\max}$ increase.

Since real-world sequences also contain added noise due to compression artifacts, the above calculation also needs to be conducted for the case of noisy images described again by the following equation:

$$E\left[\left(\hat{Y}-Y\right)^2\right] = \frac{\sigma_Y^2}{m} - 2(1+m(q_{x,\max}\cdot\log\alpha_x, q_{y,\max}\cdot\log\alpha_y))\cdot\sigma_Y^2 + \sigma_Y^2 + \frac{\sigma_\eta^2}{m} \tag{4.33}$$

A typical curve for $E[(\hat{Y}-Y)^2]$ may be found in Figure 4.2, where $q_{x,\max} = q_{y,\max} = 7, \alpha_x = \alpha_x = 0.995$, $\frac{\sigma\eta^2}{\sigma_Y^2} = 0.7$. These values correspond to the properties of the *RaceHorses* sequence, see Appendix C. Setting the first derivative to zero again

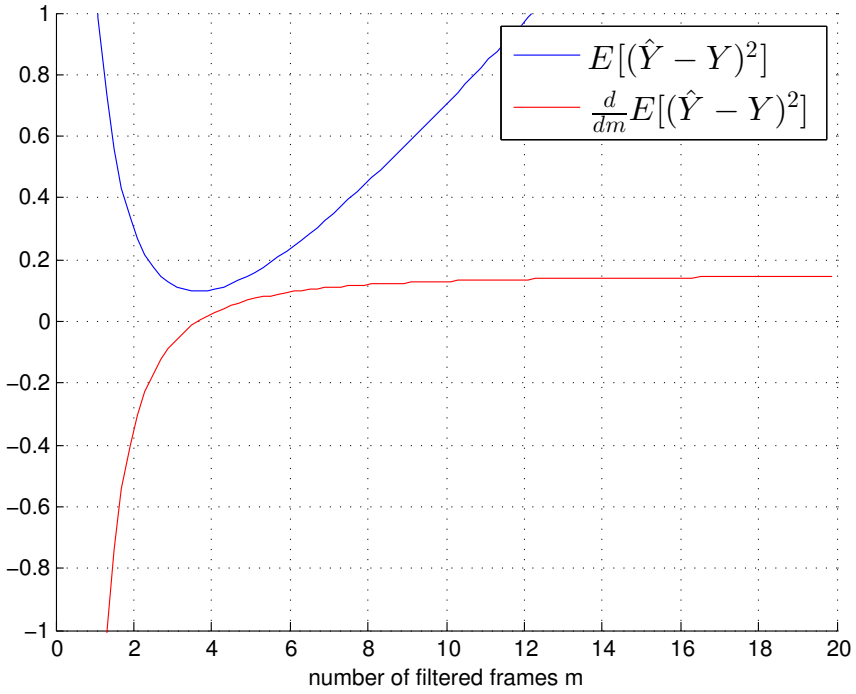


Figure 4.2: The achievable distortion variance $E[(\hat{Y}-Y)^2]$ (blue) after filtering is convex on $\mathbb{R}_0^+$. The root of its derivative (red) corresponds to the global minimum of $E[(\hat{Y}-Y)^2]$. The input values for this simulation were taken from the *RaceHorses* sequence.

yields an equation that can be used to determine the optimal filter length m_{opt}.

$$\begin{aligned}
\frac{\partial E\left[\left(\hat{Y}-Y\right)^2\right]}{\partial m} &= -\frac{\sigma_Y^2}{m_{\text{opt}}^2} - 2(q_{\text{x,max}}\log\alpha_x + q_{\text{y,max}}\log\alpha_y)\cdot\sigma_Y^2 - \frac{\sigma_\eta^2}{m_{\text{opt}}^2} = 0 \\
0 &= -\sigma_Y^2 - 2(q_{x,\max}\log\alpha_x + q_{y,\max}\log\alpha_y)\cdot m_{\text{opt}}^2\cdot\sigma_Y^2 - \sigma_\eta^2 \qquad (4.34)\\
0 &= -1 - 2(q_{x,\max}\log\alpha_x + q_{y,\max}\log\alpha_y)\cdot m_{\text{opt}}^2 - \frac{\sigma_\eta^2}{\sigma_Y^2} \\
m_{\text{opt}} &= \sqrt{-\frac{1+\frac{\sigma_\eta^2}{\sigma_Y^2}}{2(q_{x,\max}\log\alpha_x + q_{y,\max}\log\alpha_y)}}
\end{aligned}$$

The roots of Equation 4.34 only correspond to a minimum of the filtered variance, if the second derivative is greater than zero. Which is always true for any given variances σ_Y^2, σ_η^2:

$$\frac{d^2E\left[\left(\hat{Y}-Y\right)^2\right]}{dm^2} = \frac{\sigma_Y^2}{m^3} + \frac{\sigma_\eta^2}{m^3} > 0 \qquad (4.35)$$

Based on real-world sequences and ground truth motion data the applicability of the formulae derived above will be demonstrated in Chapter 7 as well.

4.3 Trajectory Structure and GOP-Structure

The original idea of a motion trajectory in the temporal direction of a video sequence needs to be adapted to the actual motion information available at both encoder and decoder. The following three cases can easily be distinguished and will be examined seperately in later chapters.

- Ideally, pixel-wise sub-pel motion information is available which allows both encoder and decoder to construct perfect one-dimensional pixel trajectories for the entire image content. This scenario which was examined in Chapter 3 can actually have practical relevance, for instance, if a global motion model accurately describes all pixels within a frame. Another realization could be an elastic model [39] which is theoretically also able to describe the motion of every pixel within a frame.
- A more realistic scenario occurs when a video codec such as H.264/AVC in its baseline configuration is used. With an IPPP coding structure, motion

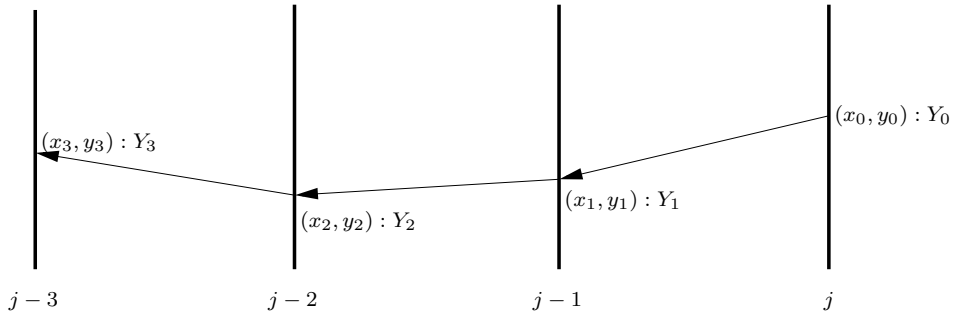


Figure 4.3: If an IPPP coding structure is used, a motion trajectory can be build that comprises all frames of the sequence, unless intra-coded blocks occur.

information is available for almost all pixels of a P-frame. These vectors will minimally describe a block of 4×4 neighboring pixels and they will point to the previously encoded/decoded frame. Thus, the trajectory will contain one pixel per frame. A visualization of such a trajectory may be found in Figure 4.3. No motion information is, however, available if intra-coded blocks occur. In this case the filter cannot be applied. Theoretically, the constructed trajectories can have infinite length. A trajectory will of course be interrupted by intra blocks and also by motion vectors pointing to a point outside of the frame boundaries. This case will be examined in Section 5.1 which was first published in [12].

- The most general case has again no practical value but offers interesting theoretical aspects. If motion information for every pixel in a frame relative to every other frame of the video sequence is available, then no errors are introduced through the concatenation of these vectors since every trajectory point can directly be accessed. This means that a multitude of pixel copies is available for filtering even if the respective image point is, for instance, hidden behind other objects in some parts of the sequence. A practical realization of this concept is offered by the B-frame structure both employed in the main/extended profiles of H.264/AVC and in HEVC. Another realization are the multihypothesis frames found in [17]. The trajectory structure derived from hierarchical B-frames may be found in Figure 4.4. The trajectory then corresponds to a tree of pixel locations that is split at every B-frame. Again, individual parts of the trajectory may be interrupted if a P-predicted block or an I-predicted block are referenced. The advantage of the tree structure is the possibility to track an image point over multiple frames even if it is not visible in intermediate images. A description of this case may also be found in Section 5.1.

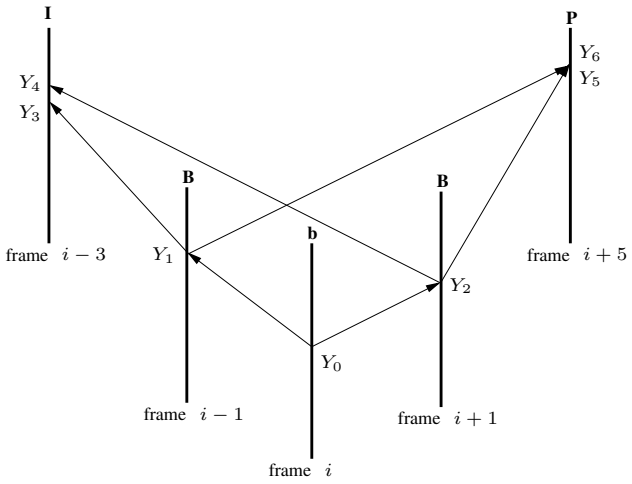


Figure 4.4: If an IBBB coding structure is used, the motion trajectory is split at every B-frame thus forming not a one-dimensional trajectory path but a tree of possible trajectory locations.

4.4 Chapter Summary

In this chapter formulae have been derived that predict the behavior of the TTF under certain circumstances. Based on the ideal case of noise-free videos with perfect motion knowledge, the model was step by step expanded to include both noisy videos and motion estimation errors. It is assumed that a video sequence can sufficiently be characterized by the maximum motion estimation error components $q_{\mathrm{x,max}}$, $q_{\mathrm{y,max}}$ and the spatial image correlation coefficients α_x and α_y since those are the only independent variables in the analytical description of the TTF. The resulting equations will later be used to predict the optimal filter length in Chapter 7.

Chapter 5

Practical Realizations of Temporal Trajectory Filters

A good idea is about ten percent and implementation and hard work, and luck is 90 percent.[sic] - Guy Kawasaki

5.1 Temporal Trajectory Filtering

The original idea for the Temporal Trajectory Filter was published in [12]. The first implementation already made use of two filtering parameters and realized variable filter lengths per processed pixel. Additional modifications and extensions of the filter will now be described.

The following Sections 5.1.1 to 5.1.5 were first published in [9] ©IEEE 2012.

5.1.1 Introduction

In this section, the trajectory filter is discussed in more detail and its properties are more closely examined. The main advantage of the new approach over other existing methods is its ability to compute a trajectory over a large number of frames with the use of only three filter parameters. These enable the filter to perform optimal filtering per pixel to reduce block artifacts, which sets it apart from all block-based approaches. In effect, the filter can convert simple P- or B-frames into multi hypothesis frames without transmitting additional motion information within the bit stream. Details on how to derive pixel trajectories directly from an H.264/AVC encoded bit stream are given in Subsection 5.1.2. The subsequent filter design is

described in Subsection 5.1.3 together with an algorithm to efficiently compute optimal filter parameters. The encoder and decoder design is also detailed thereafter. Subsection 5.1.4 describes the experimental evaluation of the filter and compares its performance with both the H.264/AVC baseline and extended profile. In addition, the filter is compared with the Wiener-based QALF and both are investigated concerning their computational complexity. Finally, Subsection 5.1.5 summarizes the Section and provides a conclusion.

5.1.2 Formation of Temporal Pixel Trajectories

In order to perform pixel-wise trajectory filtering on a video sequence, it becomes necessary to identify the individual motion of every pixel at the decoder. Moreover, the implementation of the filtering concept described in Subsection 5.1.4 will use the filter in the decoding loop at both encoder and decoder. In this case an efficient way to compute trajectories from data conveyed in the bit stream is needed. To this end, the concatenation of coded H.264/AVC block-based motion vectors (MV) is now considered, which avoids the transmission of unnecessary side information to the decoder.

Concatenation of Motion Vectors

The following description uses hierarchical B-frames as the underlying coding structure. Nevertheless, all equations are equally applicable to P-frames, in which case the second motion vector per block is simply omitted.
$(dx_{i,0}, dy_{i,0})^T$ and $(dx_{i,1}, dy_{i,1})^T$ shall denote the motion vectors for a certain block in a bi-directionally predicted frame i for reference lists 0 and 1 respectively. In this context the translational block motion is assumed to be identical to the motion of every pixel within that block, which yields the two motion vector fields $(dx_{i,0}(x,y), dy_{i,0}(x,y))^T$ and $(dx_{i,1}(x,y), dy_{i,1}(x,y))^T$. Frames referenced by these vectors shall be indicated by $\mathrm{ref}_{i,0}(x,y)$ and $\mathrm{ref}_{i,1}(x,y)$. Starting with an arbitrary pixel $(x_0, y_0)^T$ in frame i of a video sequence the two reference locations in frames $\mathrm{ref}_{i,0}(x,y)$ and $\mathrm{ref}_{i,1}(x,y)$ are then given by

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} dx_{i,0}(\lfloor x_0 \rfloor, \lfloor y_0 \rfloor) \\ dy_{i,0}(\lfloor x_0 \rfloor, \lfloor y_0 \rfloor) \end{pmatrix},$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} dx_{i,1}(\lfloor x_0 \rfloor, \lfloor y_0 \rfloor) \\ dy_{i,1}(\lfloor x_0 \rfloor, \lfloor y_0 \rfloor) \end{pmatrix}. \tag{5.1}$$

Through the concatenation of several motion vectors a potential trajectory for every single pixel can now be formed. The next two locations derived from $(x_1, y_1)^T$ are

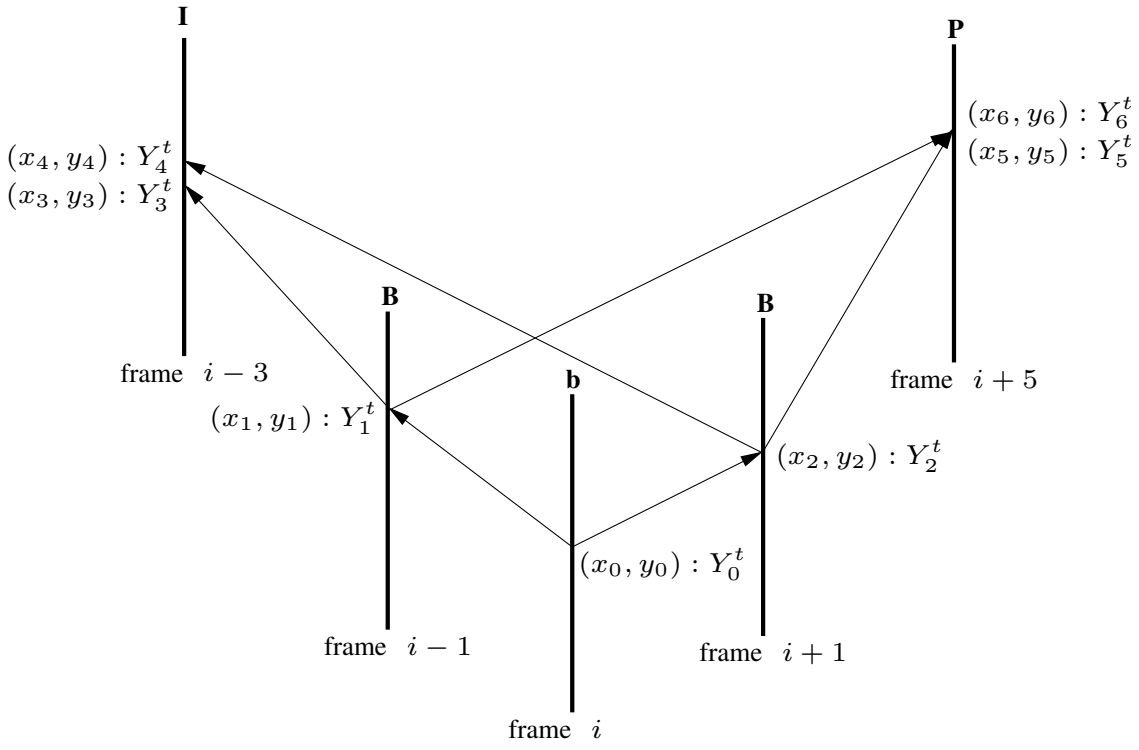


Figure 5.1: Starting with a b-frame the trajectory for each pixel is computed through the concatenation of motion vectors yielding here a total of seven luminance samples along the trajectory. Note that only some hierarchical B-frames out of a GOP of size 8 are depicted for simplification.

for example

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} dx_{\mathrm{rf},0}(\lfloor x_1 \rfloor, \lfloor y_1 \rfloor) \\ dy_{\mathrm{rf},0}(\lfloor x_1 \rfloor, \lfloor y_1 \rfloor) \end{pmatrix},$$

$$\begin{pmatrix} x_6 \\ y_6 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} dx_{\mathrm{rb},1}(\lfloor x_1 \rfloor, \lfloor y_1 \rfloor) \\ dy_{\mathrm{rb},1}(\lfloor x_1 \rfloor, \lfloor y_1 \rfloor) \end{pmatrix} \tag{5.2}$$

$$\text{with} \qquad \mathrm{rf} = \mathrm{ref}_{i,0}(x_1, y_1),\ \mathrm{rb} = \mathrm{ref}_{i,1}(x_1, y_1).$$

This relationship is illustrated by Figure 5.1. Contrary to the original idea of a trajectory, which associates every pixel with a single motion path through the sequence, the trajectory now branches into two individual trajectories at each B-frame. The resulting trajectory structure for an exemplary B-frame of the lowest hierarchy order is shown in Figure 5.1. Every one of these pixel locations yields a different luminance sample $Y_0, ..., Y_{N-1}$ which can potentially be used to compute a mean value

Y_{opt}. The originally reconstructed pixel Y_0 can then be replaced by Y_{opt} to achieve noise reduction. However, not necessarily do all predicted image locations actually describe the true motion of a pixel. As in the case of frame $i-3$ in Figure 5.1 only one of several pixels within an image can actually be part of the trajectory. Therefore, it becomes necessary to distinguish between motion vectors that do correctly describe a pixel's true motion and those that do not. Criteria that enable the decoder to perform this decision will be discussed in Subsection 5.1.2. Nevertheless, the hierarchical B-frame structure offers an opportunity that is not present in the simple case of frame-wise concatenation of motion vectors. Since an individual location along the trjaectory of a pixel may be reached via different paths, it is now possible to track a single image point over several frames even if said pixel is not visible in all frames.

Distinguishing Between True and False Trajectories

To enable both encoder and decoder to select motion vectors that correctly describe a pixel's trajectory three criteria have been developed that together control the filtering process.

Absolute error along the trajectory

One indicator for a falsely predicted trajectory is a sudden change between consecutive luminance samples Y_{i+1} and Y_i. Such a change might for instance be caused by moving foreground objects concealing part of the background. Another reason could be a lighting change during the video sequence. In this case the trajectory might still be correctly predicted, but temporal filtering would introduce further errors. Using the deviation measure $\Delta Y_i = |Y_{i+1} - Y_i|$ a trajectory is subsequently only continued if

$$\Delta Y_i < T_Y. \tag{5.3}$$

Temporal motion consistency

Another property of a pixel trajectory, that can be used as a reliability test, is temporal motion consistency. It is assumed that the individual translational motion of a pixel changes only slightly from frame to frame. In order to be able to compare motion vectors from different frames, these are first scaled according to the reference

frames they point to:

$$\begin{aligned}(dx_{i,0}(x,y))' &= \frac{dx_{i,0}(x,y)}{\mathrm{ref}_{i,0}(x,y)-i} \\ (dy_{i,0}(x,y))' &= \frac{dy_{i,0}(x,y)}{\mathrm{ref}_{i,0}(x,y)-i}\end{aligned} \tag{5.4}$$

The trajectory for a given pixel is only continued, if the euclidean distance between the scaled version of the motion vector $(dx_i, dy_i)^T$ pointing to the current location of the trajectory and the scaled versions of the vectors $(dx_{r0,0}, dy_{r0,0})^T$, $(dx_{r0,1}, dy_{r0,1})^T$ pointing to the reference frames for the current location is smaller than a given threshold

$$\sqrt{((dx_{r0,0})' - (dx_i)')^2 + ((dy_{r0,0})' - (dy_i)')^2} < T_{\mathrm{TC}}. \tag{5.5}$$

Where the apostrophe marks that scaled versions of the original motion vectors are used.

Spatial motion consistency

Another indicator for a motion vector, that does not describe the true motion of a pixel, can be found by comparing it with its neighboring vectors. Even at object boundaries, spatially adjacent motion vectors on the 4×4-block level should ideally be similar. If one vector differs significantly from its neighbors, it is assumed that a false motion vector has been used due to *RD*-optimization. Again, for better comparability scaled versions of all motion vectors are used. The block-vote metric for reference list 0 $BV_{i,0}(x, y)$ now gives the number of neighboring motion vectors for the 4×4-blocks surrounding the trajectory's current location (x, y), whose x- or y-components differ from the current motion vector. Figure 5.2 illustrates the functionality of the block-vote metric in combination with the scaling of motion vectors. The filtering is subsequently only continued along the trajectory when the block-vote metric for the current pixel satisfies

$$BV_{i,0}(x, y) \leq 4 - T_{\mathrm{SC}} \tag{5.6}$$

for a given threshold T_{SC}.

Adaptive Filtering of Pixels

The combination of all three thresholds introduced above ensures that every pixel, even inside the same 4×4-block, is filtered using a different trajectory path and a different number of luminance samples. Hence, block-artifacts introduced by quantized

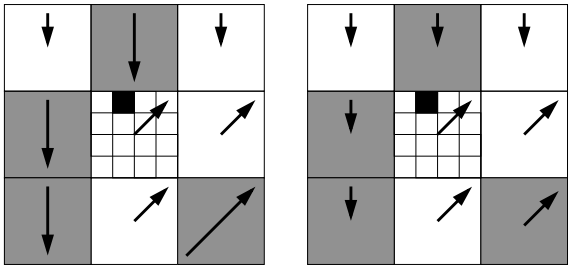


Figure 5.2: The image on the *left* shows the motion vectors for the 4×4-blocks surrounding the trajectories current location marked in black. All blocks with a gray background use a reference frame with a greater temporal distance relative to the current frame than the reference frames for the other blocks. The scaled motion vectors are shown on the *right*. The resulting number of neighboring blocks with differing motion (i.e. the block-vote metric) is 5.

DCT coefficients resulting in homogeneous blocks can effectively be compensated. Visual examples for this property will be provided in Subsection 5.1.4.

5.1.3 Derivation of Filter Parameters

In the previous paragraphs filter parameters have been described that effectively control the formation of individual pixel trajectories thereby contributing to different filter characteristics. Throughout the remainder of this section the filtering step consists of calculating the average luminance value along the trajectory as given in Equation 3.3. This operation is chosen for its simplicity so as not to increase the encoder complexity. In theory an optimal weight could be determined for every pixel along the trajectory, which would result in an increased computational overhead and therefore remains a topic for a later section. A derivation method for an optimal set of the three previously mentioned parameters will now be outlined.

MSE minimization

Since the primary target of both post and in-loop filters is the improvement of the subjective picture quality, the filter parameters are optimized with respect to a maximum SNR-gain for the current frame. The resulting effect on future frames, which would also influence the bit rate, is currently not considered. All three filter thresholds are tested iteratively at the encoder. For each of these combinations the resulting MSE is computed. Effectively, it is sufficient to follow the path of a single pixel using the associated motion vectors. During this process the MSEs for all parameter combinations can be calculated simultaneously. The combination of

filter parameters that yields the minimal MSE then needs to be transmitted to the decoder so that it can perform identical filtering operations.

Signaling of Filter Parameters

The two thresholds T_Y and T_{TC} could theoretically take on arbitrary values. For practical purposes, however, and to reduce the time needed for parameter optimization each threshold is restricted to values between 0 and 7. As a majority vote is used for the spatial motion consistency the appropriate range of values for the threshold T_{SC} is $0 \leq T_{\mathrm{SC}} < 4$. As will be shown in Subsection 5.1.4, there is virtually no temporal correlation concerning the optimum filter parameters. The thresholds do, however, change their characteristics according to the QP of the frame to be filtered. Moreover, each parameter appears to be evenly distributed over its range of possible values. Consequently, all thresholds can be signalled in the bit stream with eight additional bits per frame. An adaptivity per frame, in contrast to a fixed set of thresholds, is needed since frames with little motion can be filtered with much longer trajectories. Frames with rapidly moving foreground objects require, however, much stronger restrictions on motion and color change. The setting $T_Y = 0$ is used to disable the filter altogether as the restriction of luminance values in this case inhibits all changes on the reconstructed frame and the two other parameters can be omitted.

Encoder and Decoder Design

In general, the new in-loop filter could be applied both before and after the H.264/AVC Deblocking Filter. Especially in combination with the threshold T_{SC}, however, the filter performs better when utilized before the Deblocking Filter. The resulting modified encoder is shown in Figure 5.3. For the H.264/AVC baseline profile eight past reconstructed, filtered frames are stored at both encoder and decoder in a simple queue. To allow for trajectories over at least eight past and/or future frames the simple queue model needs to be extended when hierarchical B-frames are used. In this case four past I- and P-frames are stored also.

5.1.4 Experimental Evaluation

The proposed new in-loop filter has been implemented in C and integrated into the reference software *JM 16* [44] for the well-established codec H.264/AVC [55]. As described in Subsection 5.1.1 both the baseline profile and the extended profile with hierarchical B-frames have been tested. The precise encoder settings for both

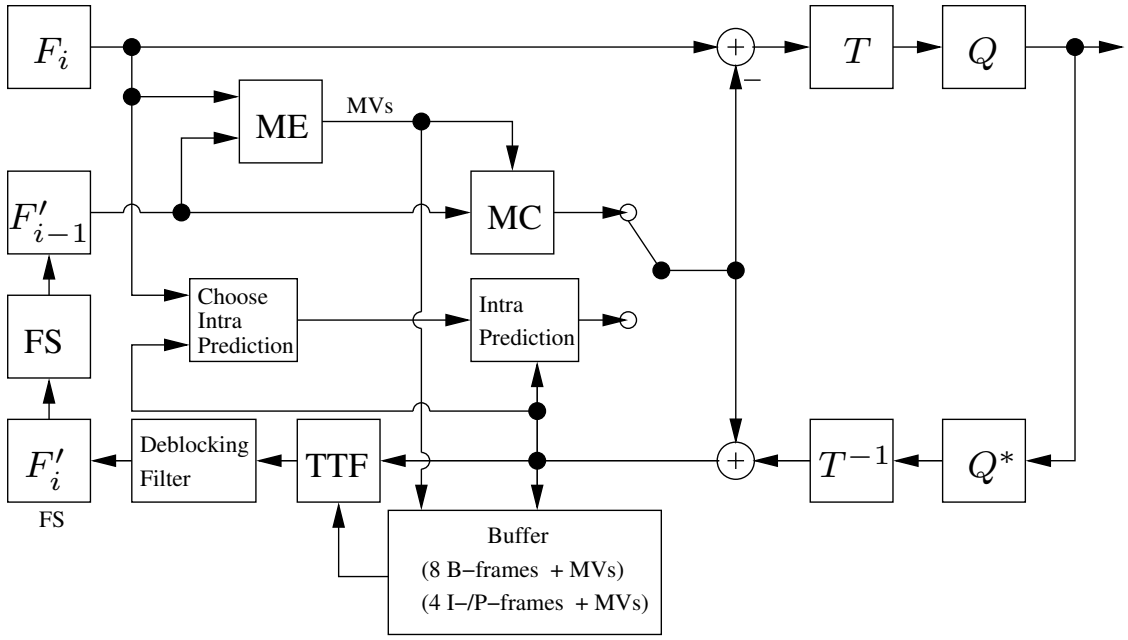


Figure 5.3: The proposed filter is integrated into the local decoder loop at the encoder before the Deblocking Filter. F'_i and F'_{i-1} are the current reconstructed picture and its predecessor stored in the frame store (FS).

cases are given in Tables 5.1 and 5.2. The in-loop filter has been applied to the set of test sequences used in the HEVC standardization activities. These range from WQVGA sequences to cropped WQXGA sequences. Among these are also sequences of various temporal resolutions ranging from 25 Hz to 60 Hz. All details concerning the sequences may be found in Table 5.3 summarized as classes B to E.

Table 5.1: Settings used for the H.264/AVC Baseline Profile.

H.264/AVC Reference Software	JM 16
Profile	baseline
Picture order / GOP size	IPPP / -
Entropy Coding	UVLC
RD Optimization	high complexity
Motion Estimation	EPZS
QP I-slice	$\in \{22, 27, 32, 37, 42\}$
QP P-slice	QP I-slice +1

Table 5.2: Settings used for the H.264/AVC Extended Profile.

H.264/AVC Reference Software	JM 16
Profile	extended
Picture order / GOP size	hierarchical B / 8
Entropy Coding	UVLC
RD Optimization	high complexity
Motion Estimation	EPZS
QP I-slice	$\in \{22, 27, 32, 37, 42\}$
QP P-slice	QP I-slice +1
QP B-slice	QP I-slice +3

Objective Measurements

For the baseline and the extended profile all sequences have been encoded using H.264/AVC with and without the additional in-loop filter. The resulting bit rates and PSNR values were compared using the Bjøntegaard metric [4]. The respective BD-PSNR values and BD-rates for QP 22 to 37 are given in the second and third column of Table 5.4 for the baseline and Table 5.5 for the extended profile. The overall average BD-rate for the baseline profile is -3.61% with a maximum average gain of 9.70% for a single sequence. When hierarchical B-frames are used a bit rate reduction of up to 6.82% is achieved with an average BD-rate of -2.06%. It appears that the filter works equally well on both P- and B-frames. In addition,

Table 5.3: All test sequences used in the experiments with their respective resolutions. The number of frames corresponds in each case to a 10 s video sequence.

	Sequence	Resolution	Frames
Class D	*BasketballPass*	416x240	500
	BlowingBubbles	416x240	500
	BQSquare	416x240	600
	RaceHorses	416x240	300
Class C	*BasketballDrill*	832x480	500
	BQMall	832x480	600
	PartyScene	832x480	600
	RaceHorses	832x480	300
Class E	*Vidyo1*	1280x720	600
	Vidyo3	1280x720	600
	Vidyo4	1280x720	600
Class B	*BasketballDrive*	1920x1080	500
	BQTerrace	1920x1080	600
	Cactus	1920x1080	500
	Kimono1	1920x1080	240
	ParkScene	1920x1080	240
Additional Sequences	*Allstars*	704x576	250
	BBC-pan-13	720x576	110
	Desert	720x400	240
	Entertainment	720x576	250
	Waterfall	704x480	300

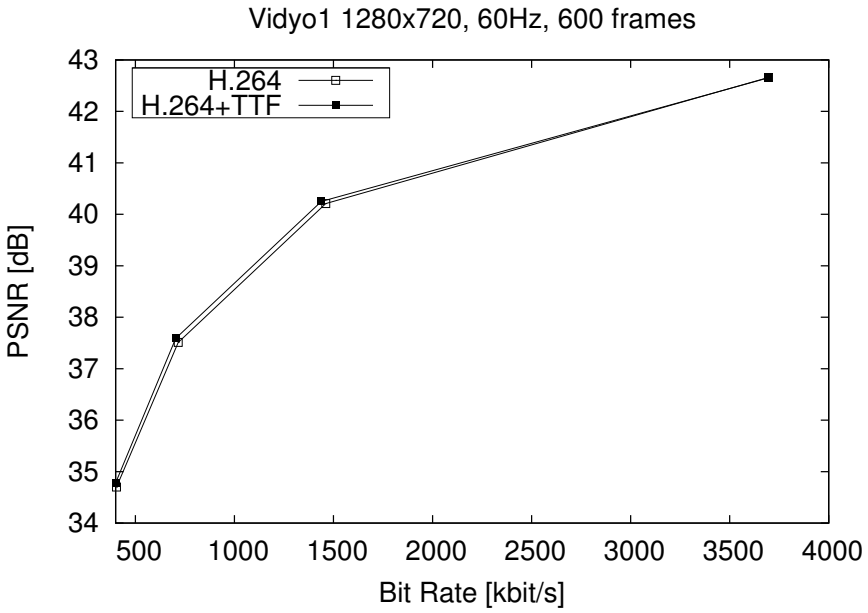
a gain is achieved for all tested sequences which justifies the overhead introduced by the transmission of the filter parameters. In particular, good results have been achieved for both high- and low-resolution sequences. Even for sequences with large moving foreground objects, such as both *RaceHorses* sequences, gains are achieved. For both the baseline and the extended profile the actual frame rate of a sequence does not seem to have a significant impact on the performance of the filter. For example both for *BQSquare* (60 Hz) and for *ParkScene* (24 Hz) bit rate reductions of more than 2% are observed. Interestingly, the gain for the extended profile is higher than the one observed for the baseline profile for both *BasketballDrill* and *Cactus*. This is due to quickly moving foreground objects, which result in occluded regions that can not be filtered if an IPPP coding structure is used. For hierarchical B-frames, however, both past and future frames are available for filtering, in which case the adaptive TTF can successfully be applied to such regions, too. The RD-curves in Figures 5.4(a) and 5.4(b) illustrate the results reported in Table 5.4 in more detail. The general behavior of the proposed filter is shown by Figure 5.4(a), where significant quality improvement for the *Video1* sequence is only present for QPs 27 and 32. At very high bit rates the motion trajectory is more accurately determined, but generally fewer artifacts impair the visual quality of the decoded sequence. Consequently, the gain achievable at QP 22 is smaller. At low bit rates, on the other hand, the reference frames are of poorer quality, which also reduces the reliability of the transmitted motion vectors. In this case, too, the effectiveness of the TTF is limited. Nevertheless widely distributed gain can be achieved for some sequences, such as *BQSquare*. The RD-curve for this particular sequence is given in Figure 5.4(b).

Additional Test Sequences

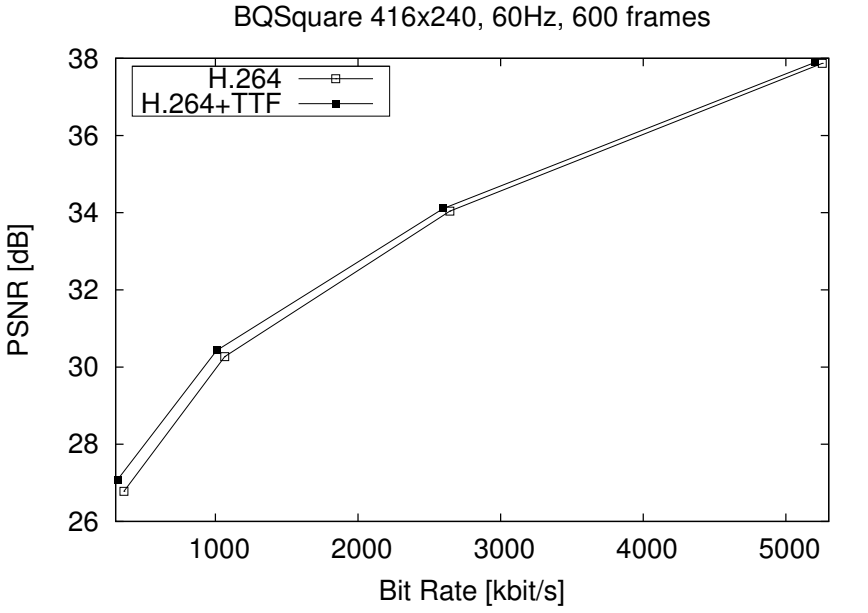
To illustrate the full potential of the TTF approach, the filter has also been tested on a number of sequences not included in the HEVC data set. These are also listed in Table 5.3. Among these are sequences with significant camera motion such as track (*Allstars*, *BBC-pan-13*) or zoom (*Waterfall*). The resulting BD-measures for the baseline and the extended profile are shown in Tables 5.4 and 5.5, too. Especially the average BD-rate of -10.88% for the baseline profile shows that the filter can improve the performance of H.264/AVC significantly in presence of camera motion.

Influence of the Deblocking Filter

As can be seen from Figure 5.3 the TTF is here used before the Deblocking Filter is applied. Experiments investigating the optimal order of both filters showed that the behavior of the codec is only minimally changed when the Deblocking Filter



(a) *Vidyo1*



(b) *BQSquare*

Figure 5.4: Exemplary RD-curves for the baseline profile. For the *BQSquare* sequence the TTF provides objective quality improvement for all depicted bit rates. For the *Vidyo1* sequence the gain is restricted to the medium bit rate range.

is instead applied before the TTF. Apparently, the noise characteristics after the application of the Deblocking Filter are still suitable for the TTF. This may in part be due to the fact, that the Deblocking Filter operates in the spatial domain only while the TTF is applied in the temporal domain. Omitting the Deblocking Filter altogether, however, drastically impairs the quality of the compressed sequence.

Comparison with QALF

In order to objectively compare the performance of the TTF with other state-of-the-art in-loop filtering concepts, all experiments have also been conducted with the QALF as described in [6]. To this end the QALF implementation provided by *KTA 2.7* in combination with *JM 16* was used. QALF was here selected since it was considered to be one of the most promising new in-loop filters at the time these experiments were conducted. The BD-PSNR and BD-rates values for all test sequences using the baseline profile may be found in the third and fourth column of Table 5.4. The respective values for the extended profile are given in Table 5.5.

Sequence	QP 22 to 37						QP 27 to 42					
	TTF		QALF		TTF+QALF		TTF		QALF		TTF+QALF	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballPass	0.03	−0.70	0.31	−6.06	0.31	−6.04	0.02	−0.37	0.26	−5.45	0.26	−5.35
BlowingBubbles	0.12	−2.90	0.31	−7.34	0.34	−7.85	0.12	−3.24	0.29	−7.54	0.31	−8.03
BQSquare	0.31	−7.41	0.56	−13.02	0.75	−16.92	0.48	−12.96	0.70	−18.26	1.05	−26.73
RaceHorses	0.11	−2.28	0.25	−4.92	0.27	−5.35	0.08	−1.71	0.22	−4.58	0.22	−4.72
avg. for class D	**0.14**	**−3.32**	**0.36**	**−7.84**	**0.42**	**−9.04**	**0.17**	**−4.57**	**0.36**	**−8.96**	**0.46**	**−11.21**
BasketballDrill	0.00	0.00	0.56	−13.14	0.56	−13.19	0.00	0.01	0.60	−14.01	0.54	−12.65
BQMall	0.16	−3.69	0.45	−10.17	0.49	−10.89	0.21	−4.27	0.52	−10.49	0.56	−10.95
PartyScene	0.23	−5.08	0.43	−9.55	0.55	−11.95	0.27	−6.79	0.46	−11.45	0.59	−14.37
RaceHorses	0.07	−1.74	0.31	−7.27	0.33	−7.56	0.09	−2.15	0.32	−7.28	0.32	−7.15
avg. for class C	**0.12**	**−2.63**	**0.44**	**−10.03**	**0.48**	**−10.90**	**0.14**	**−3.30**	**0.47**	**−10.81**	**0.50**	**−11.28**
Vidyo1	0.09	−2.66	0.68	−17.64	0.67	−17.31	0.13	−2.67	0.77	−14.73	0.76	−14.65
Vidyo3	0.19	−5.66	0.62	−17.60	0.64	−17.99	0.29	−6.23	0.89	−18.55	0.91	−18.84
Vidyo4	0.11	−3.31	0.59	−16.45	0.58	−16.29	0.14	−2.99	0.77	−15.70	0.76	−15.46
avg. for class E	**0.13**	**−3.88**	**0.63**	**−17.23**	**0.63**	**−17.20**	**0.19**	**−3.96**	**0.81**	**−16.33**	**0.81**	**−16.32**
BasketballDrive	0.05	−1.53	0.35	−11.78	0.35	−11.78	0.05	−1.30	0.46	−11.59	0.46	−11.60
BQTerrace	0.22	−9.70	0.49	−20.62	0.53	−22.05	0.44	−14.99	0.87	−28.23	0.94	−30.53
Cactus	0.00	−0.07	0.23	−8.84	0.23	−8.84	0.01	−0.15	0.32	−8.87	0.32	−8.87
Kimono1	0.14	−3.79	0.57	−14.45	0.58	−14.71	0.16	−3.46	0.60	−12.30	0.61	−12.45
ParkScene	0.31	−7.99	0.34	−8.97	0.40	−10.26	0.31	−7.22	0.38	−9.04	0.43	−9.98
avg. for class B	**0.14**	**−4.62**	**0.40**	**−12.93**	**0.42**	**−13.53**	**0.19**	**−5.42**	**0.53**	**−14.01**	**0.55**	**−14.69**
Allstars	0.11	−3.08	0.46	−12.25	0.48	−12.54	0.16	−4.16	0.50	−11.73	0.51	−12.12
BBC-pan-13	0.37	−8.63	0.75	−16.08	0.78	−16.83	0.44	−8.01	0.96	−14.05	0.92	−13.99
Desert	0.30	−7.10	0.99	−21.14	1.00	−21.45	0.39	−7.45	1.10	−19.50	1.12	−19.82
Entertainment	0.22	−4.95	0.69	−14.93	0.75	−16.32	0.31	−7.08	0.74	−16.04	0.83	−17.96
Waterfall	1.02	−28.37	1.06	−30.12	1.37	−36.76	1.27	−27.05	0.68	−24.00	1.06	−27.30
avg. add. seq.	**0.41**	**−10.88**	**0.80**	**−19.61**	**0.90**	**−21.76**	**0.50**	**−10.88**	**0.75**	**−17.47**	**0.86**	**−18.81**

Table 5.4: BD-rate and average PSNR gain for test sequences for the H.264/AVC baseline profile with TTF, QALF, and both filters in combination. Columns 1 to 6 show the results for QP 22 to 37. Columns 7 to 12 show the results for QP 27 to 42.

Sequence	QP 22 to 37						QP 27 to 42					
	TTF		QALF		TTF+QALF		TTF		QALF		TTF+QALF	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballPass	0.02	−0.41	0.18	−3.55	0.17	−3.39	0.01	−0.18	0.12	−2.68	0.12	−2.58
BlowingBubbles	0.04	−0.99	0.31	−7.75	0.31	−7.67	0.04	−1.28	0.26	−7.17	0.27	−7.34
BQSquare	0.08	−2.26	0.62	−16.96	0.63	−17.39	0.10	−2.93	0.61	−15.81%	0.66	−17.07
RaceHorses	0.01	−0.21	0.14	−2.81	0.14	−2.79	0.00	−0.04	0.11	−2.47	0.10	−2.21
avg. for class D	**0.04**	**−0.97**	**0.31**	**−7.77**	**0.31**	**−7.81**	**0.04**	**−1.11**	**0.28**	**−7.03**	**0.29**	**−7.30**
BasketballDrill	0.01	−0.14	0.35	−8.28	0.35	−8.19	0.01	−0.23	0.34	−8.01	0.33	−7.79
BQMall	0.02	−0.35	0.28	−6.09	0.28	−6.03	0.01	−0.26	0.28	−5.23	0.28	−5.21
PartyScene	0.05	−1.28	0.47	−11.32	0.47	−11.25	0.08	−2.10	0.40	−10.64	0.40	−10.57
RaceHorses	0.01	−0.29	0.18	−4.23	0.18	−4.19	0.02	−0.38	0.15	−3.60	0.16	−3.62
avg. for class C	**0.02**	**−0.51**	**0.32**	**−7.48**	**0.32**	**−7.41**	**0.03**	**−0.74**	**0.30**	**−6.87**	**0.29**	**−6.80**
Vidyo1	0.14	−3.89	0.42	−11.19	0.41	−11.10	0.13	−2.85	0.49	−10.78	0.48	−10.50
Vidyo3	0.12	−3.05	0.55	−14.11	0.58	−14.96	0.13	−2.74	0.69	−14.13	0.73	−14.77
Vidyo4	0.10	−2.76	0.39	−10.86	0.39	−10.78	0.07	−1.59	0.44	−9.97	0.43	−9.66
avg. for class E	**0.12**	**−3.23**	**0.45**	**−12.05**	**0.55**	**−12.28**	**0.11**	**−2.39**	**0.54**	**−11.63**	**0.55**	**−11.64**
BasketballDrive	0.03	−0.86	0.18	−5.75	0.17	−5.46	0.03	−0.59	0.22	−5.18	0.21	−4.90
BQTerrace	0.14	−6.82	0.45	−21.16	0.45	−21.47	0.22	−6.83	0.75	−22.39	0.77	−22.49
Cactus	0.03	−0.96	0.15	−5.36	0.16	−5.60	0.02	−0.70	0.17	−4.72	0.17	−4.83
Kimono1	0.09	−2.43	0.40	−9.04	0.43	−10.94	0.08	−1.99	0.40	−9.04	0.40	−9.05
ParkScene	0.14	−3.80	0.23	−6.17	0.23	−6.17	0.13	−3.46	0.24	−6.28	0.25	−6.48
avg. for class B	**0.09**	**−2.98**	**0.29**	**−9.88**	**0.29**	**−9.93**	**0.10**	**−2.71**	**0.36**	**−9.52**	**0.36**	**−9.55**
Allstars	0.10	−2.6	0.39	−10.22	0.42	−10.81	0.06	−1.56	0.38	−8.29	0.38	−8.54
BBC-pan-13	0.12	−2.26	0.54	−10.97	0.60	−11.62	0.09	−1.74	0.58	−9.86	0.50	−8.54
Desert	0.10	−2.14	0.61	−13.09	0.58	−11.58	0.08	−1.30	0.58	−10.20	0.50	−8.92
Entertainment	0.05	−1.19	0.52	−11.17	0.50	−10.89	0.05	−1.12	0.52	−10.61	0.50	−10.26
Waterfall	0.48	−14.36	0.83	−23.38	0.83	−23.49	0.47	−12.81	0.80	−20.67	0.81	−20.78
avg. add. seq.	**0.17**	**−4.51**	**0.58**	**−13.77**	**0.58**	**−13.68**	**0.15**	**−3.70**	**0.57**	**−11.95**	**0.58**	**−13.68**

Table 5.5: BD-rate and average PSNR gain for test sequences for the H.264/AVC extended profile with TTF, QALF and both filters in combination. Columns 1 to 6 show the results for QP 22 to 37. Colums 7 to 12 show the results for QP 27 to 42.

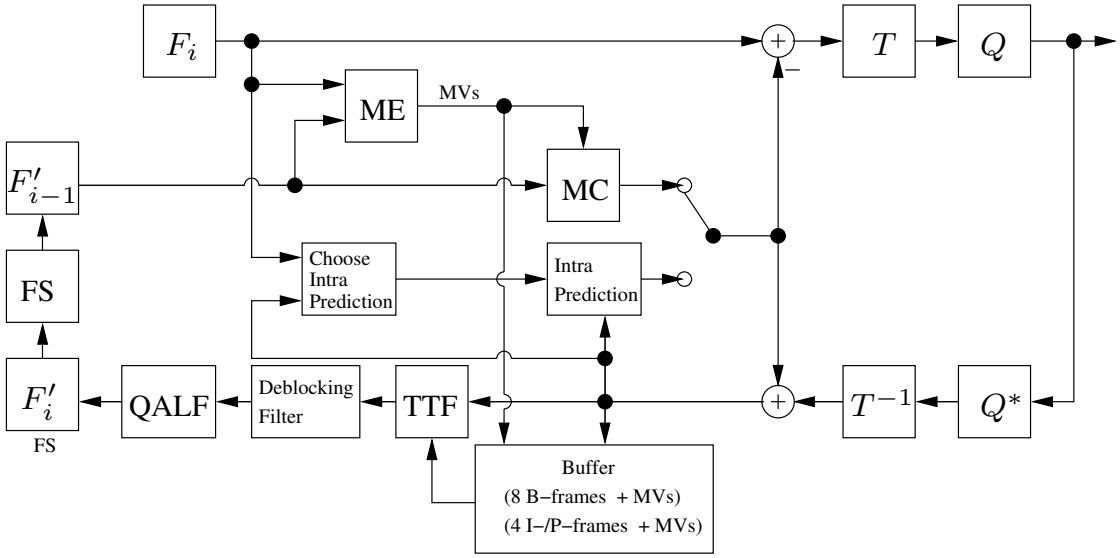


Figure 5.5: When combining TTF and QALF, the TTF is used in front of the Deblocking Filter while the QALF is applied afterwards.

As these results show, the QALF outperforms the TTF for all sequences, although similar gains are achieved for *ParkScene* and *Waterfall*. This is only to be expected since the QALF is an already heavily RD-optimized filter having undergone several major improvements over the last years. Nevertheless, the TTF still manages to produce BD-rates that are at least in the same order of magnitude as those provided by QALF.

In addition to the experiments described above, both filters were also tested in combination with the aim of proving that they are not necessarily mutually exclusive. The resulting modified H.264/AVC encoder with both QALF and TTF is shown in Figure 5.5. The BD-rates for the combination of both filters when using the baseline profile are shown in the fourth column of Table 5.4. Constructive interference can be observed for all tested sequences except for class E, where slight losses occur for *Vidyo1* and *Vidyo4*. In addition, the combined gain of both filters is significantly higher than that provided by QALF for *BQSquare*, *PartyScene*, *ParkScene*, *BQTerrace* and *Waterfall*. As can be seen, the combination of both filters is most effective when the TTF alone already provides a gain of more than 5%. This finding also indicates that QALF and TTF do not necessarily reduce the impact of the same noise sources at the encoder. Even though the noise observed in the temporal domain is essentially white, it may originate from several independent sources like motion estimation and quantization of DCT coefficients. For each of these sources one of the two filters may be better suited. A combination of temporal and spatial filtering

proves to be even more effective in this context.
Table 5.5 shows the equivalent average BD-rates per class when hierarchical B-frames are used. In all cases, except for the *Waterfall* sequence, the combination of both filters provides lower gain than QALF alone. For the *Desert* sequence, for example, this can be explained by the heat noise present in the original sequence. Although the TTF improves the quality of an entire frame, it removes the heat noise. The QALF then tries to reconstruct the previously filtered pixels which results in a very large number of bits needed for the quadtree. The interaction between both filters, especially in combination with hierarchical B-frames, needs to be further investigated in the future.

Performance at low bit rates

Since block-artifacts are most noticeable at low bit rates all experiments were also conducted for QP 42 resulting in already strongly reduced visual quality for all sequences. As the right half of Tables 5.4 and 5.5 show the TTF performs better still at poor quality. The QALF's performance is also improved, although the difference between both filters is decreased at least for the baseline profile.

Complexity Analysis

Since the computational complexity of both QALF and TTF is mostly located at the encoder the average encoding time ratios compared to H.264/AVC baseline and extended profile are shown in Tables 5.6 and 5.7. The decoding time ratios are given also. All time-measurements were conducted on identical conventional 3 GHz-CPUs using a single-thread implementation of the respective method. As expected both filters increase the encoder complexity considerably when hierarchical B-frames are used. Even then, the TTF is faster than the QALF despite the fact that the TTF encoder still only uses conventional floating-point arithmetics. In the baseline case, the encoding times for the TTF vary depending on the QP. Especially at low bitrates the modified encoder is faster than the original H.264/AVC encoder. This is due to the highly improved quality of the reference pictures used for motion estimation. Consequently, the EPZS [46] algorithm stops earlier because of its early termination criterion, resulting in a speed-up of the motion estimation, which accounts for most of the encoder run time. The EPZS algorithm in combination with the TTF only needs about 75% of the original motion estimation time for both tested profiles. As a result the average encoding time over all tested QPs lies at about 84% of the reference encoder runtime for the baseline and 198% for the extended profile. When hierarchical B-frames are used, the number of possible trajectories increases exponentially. This slows down the encoder considerably. Concerning the decoder,

the complexity for QALF is usually smaller except for the additional sequences where both filters require an additional runtime of about 50% of the reference decoding time. One option to further simplify the decoding process for the TTF will be the usage of bit-shift operations instead of the previously described averaging. In addition, the computation of the optimal filter parameters can easily be parallelized at the encoder.

Sequence	TTF		QALF	
	T_{ENC}	T_{DEC}	T_{ENC}	T_{DEC}
average for class D	83%	149%	184%	112%
average for class C	82%	165%	170%	137%
average for class E	92%	185%	158%	134%
average for class B	79%	157%	166%	143%
average for additional sequences	89%	149%	159%	164%

Table 5.6: Per class average encoding and decoding time ratios compared to H.264/AVC baseline profile (QP 22 to 42).

Sequence	TTF		QALF	
	T_{ENC}	T_{DEC}	T_{ENC}	T_{DEC}
average for class D	188%	148%	482%	113%
average for class C	188%	174%	490%	121%
average for class E	250%	179%	466%	136%
average for class B	195%	152%	465%	128%
average additional sequences	198%	149%	478%	146%

Table 5.7: Per class average encoding and decoding time ratios compared to H.264/AVC extended profile (QP 22 to 42).

Subjective Examples

Apart from improving the objective quality of the decoded video the additional in-loop filter also improves the subjective quality. As outlined in Subsection 3.2 the main purpose of the TTF is to reduce the impact of block-artifacts on the performance of H.264/AVC. These are expected to occur especially at the boundaries of moving objects in a video sequence and at high QP-values. Figure 5.6 shows frame 500 of the decoded *BQSquare* sequence both with and without the application of the TTF filter (bit rate reduced by 19%, PSNR increased by 0.28 dB for the depicted QP). Especially in the upper left corner the distortion is visibly reduced by the TTF. The same can be observed in the examples shown in Figure 5.7 which displays

(a) H.264/AVC at 358 kbit/s, 26.8 dB (b) H.264/AVC + TTF at 309 kbit/s, 27.1 dB (c) Original frame

Figure 5.6: An enlarged part of frame 200 from the decoded *BQSquare* sequence for QP 37.

(a) H.264/AVC at 2683 kbits/s, 30.1 dB (b) H.264/AVC + TTF at 2355 kbit/s, 30.3 dB (c) Original frame

Figure 5.7: An enlarged part of frame 500 from the decoded *BQTerrace* sequence for QP 37.

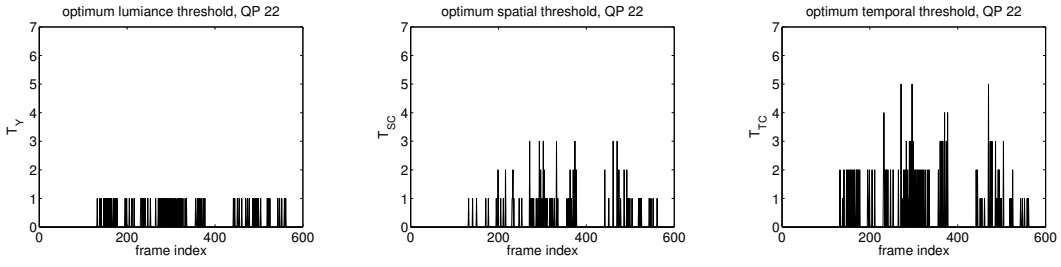


Figure 5.8: Optimal thresholds T_Y, T_{SC} and T_{TC} for the *BQMall* sequence for QP 22.

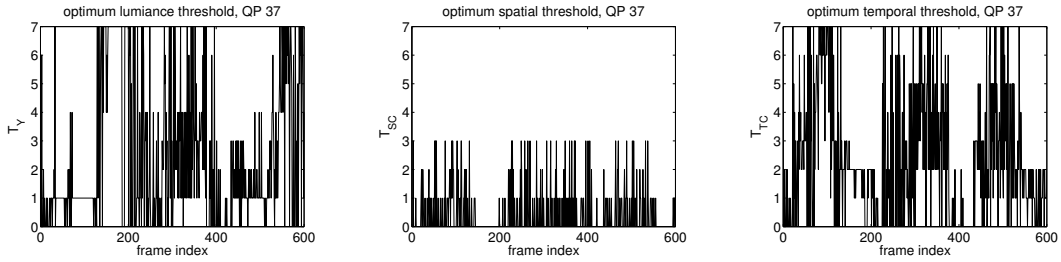


Figure 5.9: Optimal thresholds T_Y, T_{SC} and T_{TC} for the *BQMall* sequence for QP 37.

frame 500 from the *BQTerrace* sequence. Again the noise introduced by the codec is reduced without the introduction of new artifacts. In addition, the bit rate is significantly reduced (bit rate reduced by 12%, PSNR increased by 0.25 dB for the depicted QP).

Analysis of the Filter Parameters

In Subsection 5.1.3 it has been implied that the optimal filter parameters are essentially random variables. To support this assumption Figure 5.8 shows the optimum thresholds transmitted for the *BQMall* sequence for the baseline of H.264/AVC using a QP of 22. The respective thresholds for QP 37 are depicted in Figure 5.9. Essentially, all three thresholds are indeed random. Their variation, however, strongly depends on the encoded sequence and the QP used. Since stronger artifacts are expected at high QPs, the filter here is intended to produce much larger deviations from the originally reconstructed picture. This fact is supported by a significantly higher average value for the luminance threshold T_Y for QP 37. For QP 22 the filter usually only allows minor changes to the reconstruced frame as can be seen in Figure 5.8. In addition, motion vectors are also expected to be less accurate at low bit rates. Accordingly, the average value for the temporal consistency threshold is also increased in such cases. In the experiments described above, a fixed set of thresholds was applied on frame basis. It is, of course, also possible to adapt the thresholds spatially. This variant of the TTF will be investigated in Section 5.3.

Analysis of the Trajectories

Figures 5.10 and 5.11 show histograms of the trajectory lengths over all frames of the *BQMall* sequence both for the baseline and the extended profile. In these two cases all trajectories with a length greater than seven are for simplification summarized into one bin of the histogram. Both for high and low QPs these figures

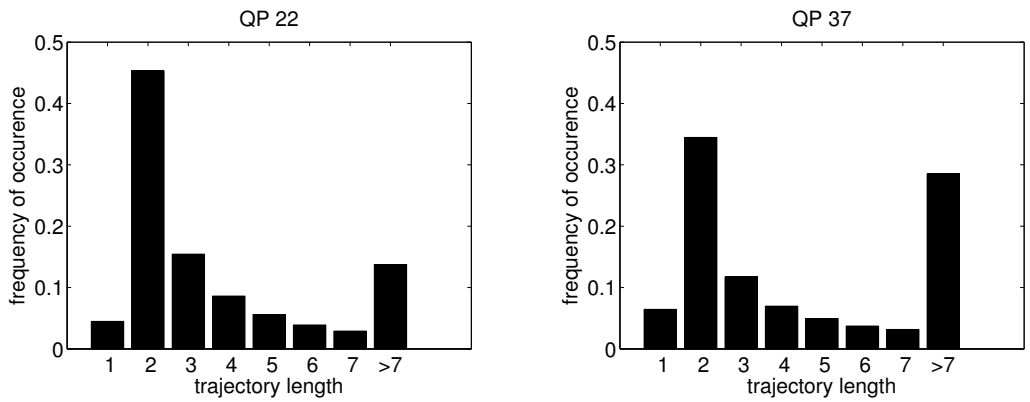


Figure 5.10: Average trajectory lengths for the *BQMall* sequence when using the baseline profile.

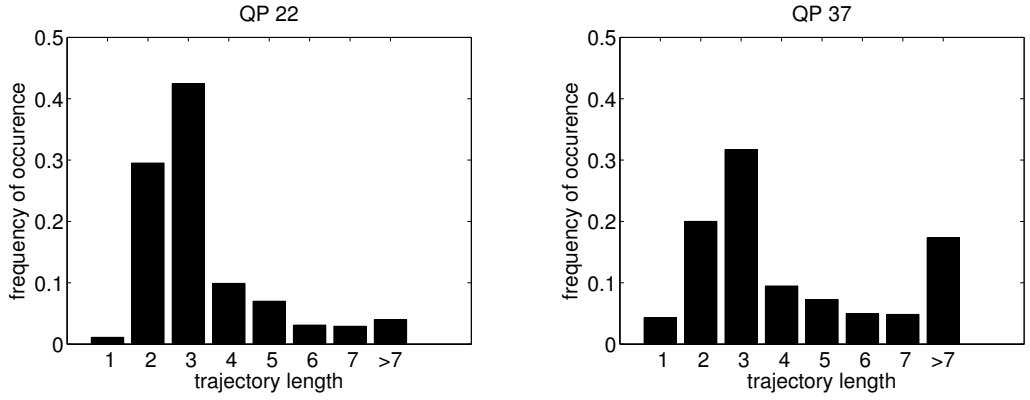


Figure 5.11: Average trajectory lengths for the *BQMall* sequence when using the extended profile.

show similar distributions of the trajectory lengths. In the baseline case the filter essentially extends simple P-frames to B-frames by filtering around 40% of the pixels over two frames. For the remaining 50% trajectories of even greater length are used. That B-frames themselves can still be improved, can be seen from Figure 5.11. Around 70% of the pixels for the extended profile are filtered using 3 or more frames. Consequentially, the major advantage of the filter is its ability to turn every frame of a video sequence into a multihypothesis frame without having to transmit additional motion and mode information on the macro-block level. This is one of the advantages of the algorithm compared to other multihypothesis approaches such as the ones described in [54], [18], and [17]. There, each pixel, or more precisely block of pixels, included in the filtering process for calculating the prediction signal needs to be referenced by a *hypotheses* or motion vector with an individual reference

index and derived motion information. This overhead can completely be omitted in case of the TTF which selects the appropriate pixels for filtering automatically by using the given three thresholds per frame.

5.1.5 Summary

The proposed new adaptive in-loop filter has been shown to out-perform both objectively and subjectively the H.264/AVC reference codec over a large variety of sequences with an average BD-rate of -3.61% for the baseline and -2.06% for the extended profile, when only the HEVC test data set is used. Average gains of up to 28.37% are reported on sequences with significant global camera motion. HEVC itself was not used as a reference here, since the standardization project was still in a preliminary stage at the time the experiments were conducted.
At least for the baseline profile, the combination of TTF and the EPZS algorithm for motion estimation also decreases the encoder run time by up to 21%. All three thresholds described in this section appear to be good indicators for correctly predicted trajectories, since the picture quality is always improved. Nevertheless, the MSE-based optimization of these thresholds does not necessarily constitute an optimal solution as only the picture quality and not the required bit rate is considered. In addition, it is currently not possible to compare the predicted trajectories with the true motion of individual pixels, since the required computational overhead would be significant. A later conducted experiment based on the true pixel motion has already been described in Chapter 3. A combination with the Wiener-based QALF has also been tested indicating that both filters in combination can potentially provide an even higher gain.

5.2 Weighted Temporal Long Trajectory Filtering

The original implementation of the TTF described above used a simple averaging filter in an attempt to remove temporal noise from a decoded video sequence. The filter did not take into account the actual nature of this noise, which is introduced by the compression algorithm. By exploiting this knowledge, significantly better results can be obtained

5.2.1 Introduction

In Chapter 4 it was implicitly assumed, that all frames of a compressed video sequence are subject to the same noise source with unvarying characteristics. However, as will be shown in the following section, the TTF's performance can be significantly improved, if the actual noise depending on the QP value per frame is taken into account. This leads to a new concept of a weighted temporal filter. Additionally, trajectories have previously always been interrupted when a misfitting sample along the trajectory occured. Here, it will be shown that the TTF performs much better, when the noisy samples are omitted and the formation of the trajectory is continued nonetheless.

5.2.2 Lagrangian Minimization and its Applications in Video Coding

In video compression, trade-off problems between bit rate and quality are often solved using Lagrangian minimization. A description of the Lagrangian methods employed in H.264/AVC may, for instance, be found in [55]. According to [22] "Lagrange multipilers are variables with the help of which one constructs a Lagrange function for investigating problems on conditional extrema. The use of Lagrange multipliers and a Lagrange function makes it possible to obtain in a uniform way necessary optimality conditions in problems on conditional extrema." In general, the extremum of a function $f(x_1, ..., x_n)$ subject to m constraints $g_i(x_1, ..., x_n) = 0, i = 1..m, m < n$ can be found by formulating the Lagrangian function

$$F(x_1, ..., x_n, \lambda) = f(x_1, ..., x_n) + \sum_{i=1}^{m} \lambda_i (b_i - g_i(x_1, ..., x_n)). \tag{5.7}$$

The partial derivatives of F with respect to both $x_i, \forall i$ and λ are set to zero in order to determine the minimum of F. The location of the minimum $(x_1^*, ..., x_n^*)$ together with the optimal cost variables $\lambda_1, ..., \lambda_m$ can then be found by solving the resulting equation system consisting of n equations in F and m equations in g_i. In the context of video compression one frequent problem is the distribution of the available bit rate R over several parameters that need to be transmitted. In the applications discussed later in this thesis, the quantity to be minimized is the reconstruction error variance constraint to a fixed bit rate.

5.2.3 Theoretical Basis

For any given pixel $(x_0, y_0)^T$ in frame j it is assumed that its locations $(x_i, y_i)^T$, $1 \leq i < N$, in $N-1$ previous frames are also known. If $Y_n(x, y)$ denotes the luminance component of frame n at location $(x, y)^T$ the distorted versions of the original sample $Y_j(x_0, y_0)$ in any of the $N-1$ previous frames given by

$$Y_{j-i}(x_i, y_i) = Y_j(x_0, y_0) + n_i, 1 \leq i < N. \tag{5.8}$$

Even though the motion of the pixel is perfectly known, a noise term n_i with variance σ_i^2 is introduced due to the reduced quality of the encoded sequence. As described in [12], it can be assumed that all n_i are uncorrelated. These error terms are to a large extent due to motion estimation errors on block level and thus change from frame to frame. A filtered version of the original luma component can then be computed by calculating a weighted mean

$$\begin{aligned} Y_j^*(x_0, y_0) &= \tfrac{1}{N} \textstyle\sum_{i=0}^{N-1} \beta_i \cdot \hat{Y}_{j-i}(x_i, y_i) \\ &= \tfrac{1}{N} \textstyle\sum_{i=0}^{N-1} \beta_i \cdot Y_j(x_0, y_0) + \tfrac{1}{N} \textstyle\sum_{i=0}^{N-1} \beta_i \cdot n_i \\ &= \tfrac{1}{N} Y_j(x_0, y_0) \textstyle\sum_{i=0}^{N-1} \beta_i + \tfrac{1}{N} \textstyle\sum_{i=0}^{N-1} \beta_i \cdot n_i. \end{aligned} \tag{5.9}$$

Where β_i are the individual weights per frame with $\sum_{i=0}^{N-1} \beta_i = N$ to make the filter unbiased. This leads to the definition of a new noise term $\tilde{n}_j$ for the filtered pixel

$$Y_j^*(x_0, y_0) = Y_j(x_0, y_0) + \underbrace{\frac{1}{N} \sum_{i=0}^{N-1} \beta_i \cdot n_i}_{\tilde{n}_j}. \tag{5.10}$$

The variance of the filtered noise $\tilde{n}_j$ is subsequently given by

$$\begin{aligned} \sigma_{\tilde{n}}^2 &= E\left[\tilde{n}_j \tilde{n}_j\right] \\ &= E\left[\frac{1}{N^2} \sum_{l=0}^{N-1} \beta_l n_l \cdot \sum_{k=0}^{N-1} \beta_k n_k\right] = \frac{1}{N^2} \sum_{m=0}^{N-1} \beta_m^2 \sigma_m^2. \end{aligned} \tag{5.11}$$

As the filter is to minimize $\sigma_{\tilde{n}}^2$ constraint to $\sum_{i=0}^{N-1} \beta_i = N$, the minimum may be found by Lagrangian minimization

$$\begin{gathered} \tfrac{\partial}{\partial \beta_i} \left[\textstyle\sum_{m=0}^{N-1} \beta_m^2 \sigma_m^2 - \lambda \left(N - \textstyle\sum_{k=0}^{N-1} \beta_k\right)\right] = 0 \\ 2\beta_i \sigma_i^2 + \lambda = 0 \Rightarrow \beta_i = -\tfrac{\lambda}{2\sigma_i^2} \\ \textstyle\sum_{k=0}^{N-1} \beta_k = N \Leftrightarrow \sum_{k=0}^{N-1} -\tfrac{\lambda}{2\sigma_k^2} = N \end{gathered} \tag{5.12}$$

$$\begin{gathered} \lambda = \frac{N}{\sum_{k=0}^{N-1} -\frac{1}{2\sigma_k}} \\ \beta_i = -\tfrac{\lambda}{2} \tfrac{1}{\sigma_i^2} = \frac{N/\sigma_i^2}{\sum_{k=0}^{N-1} \frac{1}{\sigma_k^2}}. \end{gathered} \tag{5.13}$$

In theory, the reconstruction error variance σ_k^2 for every pixel along the trajectory would be required to calculate the optimal filter weight β_i. According to Wiegand and Girod [53] the distortion variance in a reconstructed frame is given by

$$D_{\mathrm{REC}} = \frac{Q_{\mathrm{step}}^2}{3} \text{ with zero mean.} \tag{5.14}$$

Where Q_{step} is the quantizer step size selected by the quantization parameter QP. Both in H.264/AVC and in HEVC, Q_{step} is roughly

$$Q_{\mathrm{step}} = 0.625 \cdot 2^{\frac{\mathrm{QP}}{6}}. \tag{5.15}$$

Subsequently, the optimal filter weight for frame i according to its QP may be calculated

$$\begin{aligned} \sigma_i^2 &= D_{\mathrm{REC},i} = \frac{1}{3}\left(0.625 \cdot 2^{\frac{\mathrm{QP}_i}{6}}\right)^2 \\ \beta_i(\mathrm{QP}_i) &= 3 \cdot \left(0.625 \cdot 2^{\frac{\mathrm{QP}_i}{6}}\right)^{-2}. \end{aligned} \tag{5.16}$$

In the case of the H.264/AVC baseline profile this yields identical weights for every frame. When varying QPs are used as in the HEVC low-delay setting, the optimal weights can be calculated at the decoder requiring no additional side information.

5.2.4 Filter Design

In the low-delay high efficiency setting of HEVC with an IBBB coding structure, every B-predicted block can have up to two motion vectors pointing to one of the last four encoded pictures. It is assumed that the motion vector for a given block also describes the individual motion of every pixel within the block. The components of the two resulting motion vector fields for frame i shall be denoted by $(dx_{i,0}, dy_{i,0})^T$ and $(dx_{i,1}, dy_{i,1})^T$. Starting again with pixel $(x_0, y_0)^T$ in frame i, two possible locations of the pixel in the referenced frames are therefore given by

$$\begin{aligned} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} &= \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} dx_{i,0}(\lfloor x_0 \rfloor, \lfloor y_0 \rfloor) \\ dy_{i,0}(\lfloor x_0 \rfloor, \lfloor y_0 \rfloor) \end{pmatrix}, \\ \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} &= \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} dx_{i,1}(\lfloor x_0 \rfloor, \lfloor y_0 \rfloor) \\ dy_{i,1}(\lfloor x_0 \rfloor, \lfloor y_0 \rfloor) \end{pmatrix}. \end{aligned} \tag{5.17}$$

This is identical to the trajectory description in Equation 5.1 which described the H.264/AVC extended profile. Figure 5.12 shows how the concatenation of motion vectors is used to derive possible pixel locations over a GOP of four frames. However,

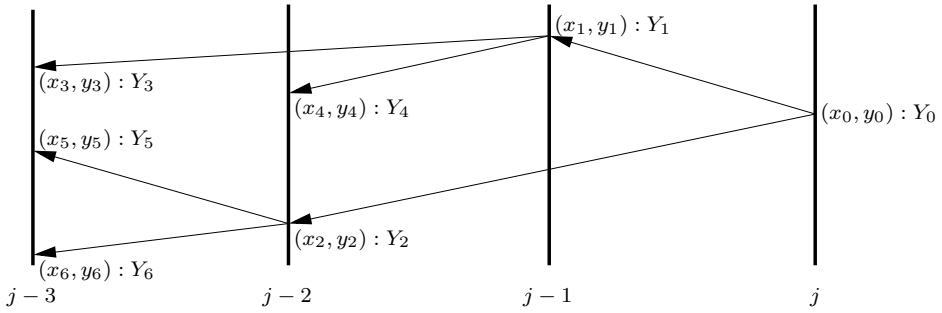


Figure 5.12: Starting at a pixel (x_0, y_0) with luminance Y_0 in an arbitrary B-frame i, possible trajectory locations are derived through the concatenation of motion vectors pointing to previously encoded B-frames.

not all of these describe the true motion of the pixel. It becomes therefore necessary to discard those motion vectors that have purely been chosen due to rate-distortion optimization and thus may not relate to the true motion of pixels. To this end three thresholds are used. In each of the following equations the motion vectors are scaled according to the temporal distance that they span.

Absolute Error Along the Trajectory

For every pixel the absolute difference of two consecutive luminance samples $\Delta Y_i = Y_{i+1} - Y_i$ together with the respective chrominance differences ΔU_i and ΔV_i are calculated. A sudden change in one of these differences is assumed to indicate that a motion vector no longer describes the true motion of a pixel. The trajectory is only continued if

$$\Delta Y_i < T, \Delta U_i < T, \Delta V_i < T, T = \begin{cases} 2T_Y, \mathrm{QP} < 30 \\ 4T_Y, \mathrm{QP} \geq 30 \end{cases} \tag{5.18}$$

for a given threshold T_Y, $0 \leq T_Y \leq 7$. In how far a hard decision in this context is justified, will be shown in Chapter 6. There soft-decision methods will also be used.

Spatial Motion Consistency and Temporal Consistency

Both the spatial and the temporal criterion used in [9] are here used in a similar manner as described in 5.1. However, the spatial motion consistency criterion given in Equation 5.6 is now extended to three bits as well, giving more flexibility to the encoder

$$BV_{i,0}(x, y) \leq 8 - T_{\mathrm{SC}}. \tag{5.19}$$

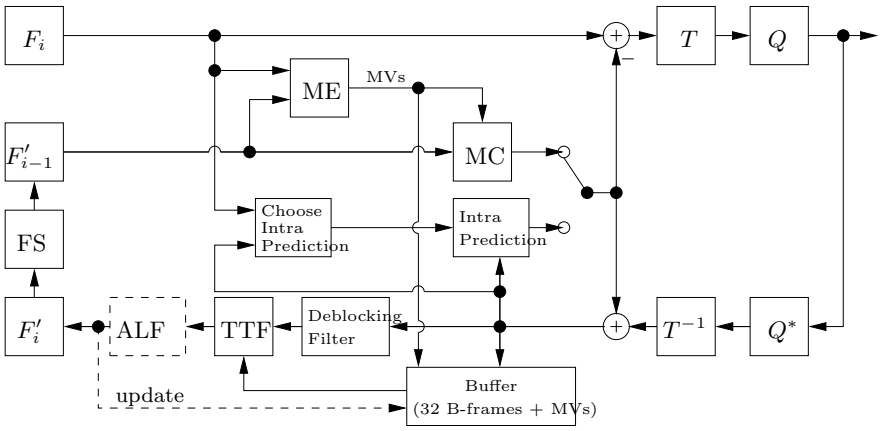


Figure 5.13: The TTF is included in the local decoder loop of the encoder after the Deblocking Filter. For the first test the ALF was disabled. When both ALF and TTF are used together, the respective frame in the TTF's buffer is updated after the ALF has been applied.

Parameter Calculation

All possible parameter combinations can be tested simultaneously at the encoder. The parameter combination yielding the minimum MSE is then selected. Each of the thresholds is transmitted to the decoder requiring nine additional bits per frame. A tenth bit can be used to disable the filter for the current frame altogether, in which case the other thresholds are simply omitted.

5.2.5 Experimental Evaluation

The TTF has been integrated into the HEVC test model HM 3.0. Up to 32 previously decoded unfiltered frames are kept in a buffer to be used for the trajectory formation. The resulting encoder is depicted in Figure 5.13, where the TTF is used after the Deblocking Filter. In this setting the ALF (dotted connections) was disabled. Tests have been conducted for a variety of sequences listed in Table 5.8. The exact configuration for the low-delay high efficiency setting may be found in [37].

Sequence	Resolution, framerate in Hz	TTF vs. HEVC		ALF vs. HEVC		TTF+ALF vs. HEVC		TTF+ALF vs. HEVC+ALF		TTF+ALF simple average	
		Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BlowingBubbles	416x240, 50	0.03	−0.70	0.05	−1.29	0.08	−1.92	0.03	−0.64	0.02	−0.42
BQSquare	416x240, 60	0.18	−4.77	0.21	−5.71	0.44	−11.48	0.23	−6.06	0.08	−2.21
RaceHorses	416x240, 30	0.00	−0.05	0.05	−0.99	0.00	−0.05	0.01	−0.13	0.01	−0.12
PartyScene	832x480, 50	0.02	−0.53	0.13	−3.13	0.15	−3.53	0.02	−0.42	0.01	−0.17
Vidyo1	1280x720, 60	0.02	−0.60	0.17	−4.95	0.17	−4.79	0.00	0.18	0.01	−0.35
Vidyo3	1280x720, 60	0.00	0.07	0.37	−10.38	0.38	−11.09	0.01	−0.27	0.00	−0.15
Vidyo4	1280x720, 60	0.01	−0.39	0.17	−5.42	0.18	−5.80	0.01	−0.38	0.01	−0.33
BQTerrace	1920x1080, 60	0.02	−1.46	0.17	−9.73	0.19	−10.41	0.01	−0.74	0.00	−0.21
ParkScene	1920x1080, 24	0.01	−0.38	0.07	−2.07	0.07	−2.29	0.01	−0.21	0.01	−0.11
Allstars	704x576, 25	0.01	−0.25	0.25	−7.46	0.25	−7.66	0.01	−0.21	0.01	−0.26
BBC-pan-13	720x576, 25	0.00	−0.11	0.21	−6.53	0.20	−6.41	0.00	0.11	0.00	0.10
Waterfall	704x480, 25	0.26	−8.95	0.29	−9.49	0.51	−16.50	0.23	−7.61	0.20	−6.83
average		**0.05**	**−1.52**	**0.18**	**−5.60**	**0.22**	**−6.83**	**0.05**	**−1.37**	**0.03**	**−0.92**

Table 5.8: BD-rates and average PSNR-gain for the sequences used in the experiments

The HM 3.0 without the ALF is compared against the HM with the added TTF using the Bjøntegaard metric described in [4]. For comparison, the individual gain provided by the ALF per sequence was also calculated. The resulting BD-rates for both filters may be found in Table 5.8 in columns 4 and 6. For all tested sequences except *Vidyo3* the TTF produces a bit rate reduction. For *Vidyo3* the increase of 0.07% is, however, only very slight. Nevertheless, the ALF produces a higher gain than the TTF for all tested sequences. This is only to be expected, as the ALF has undergone many significant improvements over the last years. Even though, both filters produce similar gains both for *BQSquare* and for *Waterfall*. The average bit rate reduction produced by the TTF for the dataset is 1.5%.

The true potential of both filters can be exploited when both are used in combination. In this setting, the average encoding time is increased by 190% compared to the HEVC encoder with the ALF. The decoder complexity, however, is only increased by about 30%. In order to show that ALF and TTF are not mutually exclusive, columns 8 and 10 of Table 5.8 compare HM 3.0 with and without ALF against the combination of ALF and TTF. In this case, a modified encoder as depicted in Figure 5.13 with the dotted connections is used. The combination of both filters outperforms the test model HM 3.0 for almost all sequences. For *BQSquare* and *BlowingBubbles* in particular, the gain produced by both filters together equals the sum of their individual quality improvements. A possible explanation for this finding may be different noise sources that are compensated separately by both filters, so that there is no unintended interference between the two approaches. In this combination, the TTF provides an average BD-rate of -1.4% when compared with the HEVC low-delay profile with ALF enabled. For comparison column 12 of Table 5.8 also shows the BD-rate produced by TTF and ALF if no long trajectories and a simple average instead of a weighted mean are used. The average BD-rate for the simplified filter is only -0.9%, which provides evidence for the effectiveness of the weighted filtering. An exemplary RD-curve for the *BQSquare* sequence is given in Figure 5.14. Below 500 kbit/s the TTF performs better than the ALF. For all depicted QPs the combination of both filters outperforms the simple ALF. Apart from objective quality improvements the TTF also increases the visual quality of the decoded video. Part of an exemplary decoded frame from the *Waterfall* sequence is shown in Figure 5.15.

5.2.6 Summary

The main objective of this section was to demonstrate the possiblity of further improving the HEVC test model HM 3.0 through the use of a temporal filtering approach. In combination with the optimal sample weighting described in Subsec-

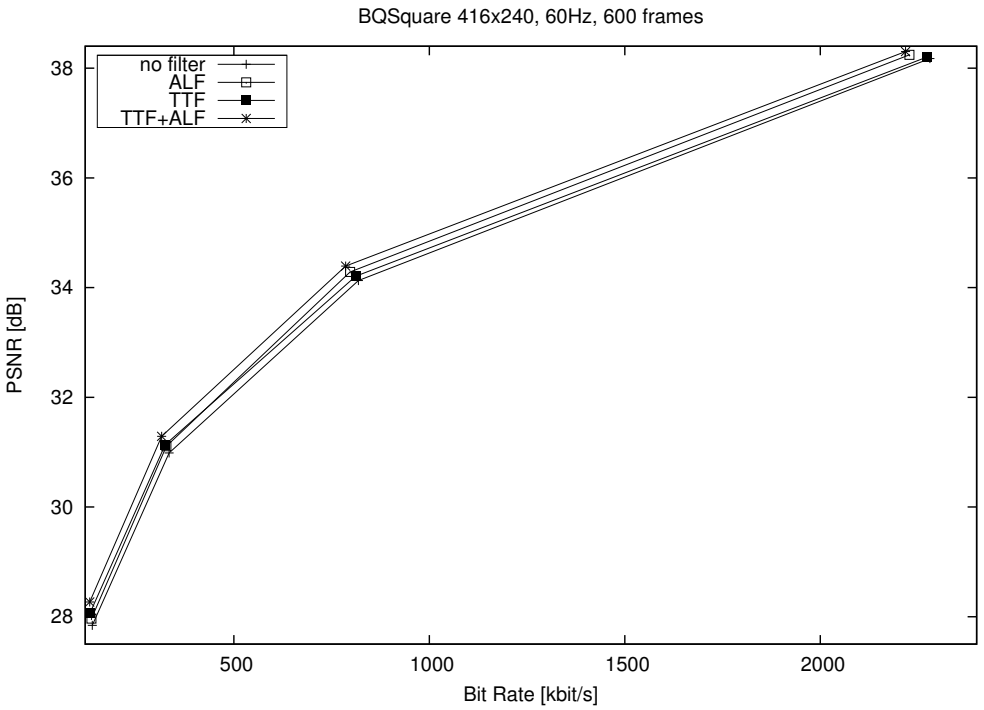
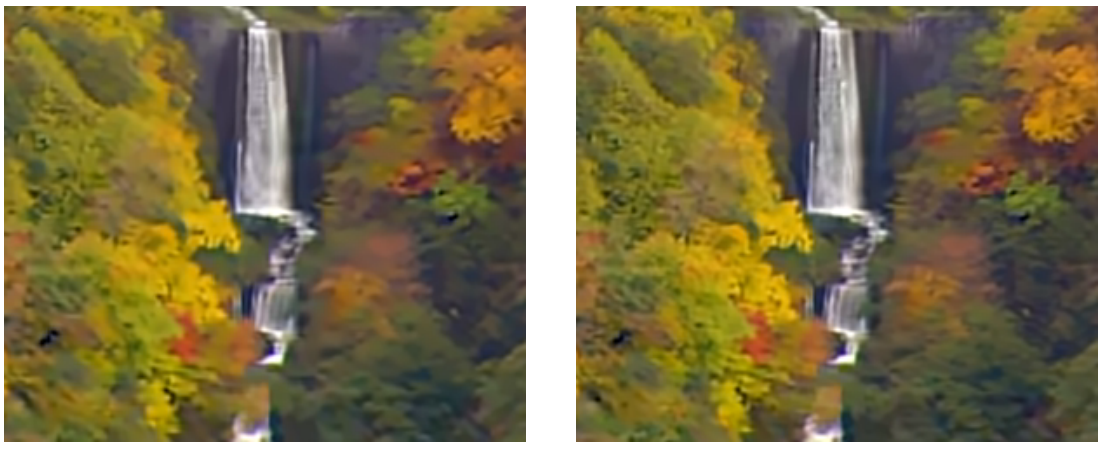


Figure 5.14: RD-curves for all three tested settings for the *BQSquare* sequence, QP 22 to 37.



(a) ALF, 31.1 dB at 69.3kbit/s

(b) TTF+ALF, 31.3 dB at 69.2kbit/s

Figure 5.15: Exemplary decoded frames from the *Waterfall* sequence.

tion 5.2.3, the proposed filter produces an average BD-rate of -1.4% when included in the HEVC test model. Additional improvements may be achieved by further investigating both long trajectories and weighted averaging separately.

5.3 Quadtree-Based Temporal Trajectory Filtering

The following section will further develop the adaptability of the TTF with a concept called quadtree partitioning. This technique has previously shown its suitability for filter control in the implementation of the quadtree-based adaptive loop filter [6]. It will be shown that the same approach can be used with great success within the TTF.

5.3.1 Introduction

In [10] a quadtree partitioning algorithm was added to the TTF with the aim of providing more adaptability to differently moving image areas. The Quadtree-based Temporal Trajectory Filter (QTTF) is examined more closely in this section. In addition, schemes for minimizing the overhead bit rate for the quadtree are presented and an optimal weighting scheme for the averaged luma samples is investigated.
The remainder of the section is structured as follows. The previously introduced TTF is revisited in Subsection 5.3.2 which also includes a motivation for the used filter parameters. Subsection 5.3.3 extends the TTF to the quadtree-based version. In addition, two methods for signaling the quadtree are introduced. Results of the experimental evaluation conducted within the environment of the test model HM 3.0 are reported in Subection 5.3.4. There the performance of QTTF is compared with the ALF also present in the early versions of HEVC. Subection 5.3.5 summarizes and concludes the Section.

5.3.2 Temporal Trajectory Filtering

In the original implementation of the TTF, all three thresholds were used for one entire frame at a time. They are optimized at the encoder with respect to the MSE. Each of the thresholds requires three bits with an additional flag bit to disable the filter if no parameter combination produces a MSE reduction. In this case the thresholds themselves are not transmitted. All three thresholds can, however, also be signaled on block level, which makes the filter much more adaptable to differently

moving regions within a frame. A method for obtaining block-based optimal filter parameters is now described.

5.3.3 Quadtree-Based Parameter Signaling

In [6] a scheme was detailed for recursively splitting a frame into a quadtree for adaptively applying the ALF only to certain image regions. A similar technique is now used to adapt the TTF to varying image content. For the QTTF a different set of thresholds is used for every quadtree partition. Both encoder and decoder shall observe the following rules when constructing or reconstructing the quadtree.

- Whenever a new block or frame is examined a *split_flag* is sent. Should the flag have the value "1" then the current block is split into four subblocks by dividing it once horizontally and vertically to produce four blocks of identical size. Each of these new blocks is then examined in turn.
- Should the flag instead be set to "0" then the splitting of the current block or frame is terminated and the combination of thresholds yielding the minimum MSE for the current block is transmitted to the decoder.

The following Subsection now details methods for finding optimal quadtree-partitions of a frame under certain constraints.

Flags and Solution Methods

As mentioned above, a *split_flag* per quadtree partition is needed to transmit the structure of the tree. In addition, an *enable_flag* is transmitted whenever the current block is not split again. If the *enable_flag* is set to "1" then the three thresholds as described in Subsection 5.3.2 are encoded. Otherwise the filter is disabled. Experiments showed that this strategy provides a much higher gain than setting all three thersholds to zero, since this would require nine additional bits. A first method for obtaining a quadtree suitable for QTTF with the associated filter parameters may be described as a *top-down* approach:
When the *top-down* approach is used, an optimal threshold combination is first calculated for the entire frame which also provides a minimal sum of squared differences $\mathrm{SSD}_{\mathrm{total}}$. The associated bit rate for transmitting the quadtree is $R_{\mathrm{total}} = 11\,\mathrm{bit}$ if the filter is enabled or $R_{\mathrm{total}} = 2\,\mathrm{bit}$ if the filter is disabled. Both bit rate and distortion can then be combined into a rate-distortion cost (RD-cost)

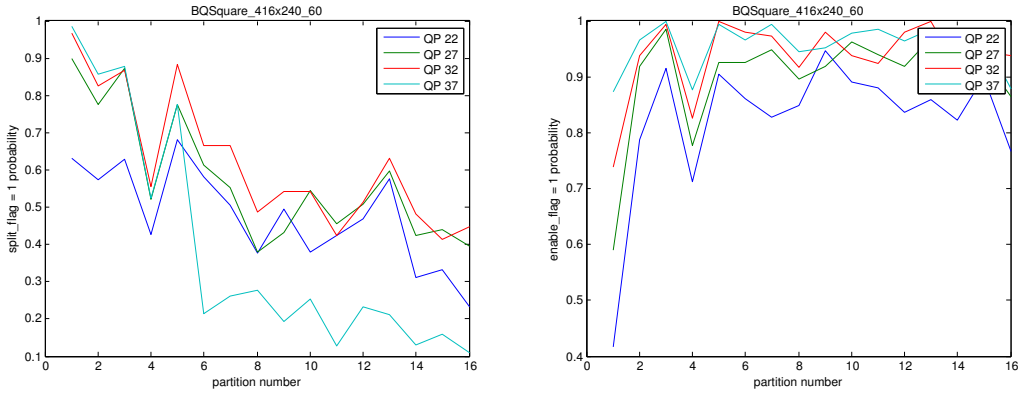
$$C_{\mathrm{total}} = \mathrm{SSD}_{\mathrm{total}} + R_{\mathrm{total}} \cdot \lambda, \tag{5.20}$$

where the Lagrangian multiplier λ combines bit rate and sum of squared differences into a single cost function. In this implementation the same λ as for the Adaptive Loop Filter in HM 3.0 is used. In a next step, a set of filter parameters for all four subpartitions of the current partition is computed. These yield four distortion values SSD_1 to SSD_4 with an associated bit rate R_{new}. The new distortion values and the required bit rate are again combined into a new RD-cost, which represents the cost of filtering the subpartitions separately instead of the entire partition on its own

$$C_{\mathrm{new}} = (\mathrm{SSD}_1 + \mathrm{SSD}_2 + \mathrm{SSD}_3 + \mathrm{SSD}_4) + R_{\mathrm{new}} \cdot \lambda. \tag{5.21}$$

The *top-down* algorithm essentially splits every partition as long as $C_{\mathrm{new}} < C_{\mathrm{total}}$. In order to distribute the partitions homogeneously over the entire frame, all partitions of equal size are examined before one of them is split. Blocks of size 32×32 are for example only examined when all 64×64 blocks have been processed. The main advantage of this approach is the relatively small time required to find a well-suited quadtree for a given frame. Nevertheless, it is not guaranteed that an optimal solution is found since the splitting is stopped as soon as the RD-cost no longer decreases with a new partition. A *brute-force* solution that eliminates this problem is now also developed with an algorithmic structure similar to the one described in [6].

Due to the linear dependency between distortion, rate, and RD-cost, each partition of the quadtree can be treated independently. Should an optimal solution be found for one out of subpartitions then this solution is also part of the optimal parameter configuration for the parent partition, unless it is less expensive to not split the parent partition at all. Based on this principle the following *brute-force* algorithm can be used. Starting at the smallest possible partition level an optimal combination of thresholds is calculated for every partition. Afterwards the next higher layer of partitions is examined. For each of these the cost of signaling thresholds on this level with the associated SSD is compared with the RD-cost of signaling the thresholds on the next lower partition level. Based on this comparison the optimal solution per partition is updated and the algorithm proceeds with the next higher hierarchy level in the quadtree. This manner of operation guarantees that the quadtree remains optimal in all its subtrees. The optimality of this solution was analytically proven by Wiegand et al. in [52] where the authors also first described a complete quadtree-based video codec. Both the *top-down* approach and the *brute-force* solution are examined in Subsection 5.3.4. As will be shown the resulting quadtree can take up a significant number of bits per frame. For this reason a context modeling approach to compress the quadtree with the HM's CABAC [36] engine is now investigated.



(a) *split_flag* distribution (b) *enable_flag* distribution

Figure 5.16: Exemplary probability distributions for the *BQSquare* sequence for QP values from 22 to 37.

Dedicated Context Models for CABAC

Figure 5.16 shows exemplary distributions of the *split_flag* and the *enable_flag* for the *BQSquare* sequence. The partition number is given hierarchically to the quadtree nodes. The first partition is always the entire frame. The four subpartitions of the frame have the partition numbers 2 to 5. When examining these distributions it becomes obvious that the probability for $split\_flag = 1$ and $enable\_flag = 1$ depends strongly on the employed QP. Once a frame has at least been split once, the probability, that the *enable_flag* is set, is increased significantly. A similar obervation can be made for the *split_flag*, where the probability, that the frame is split into four subpartitions depends essentially on the QP. After the initial splitting operation, all splitting probabilities are in the range of 50% with the exception of QP 37 where the splitting of a partition is much less likely. These observations justify the use of two separate context models per flag. The first one models the probability that the *split_flag* or *enable_flag* is set for the entire frame. The second one represents the probability for one of the two flags for all remaining partitions. When conducting the experiments, it became apparent that the probabilities for both flags also depend on the position within a group of pictures (GOP). In the HEVC test model HM 3.0 utilizing the low-delay high efficiency setting only the very first frame is an Intra-frame. Afterwards the same QPs are assigned periodically to four consecutive frames: $\{\mathrm{QP}+1, \mathrm{QP}+3, \mathrm{QP}+2, \mathrm{QP}+3\}$, where QP is the quantization parameter for the Intra-frame. According to the relative position in one of these GOPs the probabilities for *split_flag* and *enable_flag* were measured. The resulting probabilities for the four possible context models along with their associated CABAC-states are provided in Subsection 5.3.4. Due to the additional computation time required by the

CABAC engine, CABAC was only used after a quadtree had been constructed and not during the optimization of the quadtree itself. There, only bins were counted. In all experimental settings where CABAC was not used, the quadtree was simply conveyed in the slice header as raw data.

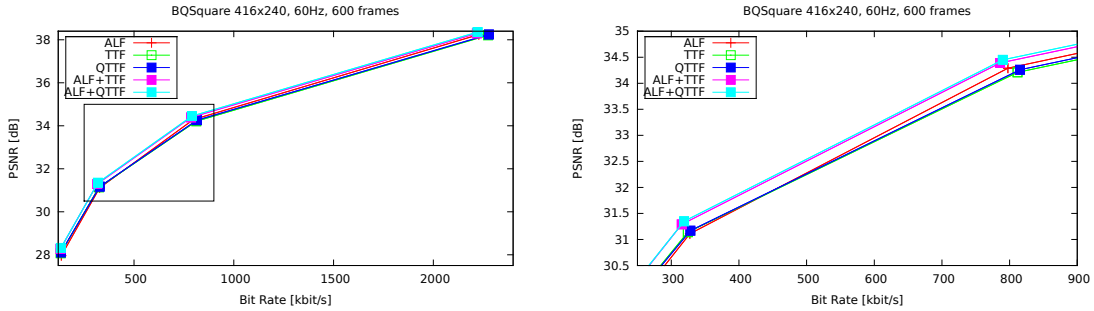
5.3.4 Experimental Evaluation

The QTTF algorithm has been integrated into the HEVC test model HM 3.0. Tests were performed on all HEVC test sequences as listed in Table 5.9 for the low-delay high efficiency setting of HEVC. For comparison the respective results provided by the TTF as reported in [10] are also reproduced here. Each sequence was encoded at four QPs from 22 to 37. The resulting bit rates and PSNR values are compared using the Bjøntegaard metric [4]. In addition, the gain provided by the ALF algorithm is listed separately for comparison as well. Table 5.10 provides results for both

	Sequence	Resolution	Frames	Frequency
Class D	*BasketballPass*	416x240	500	50 Hz
	BlowingBubbles	416x240	500	50 Hz
	BQSquare	416x240	600	60 Hz
	RaceHorses	416x240	300	30 Hz
Class C	*BasketballDrill*	832x480	500	50 Hz
	BQMall	832x480	600	50 Hz
	PartyScene	832x480	600	50 Hz
	RaceHorses	832x480	300	30 Hz
Class E	*Vidyo1*	1280x720	600	50 Hz
	Vidyo3	1280x720	600	50 Hz
	Vidyo4	1280x720	600	50 Hz
Class B	*BasketballDrive*	1920x1080	500	50 Hz
	BQTerrace	1920x1080	600	60 Hz
	Cactus	1920x1080	500	50 Hz
	Kimono1	1920x1080	240	24 Hz
	ParkScene	1920x1080	240	24 Hz

Table 5.9: All test sequences used in the experiments with their respective resolutions. The number of frames corresponds in each case to a 10 s video sequence.

TTF and QTTF compared against the HEVC low-delay profile with ALF disabled. For better comparability the gain produced by the ALF itself is also shown. In addition, the combined gain of ALF and TTF or QTTF respectively is provided. In general, the ALF performs better than both trajectory filters. Since the ALF has, however, been optimized over a period over several years, this is only to be expected.



(a) QP 22 to 37 (b) QP 27 to 32

Figure 5.17: Exemplary RD-curves for the *BQSquare* sequence.

In addition, similar bit rate reductions are achieved for *BlowingBubbles*, *BQSquare*, and *PartyScene*. As can be seen for example from the sequences *BlowingBubbles* and *BQSquare*, the combined gain of ALF and QTTF can be greater than the sum of the gains provided by the filters individually. Figure 5.17 shows the respective RD-curves for all tested filters and their combinations for the *BQSquare* sequence. As shown in the enlarged part on the right, the QTTF performs better than ALF in the lower bit rate range. In addition, the combination of ALF and QTTF significantly outperforms the individual application of each filter.

Sequence	TTF		QTTF		ALF		TTF+ALF		QTTF+ALF	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballPass	0.01	−0.10	0.01	−0.13	0.08	−1.55	0.08	−1.59	0.08	−1.62
BlowingBubbles	0.03	−0.70	0.03	−0.81	0.05	−1.29	0.08	−1.92	0.09	−2.19
BQSquare	0.18	−4.77	0.20	−5.27	0.21	−5.71	0.44	−11.48	0.47	−12.24
RaceHorses	0.00	−0.05	0.01	−0.24	0.05	−0.99	0.05	−1.12	0.06	−1.18
avg. for class D	**0.06**	**−1.41**	**0.06**	**−1.61**	**0.10**	**−2.39**	**0.16**	**−4.03**	**0.18**	**−4.31**
BasketballDrill	0.00	0.01	0.01	−0.21	0.22	−5.41	0.23	−5.53	0.22	−5.51
BQMall	0.02	0.62	0.00	−0.10	0.13	−3.29	0.14	−3.37	0.14	−3.35
PartyScene	0.02	−0.53	0.04	−1.05	0.13	−3.13	0.15	−3.54	0.15	−3.54
RaceHorses	0.00	−0.13	0.00	−0.13	0.10	−2.51	0.10	−2.55	0.10	−2.64
avg. for class C	**0.01**	**−0.32**	**0.01**	**−0.37**	**0.15**	**−3.59**	**0.16**	**−3.75**	**0.15**	**−3.76**
Vidyo1	0.02	−0.60	0.03	−0.71	0.17	−4.95	0.17	−4.79	0.19	−5.47
Vidyo3	0.00	0.07	0.01	−0.18	0.37	−10.83	0.38	−11.09	0.38	−11.08
Vidyo4	0.01	−0.39	0.02	−0.78	0.17	−5.42	0.18	−5.80	0.19	−6.07
avg. for class E	**0.01**	**−0.31**	**0.02**	**−0.56**	**0.24**	**−7.07**	**0.24**	**−7.23**	**0.25**	**−7.54**
BasketballDrive	0.00	−0.01	0.00	−0.05	0.14	−5.65	0.14	−5.67	0.14	−5.66
BQTerrace	0.02	−1.46	0.03	−1.96	0.17	−9.73	0.19	−10.41	0.19	−10.92
Cactus	0.00	−0.14	0.01	−0.29	0.09	−3.60	0.09	−3.76	0.10	−3.91
Kimono1	0.00	−0.03	0.01	−0.16	0.15	−4.60	0.15	−4.59	0.15	−4.68
ParkScene	0.01	−0.38	0.02	−0.62	0.07	−2.07	0.07	−2.29	0.08	−2.60
avg. for class B	**0.01**	**−0.40**	**0.01**	**−0.62**	**0.12**	**−5.13**	**0.13**	**−5.34**	**0.13**	**−5.55**
Waterfall	0.26	−8.95	0.31	−10.26	0.29	−9.49	0.51	−16.50	0.56	−17.85

Table 5.10: BD-rate and average PSNR gain for all test sequences compared against the HEVC low-delay profile with ALF disabled.

Dedicated CABAC Context Models

In Subsection 5.3.3 the use of four separate context models for CABAC has been motivated. To derive reliable initial values for each of these, a training data set consisting of four TV sequences was chosen and encoded at QPs from 22 to 37. The sequences used may be found in Table 5.11. The resulting probabilities for the CABAC states are given in Tables 5.12, 5.13, 5.14, 5.15. In Table 5.16 two separate

Sequence	Resolution	Frames	Frequency
Allstars	704x576	250	25 Hz
BBC-Pan-13	720x576	110	25 Hz
Desert	720x400	240	25 Hz
Stanford	720x480	300	25 Hz

Table 5.11: Training sequences used to develop the CABAC context models.

iQP	pos0	pos1	pos2	pos3
22	0.09	0.10	0.11	0.10
27	0.13	0.22	0.26	0.22
32	0.17	0.31	0.36	0.29
37	0.14	0.26	0.39	0.35

Table 5.12: Probability for the *enable_flag* for partition number 1 depending on the position within the GOP.

iQP	pos0	pos1	pos2	pos3
22	0.85	0.84	0.86	0.77
27	0.86	0.87	0.88	0.88
32	0.88	0.90	0.92	0.90
37	0.92	0.87	0.90	0.91

Table 5.13: Probability for the *enable_flag* for partition number > 1 depending on the position within the GOP.

mechanisms to further improve the performance of QTTF are examined. The third column shows the BD-rate provided by the combination of QTTF and ALF. When the proposed CABAC models are applied to the quadtree, a reduced bit rate can be observed for almost all sequences as shown in column 5. A very slight loss, due to an ill adapted context model, is only present for the *RaceHorses* sequence of class C and for the *BQMall* sequence. On average, the use of CABAC increases the bit rate reduction by a further 0.1%. As has been discussed in Subsection 5.3.3 the QTTF

iQP	pos0	pos1	pos2	pos3
22	0.12	0.12	0.12	0.12
27	0.07	0.10	0.09	0.12
32	0.06	0.06	0.06	0.12
37	0.02	0.02	0.03	0.09

Table 5.14: Probability for the *split_flag* for partition number 1 depending on the position within the GOP.

iQP	pos0	pos1	pos2	pos3
22	0.26	0.32	0.38	0.23
27	0.25	0.47	0.49	0.44
32	0.20	0.40	0.41	0.54
37	0.06	0.17	0.15	0.53

Table 5.15: Probability for the *split_flag* for partition number > 1 depending on the position within the GOP.

can also be improved through the calculation of an optimal quadtree. The results of the described *brute-force* approach are given in column 7. Again the original implementation of QTTF is outperformed, except for the *RaceHorses* sequence of class C and for the *Vidyo1* sequence. On average the *brute-force* solution also adds a further 0.1% BD-rate to the bit rate reduction of QTTF, which for some sequences essentially doubles the PSNR gain.

Sequence	original		CABAC		brute-force		CABAC + brute-force	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballPass	0.00	−0.07	0.01	−0.19	0.01	−0.20	0.01	−0.22
BlowingBubbles	0.04	−0.91	0.04	−1.09	0.04	−0.99	0.04	−1.02
BQSquare	0.26	−6.86	0.30	−7.85	0.26	−6.94	0.27	−7.07
RaceHorses	0.01	−0.19	0.01	−0.26	0.01	−0.20	0.01	−0.21
avg. for class D	**0.08**	**−2.01**	**0.09**	**−2.35**	**0.08**	**−2.08**	**0.08**	**−2.13**
BasketballDrill	0.00	−0.11	0.00	−0.13	0.01	−0.15	0.01	−0.16
BQMall	0.00	−0.06	0.00	0.00	0.00	−0.08	0.00	−0.09
PartyScene	0.05	−1.09	0.05	−1.26	0.05	−1.25	0.05	−1.24
RaceHorses	0.01	−0.14	0.00	−0.11	0.00	−0.09	0.00	−0.09
avg. for class C	**0.02**	**−0.35**	**0.01**	**−0.38**	**0.02**	**−0.39**	**0.02**	**−0.40**
Vidyo1	0.02	−0.53	0.02	−0.53	0.02	−0.52	0.02	−0.54
Vidyo3	0.01	−0.27	0.02	−0.48	0.01	−0.33	0.01	−0.35
Vidyo4	0.02	−0.67	0.02	−0.78	0.02	−0.71	0.02	−0.73
avg. for class E	**0.02**	**−0.49**	**0.02**	**−0.60**	**0.02**	**−0.52**	**0.02**	**−0.54**
BasketballDrive	0.00	−0.01	0.00	−0.06	0.00	0.00	0.00	−0.06
BQTerrace	0.02	−1.30	0.02	−1.40	0.02	−1.46	0.02	−1.48
Cactus	0.01	−0.32	0.01	−0.39	0.00	−0.23	0.01	−0.39
Kimono1	0.00	−0.08	0.00	−0.13	0.00	−0.08	0.00	−0.09
ParkScene	0.02	−0.53	0.02	−0.56	0.02	−0.56	0.02	−0.58
avg. for class B	**0.01**	**−0.45**	**0.01**	**−0.51**	**0.01**	**−0.47**	**0.01**	**−0.52**

Table 5.16: BD-rate and average PSNR gain for QTTF in combination with different quadtree optimization and compression techniques compared against the HEVC low-delay profile with ALF enabled.

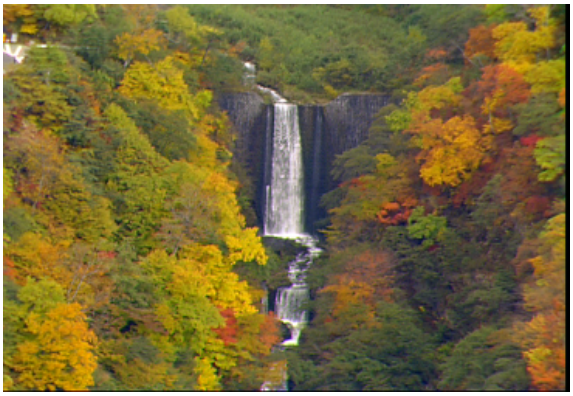
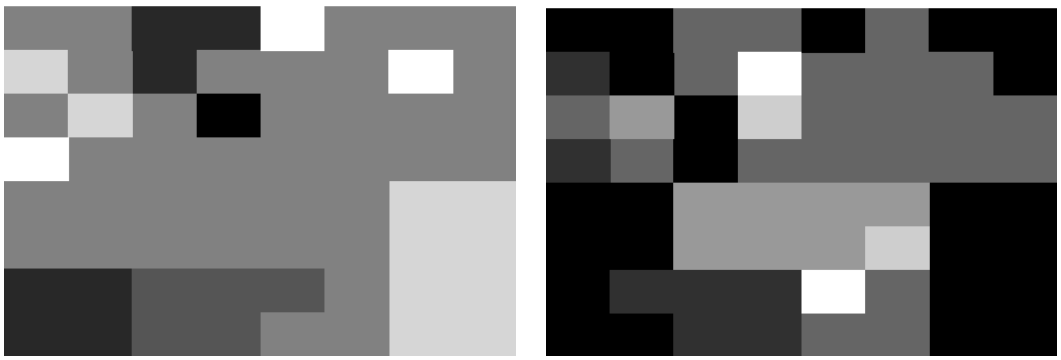


Figure 5.18: Frame 44 of the *Waterfall* sequence.

The above values are only restricted to the official MPEG set of test sequences. If, however, sequences with significant global motion are used, the performance of QTTF is even better. Even though these sequences do not fit the translation motion model of TTF, they are much better described by individual pixel motion than by the block-based approch alone. One example of such a sequence is *Waterfall*, which consists essentially of a still background and a zoom-out performed by a hand-held camera. For this sequence, which is also publicly available, TTF alone provides a bit rate reduction of 9.10%. This can be increased to 9.64% when QTTF is used, instead. Frame 44 of the *Waterfall* sequence is shown in Figure 5.18.

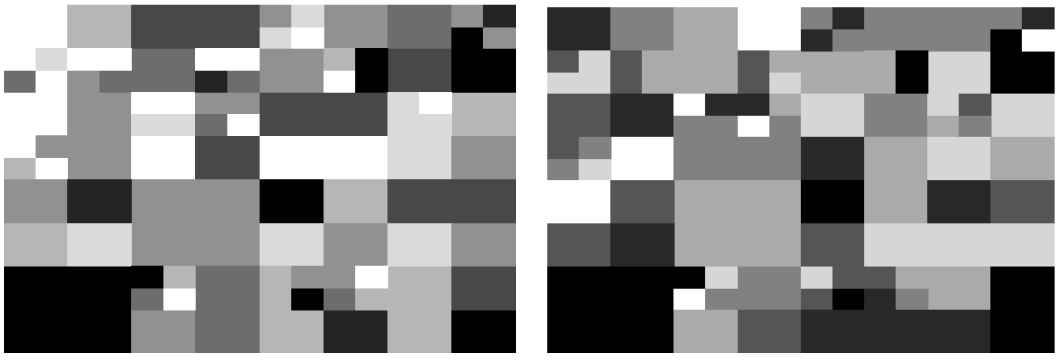
Analysis of the Quadtree Structure

In the previous section the effectiveness of the *brute-force* solution has been examined in terms of the resulting BD-rate. It can be assumed, that the QTTF will perform better depending on the depth of the associated quadtree. A larger quadtree will of course also mean more side information, which automatically increases the required bit rate for a given quality level. As shown in Figure 5.19 the smallest partition computed for frame 44 of the *Waterfall* sequence is significantly larger than the minimum possible size of 16×16 pixels. When, however, the *brute-force* method is used, the resulting quadtree has a lot more detail as is shown in Figure 5.20. Despite the increased amount of side information the modified QTTF still performs better. Since the filtering process for both TTF and QTTF is straight forward and requires no additional computations, the decoder runtime is only increased by 20% on average. Values for individual sequences may, however, differ significantly from the average, depending on the number of frames for which the respective filter is active. Adding the TTF to the HEVC encoder increases its runtime by 40% on average. For QTTF the *top-down* solution requires 80% additional runtime and the



(a) *top-down* solution: T_Y (b) *top-down* solution: T_{TC}

Figure 5.19: Exemplary quadtree partitions with associated threshold values for frame 44 of the *Waterfall* sequence computed using the *top-down* solution. Black corresponds to a threshold value of 0 and white to a threshold of 7.



(a) *brute-force* solution: T_Y (b) *brute-force* solution: T_{TC}

Figure 5.20: Exemplary quadtree partitions with associated threshold values for frame 44 of the *Waterfall* sequence computed using the *brute-force* solution. Black corresponds to a threshold value of 0 and white to a threshold of 7.

brute-force increases the encoder complexity by 110%. These values do, however, not necessarily reflect the complexity of the underlying algorithm, but are strongly influenced by the actual implementation, which as of yet has not been optimized for runtime, but makes use of conventional floating point arithmetics.

5.3.5 Summary

The proposed QTTF algorithm in its simplest configuration has been shown to provide an average bit rate reduction of 0.8% for the HEVC set of test sequences. Through the use of CABAC in combination with a *brute-force* solution the average bit rate reduction can be increased to 0.9%. Even though the QTTF itself does not yet perform as well as the ALF, it still provides gains of similar magnitude. Moreover, constructive interference between both filters can be observed for several sequences. To further improve the performance of QTTF, two approaches will be investigated further. Firstly, the quality of the motion vector field may be improved through interpolation between neighboring motion-compensated blocks. This option will be discussed in Section 5.5. Secondly, a way of continuing those trajectories, that have been interrupted, remains a subject for future work. In this scenario a lot more pixels could be filtered thus increasing the achieved PSNR gain.

5.4 A Flexible Side-Information Compression Scheme

As has been shown in Subsection 5.3.3 the side-information needed to transmit the QTTF's quadtree can be reduced by using dedicated context models for the individual flags. Further investigation into the statistical properties of these flags showed, that adding a temporal component to the context models drastically improves their efficiency. Usually CABAC models are reset if a new slice is transmitted, however, allowing a model to adapt itself over several frames can also be beneficial. In this case the final CABAC state from the previous slice just needs to be transmitted in the slice header of the subsequent one. The new algorithm is evaluated in the context of the HEVC test model HM 8.0 which is technically almost identical to the final version HM 10.0. The remainder of this section is structured as follows. Firstly the statistical properties of the *split_flag* and the *enable_flag* are reexamined and new context models are derived. Secondly, said models are evaluated in terms of the resulting BD-rate when these models are utilized within the QTTF. Finally, the compression ratio for the quadtree information is also measured.

5.4.1 Designing CABAC Context Models for QTTF

All entropy coding in the HEVC test model HM 8.0 is done using CABAC. This arithmetic coder mainly profits from precise predictions of individual symbol probabilities, in which case high compression ratios can be achieved. Consequently, the compression scheme for the quadtree will also be based on CABAC. The main portion of the side-information consists of the *split_flag* and *enable_flag*. The thresholds and threshold combinations themselves are essentially random and will not be examined here. In the case of the flags, the situation is, however, different. As will be shown, the *split_flag* and *enable_flag* represent highly correlated information. Simulations conducted on a set of test sequences showed, that both flags strongly depend on the size of the current partition. Partitions shall now no longer be referred to by a partition number but by their hierarchy level. Partition level 1 is the entire frame. Blocks with a partition level 2 are now the four equally sized subpartitions of the top-level partition. Since the highest tested resolution was 1080p material and partitions are restricted to be no smaller than 16×16, the smallest partition thus has a partition level of 8. For both flags and for all tested quantization parameters (QP 22, 27, 32, and 37) statistics were obtained for a set of training sequences listed in Table 5.4.1. As long as the partition levels are kept separately, their statistics

Sequence	Resolution	Frames	Frequency
Allstars	704x576	250	25 Hz
BBC-Pan-13	720x576	110	25 Hz
Desert	720x400	240	25 Hz
Stanford	720x480	300	25 Hz

Table 5.17: Training sequences used to develop the CABAC context models.

show quite distinct characterisits with very little variation: The *split_flag* largely depends on the image content, i.e. foreground objects, texture and object movement. Therefore, the respective CABAC model is only allowed to adapt itself within a single frame. Afterwards, the context model is reinitialized with its default context state and MPS value. The *enable_flag*, however, always shows similiar behavior throughout the duration of a sequence. The initialization is, therefore, only done at the beginning of the sequence. Afterwards the model is only reset to its original state, when an I-frame occurs. The observed average probabilities for both flags and the associated CABAC states (consisting of a probability state and a most probable symbol (MPS)) are given in Table 5.18. The general applicability of these context models will be shown in Subsection 5.4.2.

QP	partition level	*split_flag*		*enable_flag*	
		probability	(state, MPS)	probability	(state, MPS)
22	1	26.96%	(11, 0)	2.09%	(60, 0)
	2	31.32%	(8, 0)	30.41%	(9, 0)
	3	35.54%	(6, 0)	48.37%	(0, 0)
	4	21.38%	(16, 0)	66.82%	(7, 1)
	5	8.65%	(33, 0)	80.27%	(11, 1)
	6	0.66%	(63, 0)	51.08%	(2, 1)
	7	0.00%	(63, 0)	8.33%	(34, 0)
	8	0.00%	(63, 0)	7.50%	(36, 0)
27	1	37.16%	(5, 0)	4.59%	(45, 0)
	2	34.00%	(7, 0)	40.38%	(4, 0)
	3	26.13%	(12, 0)	60.95%	(5, 1)
	4	11.70%	(27, 0)	79.58%	(10, 1)
	5	2.04%	(61, 0)	77.57%	(10, 1)
	6	0.71%	(63, 0)	24.62%	(13, 0)
	7	0.00%	(63, 0)	8.38%	(34, 0)
	8	0.00%	(63, 0)	7.50%	(36, 0)
32	1	34.94%	(6, 0)	7.76%	(35, 0)
	2	32.32%	(8, 0)	48.80%	(0, 0)
	3	24.87%	(13, 0)	66.69%	(7, 1)
	4	15.61%	(22, 0)	76.11%	(10, 1)
	5	6.43%	(39, 0)	75.83%	(9, 1)
	6	0.24%	(63, 0)	53.25%	(3, 1)
	7	0.01%	(63, 0)	8.47%	(34, 0)
	8	0.00%	(63, 0)	2.50%	(57, 0)
37	1	21.14%	(16, 0)	13.32%	(25, 0)
	2	27.00%	(11, 0)	55.49%	(3, 1)
	3	18.36%	(19, 0)	63.79%	(6, 1)
	4	6.86%	(38, 0)	76.32%	(10, 1)
	5	0.34%	(63, 0)	67.44%	(7, 1)
	6	0.42%	(63, 0)	16.85%	(20, 0)
	7	0.00%	(63, 0)	7.50%	(36, 0)
	8	0.00%	(63, 0)	0.00%	(63, 0)

Table 5.18: Average symbol probability for both flags and all eight possible partition levels. In addition, the corresponding CABAC state (consisting of state index and MPS) is also given.

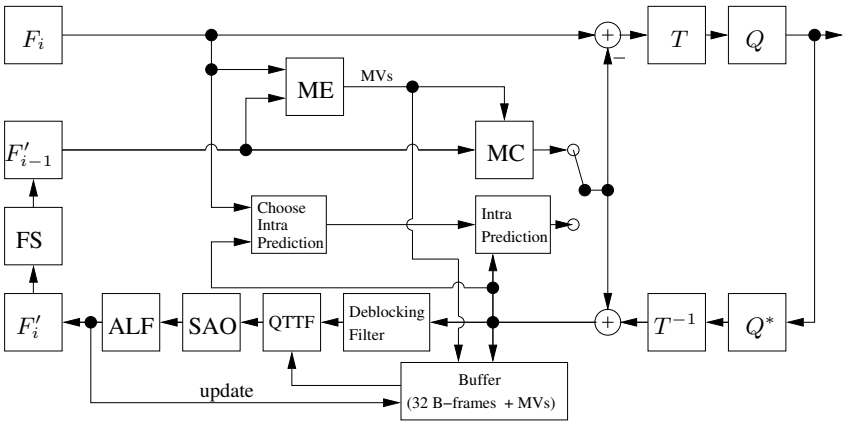


Figure 5.21: The QTTF is inserted in the local decoding loop at the encoder after the Deblocking Filter.

5.4.2 Experimental Evaluation

The QTTF has been implemented in C++ and integrated into the HEVC test model HM 3.0. The resulting modified encoder is shown in Figure 5.21. Simulations were run for the test sequences listed in Table 5.19 for the low delay, high efficiency setting as described in [37]. For better comparability results are reported both for QTTF with and without the application of the new CABAC context models. When CABAC is not used, the side-information is transmitted as raw uncompressed data in the slice header. The obtained bit rates and PSNR values were compared with the HM 3.0 anchor data using the Bjøntegaard metric [4]. The respective average PSNR gains and BD-rates may be found in columns 3 to 6 of Table 5.19. Most importantly, the QTTF produces no loss for either setting. For individual sequences bit rate reductions of up to 9.05% can be observed with an average of 1.70%. As can be seen, the QTTF does not work equally well for all sequences. However, the performance only depends on the actual video content, not on resolution or frame rate.

With the application of the new CABAC context models the average bit rate reduction is increased to 1.77%. Moreover, the performance of the filter is always improved. For individual sequences the CABAC encoding adds another 0.3% bit rate reduction to the original gain.The current implementation of QTTF increases the HM 3.0 encoding runtime by 110%. The decoder complexity is increased by 20% on average. For comparison the same filter has been integrated into the HM 8.0 software as well. Here ,the filter still produces gain compared with the HM 8.0 anchor, but the average bit rate reduction is reduced to 0.26%. Further investigation of the interactions with other tools in the current test model, such as the improved

SAO, will therefore be necessary. Most importantly the new HM generally seems to cope better with video sequences containing global motion such as *Waterfall* and *BQSquare*. The suitability of the context models for the compression of quadtrees is illustrated by Table 5.20. Here the average bit rates for the uncompressed and compressed side-information are given. A maximum bit rate saving of 55.35% for the *Vidyo1* sequence is observed. On average the side-information is reduced by 41.9%. Good compression performance is present for both the sequences where the filter is used frequently and for those where it is mainly switched off. This becomes apparent when setting the bit rate savings into the context of the observed BD-rates. I.e. for such diverse sequences as *RaceHorses* and *BQTerrace* almost identical bit rate savings are produced by the CABAC compression.

Sequence	Resolution	HM 3.0 + QTTF		HM 3.0 + QTTF + CABAC		HM 8.0 + QTTF + CABAC	
		Δ_{PSNR}	BD-rate	Δ_{PSNR}	BD-rate	Δ_{PSNR}	BD-rate
BasketballPass	416x240, 50Hz	0.01 dB	−0.20%	0.01 dB	−0.23%	0.01 dB	−0.13%
BlowingBubbles	416x240, 50Hz	0.04 dB	−0.99%	0.04 dB	−1.09%	0.00 dB	−0.06%
BQSquare	416x240, 50Hz	0.26 dB	−6.94%	0.27 dB	−7.24%	0.06 dB	−1.74%
RaceHorses	416x240, 50Hz	0.01 dB	−0.20%	0.01 dB	−0.23%	0.00 dB	−0.03%
BasketballDrill	832x480, 50Hz	0.01 dB	−0.15%	0.01 dB	−0.16%	0.00 dB	0.01%
BQMall	832x480, 50Hz	0.00 dB	−0.08%	0.00 dB	−0.10%	0.01 dB	0.00%
PartyScene	832x480, 50Hz	0.05 dB	−1.25%	0.06 dB	−1.33%	0.00 dB	−0.08%
RaceHorses	832x480, 50Hz	0.00 dB	−0.09%	0.01 dB	−0.10%	0.00 dB	−0.08%
Vidyo3	1280x720, 60Hz	0.01 dB	−0.33%	0.01 dB	−0.37%	0.01 dB	−0.35%
Vidyo4	1280x720, 60Hz	0.02 dB	−0.71%	0.02 dB	−0.75%	0.01 dB	−0.14%
ParkScene	1920x1080, 60Hz	0.02 dB	−0.60%	0.02 dB	−0.60%	0.00 dB	0.05%
BQTerrace	1920x1080, 60Hz	0.02 dB	−1.46%	0.02 dB	−1.53%	0.01 dB	−0.31%
Waterfall	704x480, 25Hz	0.27 dB	−9.05%	0.28 dB	−9.37%	0.01 dB	−0.49%

Table 5.19: All used test sequences with their respective Δ_{PSNR} and BD-rate values for all three tested settings.

Sequence	Uncompressed in kBit/s	CABAC in kBit/s	Savings
BasketballPass	0.42	0.21	50.79%
BlowingBubbles	1.96	1.26	35.47%
BQSquare	7.84	5.01	36.07%
RaceHorses (small)	0.34	0.18	47.49%
BasketballDrill	0.35	0.18	49.75%
BQMall	0.58	0.32	44.70%
PartyScene	6.44	4.38	32.03%
RaceHorses	0.40	0.24	39.49%
Vidyo3	0.46	0.21	55.35%
Vidyo4	1.34	0.93	30.35%
ParkScene	3.30	2.17	34.37%
BQTerrace	9.79	6.22	36.53%
Waterfall	2.70	1.27	52.93%

Table 5.20: Average bit rate needed to signal the QTTF side-information (QP 22 to 37) with and without application of the CABAC context models for HM 3.0.

5.5 Adpative Dense Vector Field Interpolation for Temporal Filtering

Apart from optimizing the encoder and analyzing both new threshold combinations and signaling schemes, there exists another way to improve the TTF. As illustrated in Chapter 4 the quality of the available motion information has a significant impact on the filter's performance. In this section, therefore, a scheme to better explore the block-based motion vector field transmitted in the HEVC bit stream is presented.

The following Sections 5.5.1 to 5.5.4 were first published in [15] ©IEEE 2013.

5.5.1 Introduction

The Quadtree-based Temporal Trajectory Filter (QTTF) selectively concatenates motion vectors (MV) transmitted in the bit stream of the encoded video sequence to construct individual motion trajectories over several frames for each pixel. As has been described before, the filter is controlled by three thresholds which are signaled along with a quadtree structure to apply different parameter settings to different image regions. The filter performs better when more accurate motion information

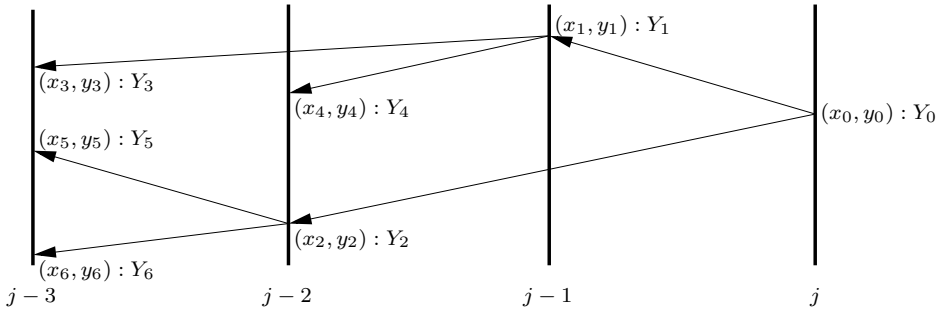


Figure 5.22: The trajectory starting at an arbitrary pixel $(x_0, y_0)^T$ in frame j is split into two subtrajectories at every new B-frame. The number of samples that can potentially be used for filtering is thus increased exponentially.

is available, as longer trajectories help to improve the filter performance. However, accurate motion information is mainly only available at very low QP values, where less artifacts occur. In this section, a method is described to derive accurate motion information at pixel level from block-based motion vectors for temporal filtering. Previous work in the area of dense motion interpolation [21] always required pixel-wise motion estimation on the raw video data which is of course not available at the decoder. The remainder of the section is structured as follows: The dense motion field interpolation algorithm used here is detailed in Subsection 5.5.2. In Subsection 5.5.3 the algorithm is firstly evaluated based on a manually annotated sequence with ground truth motion. Secondly, the compression performance of the improved filter is tested within the HEVC test model HM 8.0. All experiments conducted here made use of the HEVC main profile which uses an IBBB coding structure. In this setting each block has up to two motion vectors with different reference frames as shown in Figure 5.22. The basic functionailty of the QTTF algorithm with its three thresholds that are signaled via a quadtree is here simply retained. The new extension only acts as a pre-processing step on the block-based motion vectors.

5.5.2 Motion Field Interpolation

In order to obtain more accurate motion information, an interpolation algorithm is now applied to the transmitted motion vector field. Firstly, all motion vectors are normalized according to their temporal distance, i.e. the number of frames that they span. It is assumed, that each motion vector accurately describes the center pixel of its Prediction Unit (PU), which corresponds to a motion compensated block in H.264/AVC. The closer an arbitrary pixel is to such a block center, the closer its actual motion will be to the block motion. Since pixels between block centers may have been falsely combined into a single block, they are interpreted as being

influenced by their three closest neighboring block centers. To determine which pixel-based motion vectors are influenced by which blocks, a Delaunay triangulation is performed on the HEVC prediction unit (PU) grid. An exemplary original block prediction structure and the triangulated mesh are shown in Figure 5.23. When

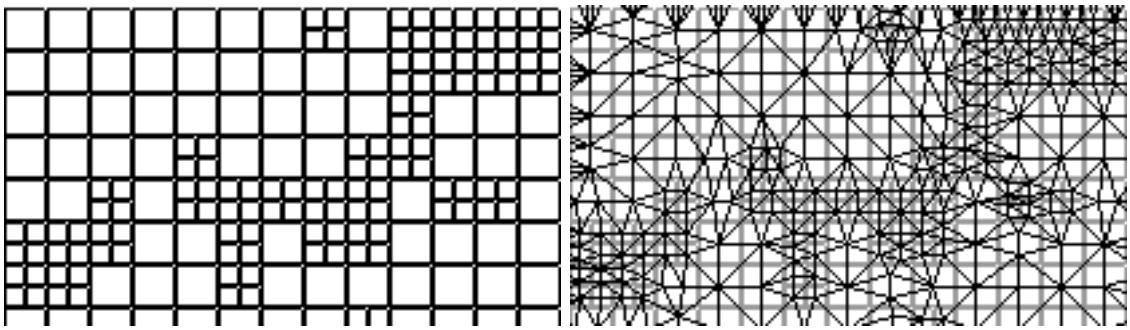


Figure 5.23: *Left:* Block prediction structure of the upper left part of frame 7 of the *BQSquare* sequence. *Right:* Resulting triangle mesh after Delaunay triangulation.

calculating the interpolated motion vector for any pixel $D = (x_D, y_D)^T$ within a frame, the following steps have to be carried out:

1. Determine the Triangle ABC within which D lies.
2. Calculate the areas spanned by the triangles ABD, ACD, and BCD. These are referred to as F_C, F_B, and F_A.
3. Calculate the interpolated MV $\vec{d}$ by weighting the MVs at the vertices of the triangle with their associated areas F_C, F_B, and F_A and rescale $\vec{d}$ according to the reference frame of its original PU.

These areas are identical to the barycentric coordinates of D. The variables used in the above algorithm are visualized in Figure 5.24. The choice of the interpolation function now depends on the underlying motion type. Linear weighting of the associated vertex vectors results in

$$\vec{d}_{\text{linear}} = \frac{\vec{a} \cdot F_A + \vec{b} \cdot F_B + \vec{c} \cdot F_C}{F_A + F_B + F_C}. \tag{5.22}$$

An example of a motion vector field produced by Equation 5.22 may be found in Figure 5.25. Here, three blocks of equal size were chosen, that originated from an area with rotational motion. In the block-based interpretation of the motion data, each pixel within one of the blocks has the motion vector associated with the center pixel of the block ($\vec{a}$, $\vec{b}$, $\vec{c}$). The linearly interpolated motion as shown in Figure 5.25

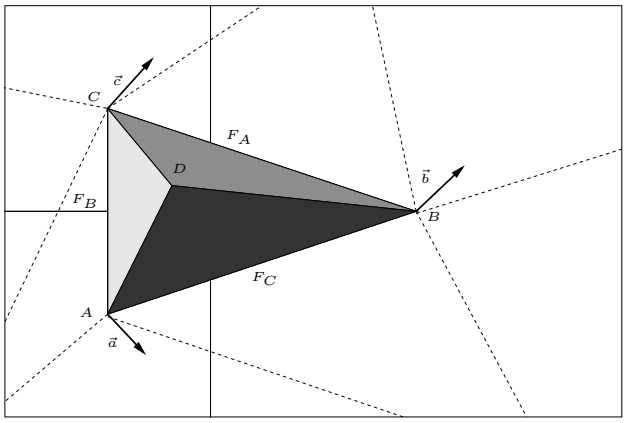


Figure 5.24: The MV at location D within the Triangle ABC is computed as a weighted mean of $\vec{a}$, $\vec{b}$, and $\vec{c}$. The subtriangles between D and the three vertices A, B, C with the areas F_A, F_B, and F_C are chosen as weights.

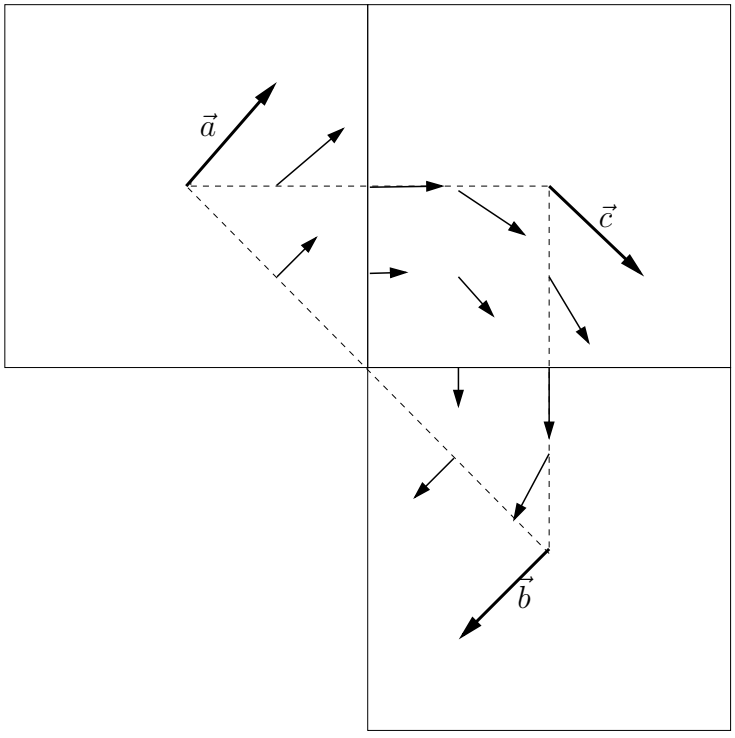


Figure 5.25: Linearly interpolated MV field between three blocks of equal size. The original motion type (rotation) is successfully reconstructed.

clearly approximates the rotation quite well. If the linear interpolation is applied to vectors at object boundaries, the quality of the interpolated MVs is, however, worse than the quality of the original motion data (see Figure 5.26 (dashed line)). If linear

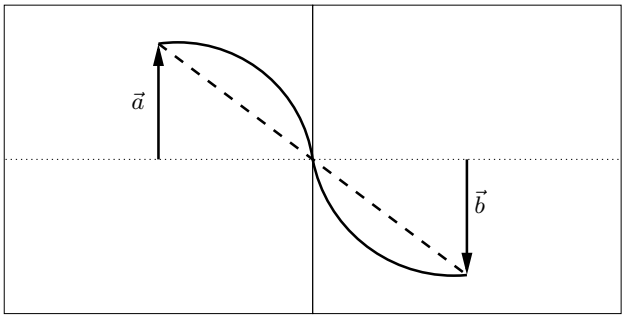


Figure 5.26: Two blocks moving in opposite directions. With linear interpolation (dashed curve) intermediate motion vectors differ significantly from the true motion. With cubic interpolation (black curve) a much sharper motion edge is preserved.

interpolation is used, the pixels close to the object edge appear to move slower than they should. In this case better results are achieved, if a cubic interpolation function is used:

$$\vec{d}_{\text{cubic}} = \frac{\vec{a} \cdot F_A^3 + \vec{b} \cdot F_B^3 + \vec{c} \cdot F_C^3}{F_A^3 + F_B^3 + F_C^3}. \tag{5.23}$$

The resulting motion distribution along the object edge may be found in Figure 5.26 (solid line). One method to determine which version of $\vec{d}$ is to be used, is to analyze the differences between the vertex MVs $\vec{a}$, $\vec{b}$, $\vec{c}$. In the current implementation, cubic interpolation is utilized whenever two vertex vectors out of $\vec{a}$, $\vec{b}$, and $\vec{c}$ only differ by at most one quarter-pel. In these cases, it is assumed that at least two vectors describe the motion of the same object. Otherwise linear interpolation is used. Should all three vectors be similar, both interpolation methods produce the same result. Examplary interpolated motion vectors may be found in Figure 5.27, where the MV distribution shows much less blockiness in the interpolated case.

5.5.3 Experimental Evaluation

The proposed method has been implemented in C++ and is employed as a preprocessing step for QTTF. It is to be noted that the interpolated MVs are only used for temporal filtering. The transmitted block-based motion field is not changed. In order to evaluate the quality of the interpolated MV field, the MPEG sequence *BlowingBubbles* was manually segmented using the Human Assisted Motion Annotation Tool [34] which subsequently produces highly accurate motion vector fields.

Sequence	Resolution	HM 8.0+QTTF		HM 8.0+QTTFi	
		Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballPass	416x240	0.00	−0.13	0.01	−0.22
BlowingBubbles	416x240	0.00	−0.06	0.01	−0.33
BQSquare	416x240	0.06	−1.74	0.07	−2.02
RaceHorses	416x240	0.00	−0.03	0.02	−0.44
BasketballDrill	832x480	0.00	0.01	0.00	−0.03
BQMall	832x480	0.00	0.04	0.00	0.01
PartyScene	832x480	0.00	−0.08	0.01	−0.20
RaceHorses	832x480	0.00	−0.09	0.00	−0.08
Vidyo3	1280x720	0.01	−0.35	0.01	−0.20
Vidyo4	1280x720	0.01	−0.14	0.00	−0.14
ParkScene	1920x1080	0.00	0.05	0.00	−0.02
Waterfall	704x480	0.01	−0.49	0.03	−0.93
average		**0.01**	**−0.17**	**0.01**	**−0.38**

Table 5.21: BD-rates and BD-PSNR for all tested sequences, both for the original QTTF implementation and for the QTTF with interpolated motion vector fields (QTTFi).

These fields will here be used as ground truth. For the tested QP range (QP 22 to 37) the interpolated MV field improves the average endpoint error per pixel by 0.7 pel. The modified QTTF with interpolation (QTTFi) has been integrated into the HEVC test model HM 8.0 and tested on the sequences shown in Table 5.21. For the given dataset using the HEVC main profile QTTF alone produced an average bit rate reduction of 0.17%. With the added MV interpolation the average BD-rate [4] is increased to 0.38%. Improvements of 0.5% are observed for those sequences where QTTF already works well. In addition, sequences with large foreground objects, like the small version of *RaceHorses*, also show good improvements. However, the new method appears to work only well for low-resolution video. For the HD-sequences even slight losses occur. The current implementation of the QTTF increases the encoder runtime by 140% on average, while the decoder complexity is only increased by 30%. The additional interpolation does not significantly change these values since very efficient and fast algorithms exist both for the computation of barycentric coordinates and for Delaunay Triangulation. Future work will focus on configuring the interpolation parameters in a manner that make the method applicable to higher resolutions as well. Preliminary results indicate, that switching between linear interpolation and raw block-based motion data may further improve the results and that the threshold for identifying similarly moving blocks should also be changed depending on the QP.

5.5.4 Summary

In this Section a novel method to interpolate dense MV fields from block-based motion data has been presented. The interpolated vectors are closer to the actual

motion of the image points in video sequences. When the new motion data is used within the QTTF in-loop filter, its performance is increased to an average bitrate reduction of 0.38% on the test dataset. Future work will include the adaptation of the filter to high-resolution sequences.

5.6 Chapter Summary

In this Chapter, publications related to temporal trajectory filtering within the decoding loop were presented. Based on the original implementation [12], the filter was extended to include B-frames as well [13]. First investigations into interactions with other in-loop filters were presented in [9]. A first implementation within the HEVC test model with extended trajectories was described in [11], where an optimal sample weighting scheme was also detailed. Further flexibility was introduced by the additional quadtree partitioning scheme [10]. Compression algorithms for which were investigated in [14]. A final improvement was presented in [15] where the compressed block-based motion fields were interpolated to produce accurate pixel-wise motion vectors. Based on results from H.264/AVC as well as HEVC test models HM 3.0 and HM 8.0, the conclusion can be drawn that the TTF performs well when used as an in-loop filter. Moreover the filter's effectiveness grows significantly when more accurate motion data is available.

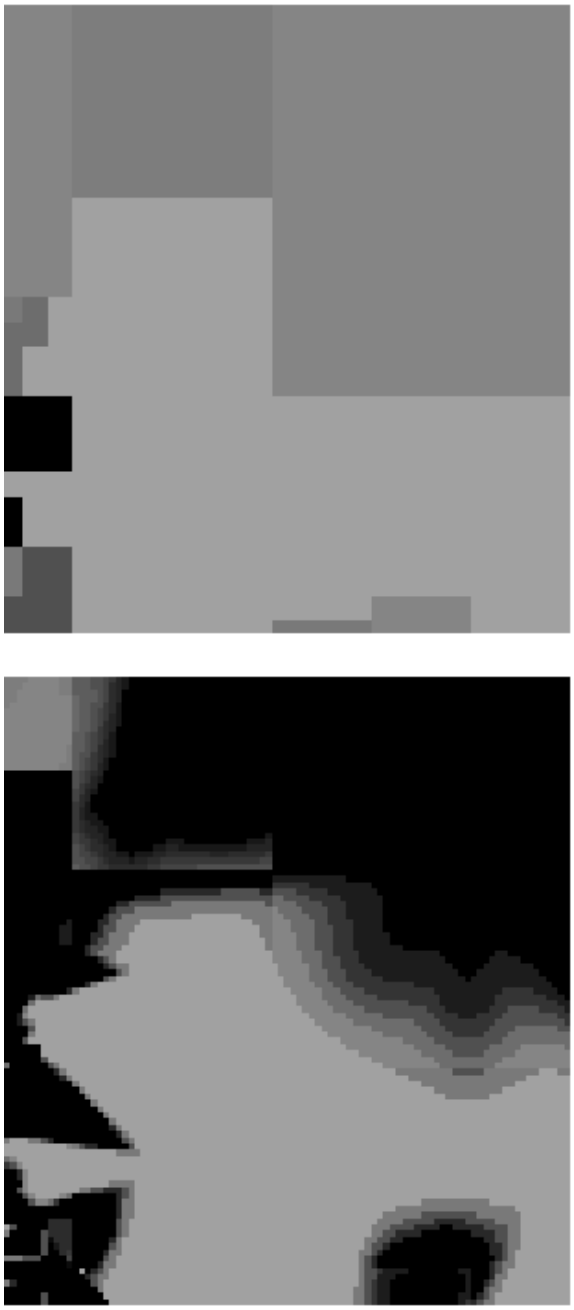


Figure 5.27: *Top:* Block-based motion vector field for reference list 0 of the upper left part of frame 7 of the BQSquare sequence. *Bottom:* Interpolated motion vector field of frame 7. A grayscale version of the x-component of each motion vector is shown.

Chapter 6

Post-Filter Implementations

We can't have full knowledge all at once. We must start by believing; then afterwards we may be led on to master the evidence for ourselves. - Thomas Aquinas

6.1 Speeding-up the Post-Filter with an Artificial Neural Network

As has been shown in Chapter 5, the TTF produced the best results in the context of the H.264/AVC baseline profile. Since the post-filter at the decoder now to be investigated can generally be expected to provide less gain than its in-loop counterpart, the baseline profile shall initially be the basis for the following investigations. In order to form longer trajectories and to make the filter more flexible, additional filter parameters shall first be introduced and examined.

6.1.1 Additional Criteria

In most application scenarios of the TTF, short to average-length trajectories will be encountered. Should a trajectory cross a shot boundary, then this is probably due to an RD-optimized decision of the encoder, whose motion vectors do not represent the motion of pixels but refer, at least in H.264/AVC, to entire macro blocks. If a motion estimation error is thus introduced, the trajectory needs to be interrupted at the respective location, since the referenced pixel does not represent a copy of the pixel to be filtered. In order to derive powerful trajectory termination criteria, the following three restrictions are again formulated:

1. The absolute luminance difference between the first pixel and the current pixel

along the trajectory should not be bigger than a threshold Th_Y.

2. The direction of the current motion vector should not differ by much from its predecessor.

3. The neighboring 8 motion vectors on 4×4 block level should also have a similar direction compared to the current motion vector.

Since no reference for quality measurement exists in a post-filtering scenario, the initially reconstructed pixel, which forms the starting point of the trajectory, is assumed to be the most reliable representation of the original pixel. The first restriction then only states that strong luminance differences imply a falsely formed trajectory. On the other hand, it is also possible that a large quantization error produced the luminance difference. In this case, it might still make sense to follow the trajectory since a large residual error does not necessarily indicate a false trajectory. How to deal with exceptionally large luminance differences will be described later.

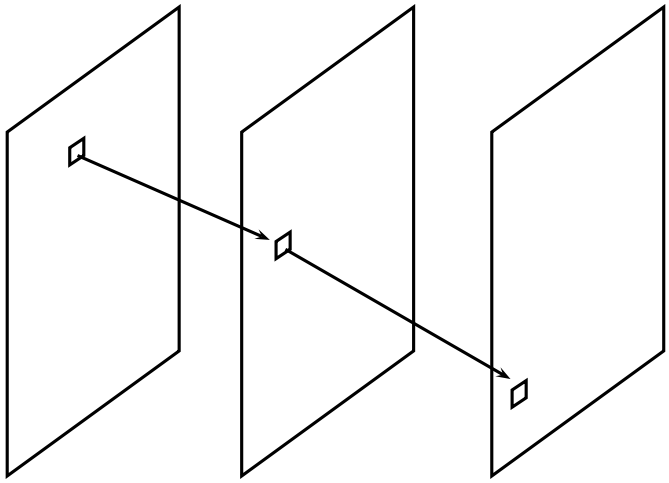
The other two criteria also need to be modified for their application in a post-filter. The second restriction states that the trajectory should not suddenly change its direction but should rather follow a linear motion principle. This assumption is validated by the utilized high frame rates resulting only in small differences between consecutive frames. Should the angle of a motion vector differ strongly from the angle of the previous motion vector then a motion estimation error has probably occured. This test is similar to the previously described motion consistency criterieon. The third restriction concerns homogeneous motion vector fields, which represent smooth and thus realistic motion. Should some of the neighboring motion vectors point in different directions, then this is again an indication for a badly predicted motion vector or for an object edge. The motion criteria now become much more important, since they represent real-world restrictions on the 2-D representation of real-world object motion. In order to test the criteria, every pixel (or trajectory node) can theoretically be associated not only with a luminance value but also with three additional values:

1. ΔY for the difference between the luminance value of the current trajectory pixel and the top-most pixel in the trajectory tree representing the pixel to be filtered

2. $\Delta \Phi_t$ for the angular difference between two temporally consecutive vectors

3. the previously used block-vote metric for the current motion vector

However, as will be shown in Chapter 7, the spatial motion consistency criterion has a very minor impact on the actual filter performance. In order to allow for a



	Y	Y_0	Y_1	Y_2
$\vec{U}$	ΔY	0	$\lvert Y_1 - Y_0 \rvert$	$\lvert Y_2 - Y_0 \rvert$
	$\Delta\Phi_t$	0	$\Phi_1 - \Phi_0$	$\Phi_2 - \Phi_1$

Figure 6.1: For each pixel along the trajectory two new parameters are added to the respective node in the trajectory tree.

quick yet exhaustive search for an optimal filter setting, only the color criteria and the temporal consistency are used here. The lower part in Figure 6.1 shows the additional criteria introduced above, which are added to the respective trajectory node. The criteria vector $\vec{U}$ contains all of the 10 above-mentioned characteristics per node. Since the root node is associated with the decoded luminance sample, its ΔY value is 0. The second parameter $\Delta\Phi_t$ is also set to zero for the root node, since it has no preceeding motion vector. As the luminance value Y_0 was chosen due to the encoder's RD-optimization as a best fit, the trajectory should not be stopped here. Otherwise, the trajectory would only include one single pixel. For the calculation of the vector entries only the luminance values and the motion information are needed. The luminance difference ΔY_i of node i is calculated as follows:

$$\Delta Y_i = |Y_i - Y_0| \tag{6.1}$$

where Y_i is the luminance value of node i. The second parameter $\Delta\Phi_{t,i}$ is calculated from the motion vector MV_{i-1} of the predecessor and the node's own motion vector MV_i

$$\Delta\Phi_{t,i} = |MV_{i-1} - MV_i|. \tag{6.2}$$

6.1.2 Filtering

The emphasis of the filtering operation lies on the averaging process along the trajectory. Nevertheless, the additional criteria also need to be checked with respect to their thresholds. The filter parameters are subsequently, as before, the transmitted thresholds. In the simplest and safest case the thresholds are determined through a *brute-force* algorithm. In this manner an optimal performance of the filter is guaranteed. However, other approaches, such as a threshold search with an artificial neural network (ANN) are also possible. Both methods have been implemented as a post-filter for the H.264/AVC baseline profile and are described below.

Brute-Force Implementation

A *brute-force* search is used whenever an arbitrary unknown function defined on a finite input space is given and an optimum of the function needs to be found. There now exist two variables ΔY, $\Delta\Phi_t$ with a search range in the intervall of $[0, 255]$, both of which can be chosen independently. This results in a total of $255^2 \approx 65 \cdot 10^3$ parameter combinations for the TTF.

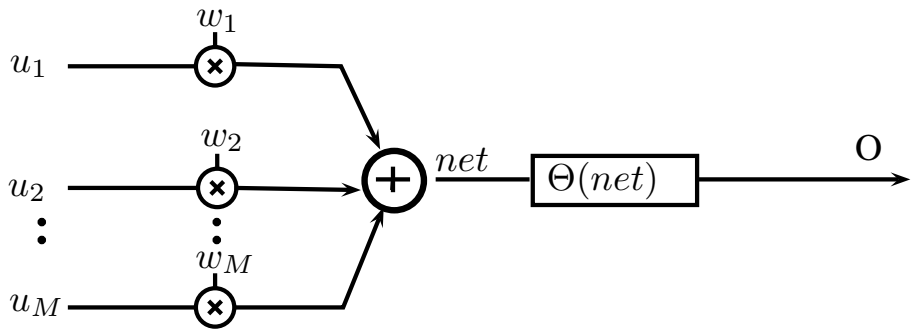


Figure 6.2: Structure of a general decision neuron. The input values u_i are first multiplied with the weights w_i and their sum net is calculated. The decision is then based on net. The output value O is directly computed by the application of the *sigmoid function* to net: $O = \Theta(net)$.

Artificial Neural Network

The thresholds described above constitute a simple possibility to control the performance of the TTF and can easily be RD-optimized. Since a post-filter is investigated in this section, no restrictions need to be placed on side-information bit rate or RD-optimization as long as the filter works in an unsupervised manner. Subsequently a much larger number of combinations could be tested. With a growing number of combinations, however, the computional complexity is also increased and an alternative way to compute the optimal threshold combination is required. To this end an artificial neural network (ANN) is chosen here, since it can, if it is well configured, quickly solve such problems. One of the biggest advantages of this approach is that the network can be trained on a certain dataset with the ability to later perform adaptive filtering.

In addition, no hard decisions are required for an ANN. Instead of terminating a trajectory or excluding certain pixels from the averaging process, the ANN can assign every luminance value an independent weight. In this context it is essentially irrelevant which criteria are provided to the network and in which manner they contribute to the solution. The *brute-force* method described above requires knowledge about the input parameters and their influence on the filter's performance. A neural network can construct an efficient filter structure from a set of input variables even if their direct influence on the filter result is unknown. Essentially, the ANN will select its own filter structure and can even cope with additional input parameters. It will also automatically select the optimal weights for all available input values. This would enable the filter to even detect temporal and spatial interdependencies that have not yet been considered during the TTF design process. To this end the

basic principle and functionality of a neural network are now described.

A neural network reflects the structure of a biological nerve system and is thus made up of small (and rather simple) processing units which are interconnected by weighted connections. With reference to the biological original the processing units are called neurons and use several input values to produce an output value that describes the status of each neuron (see Figure 6.2). An output value of 0 represents an inactive neuron, the value 1 signifies an active neuron. The first processing step in a neuron is restricted to a weighted sum of the input values u_i resulting in an internal value *net*.

$$net = \sum_{i=1}^{M} w_i \cdot u_i \tag{6.3}$$

The subsequent activation function $A(net)$ evaluates *net* and returns the status of the neuron. In the simplest case the activation function corresponds to the *Heaviside step function* which returns 0 unless a certain threshold is reached, after which the function returns 1. As will be shown later, it is generally preferable to use a differentiable activation function that only approximates the *Heaviside step function*. One such function is the *sigmoid function*

$$\Theta(net) = \frac{1}{1 + e^{-net}}, \tag{6.4}$$

a plot of which may be found in Figure 6.3. With an additional parameter β the steepness of the slope of the function can be manipulated. For the modified *sigmoid function*

$$\Theta(net) = \frac{1}{1 + e^{-\frac{net}{\beta}}} \tag{6.5}$$

a higher value of $\beta > 0$ produces a very slow rise. With a decreasing β the *Heaviside step function* is approximated. For each neuron such an activation function needs to be specified. In order to simplify the following descriptions, a simplified neuron structure is used (see Figure 6.4). Here the previously used *sigmoid function* Θ is replaced by a general activation function A. With the help of A a neuron can now also perform more complex operations like multiplications or additions. The weights w_i associated with the components u_i of the input vector $\vec{U}$ shall be given in the upper half of the graphical representation while the activation function shall always be displayed in the lower half, see Figure 6.4. Analytically, the computation of a neuron's output is now given by

$$O = A\left(\vec{U} \cdot \vec{W}\right), \tag{6.6}$$

where the dot represents the scalar product.

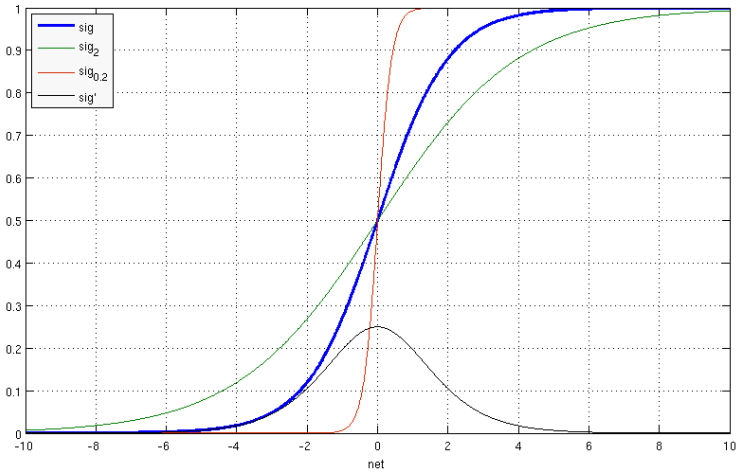


Figure 6.3: Shape of the sigmoid function $\Theta(net)$ and two modified versions with $\beta = 2$ and $\beta = 0.2$ as well as the derivative Θ'.

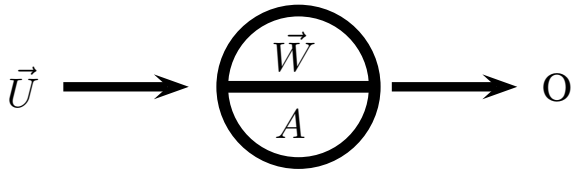


Figure 6.4: A simplified neuron representation with an input vector $\vec{U}$ whose scalar product with a weighting vector $\vec{W}$ is calculated and then processed by the activation function A.

6.1.3 TTF as an Artificial Neural Network

In order to construct an artificial neural network with the functionality of the TTF, the individual neurons need to be connected in a web structure. Afterwards appropriate weights need to be assigned to each neuron. For simplification the neural network will be devided into different layers of neurons.

Depending on the direction of a link a network can either be classified as forward-oriented or backward-oriented. Forward-oriented neural networks only consist of connections to the following layer. Backward-oriented networks equivalently only show connections to the previous layer. Feedback loops in backward-oriented networks can produce undesired side-effects such as oscillation and thus require more complex implementations.

For simplification the TTF will be implemented as a forward-oriented network with a variable number of layers. In order to adapt the network to the properties of the TTF, the application of the filter to one single trajectory will now be examined. The filtering process is devided into two separate parts that are reflected by the neural network:

- The first part processes the available criteria and decides whether the associated luminance value will be selected for inclusion in the filtering process. However, this only results in a soft decision and each luminance value is assigned a weight between 0 and 1.

- The second part represents the calculation of the filtered luminance value through a weighted sum of the available luminance values.

Figure 6.5 shows the resulting structure consisting of the two different parts. Whether a certain luminance value is included in the filtering process or not, is decided in the *internal network*. This network makes its decision based on its structure and on the input values. The neurons of the last layer (the norm neuron and the value neuron) are responsible for the averaging operation, where the norm neuron executes the summation of all *internal network* output values.

As the output of each internal network always takes on a value between 0 and 1, the sum of these outputs is a lower bound to the number of selected luminance values. The value neuron receives the luminance vector $\vec{Y} = [Y_0, Y_1, Y_2, ..., Y_N]^T$ describing the trajectory of length N and weights them with the associated output values of the N *internal networks*. Since $A = net$ for the value neuron, it only adds up its individual inputs multiplied with the given vector of luminance values and

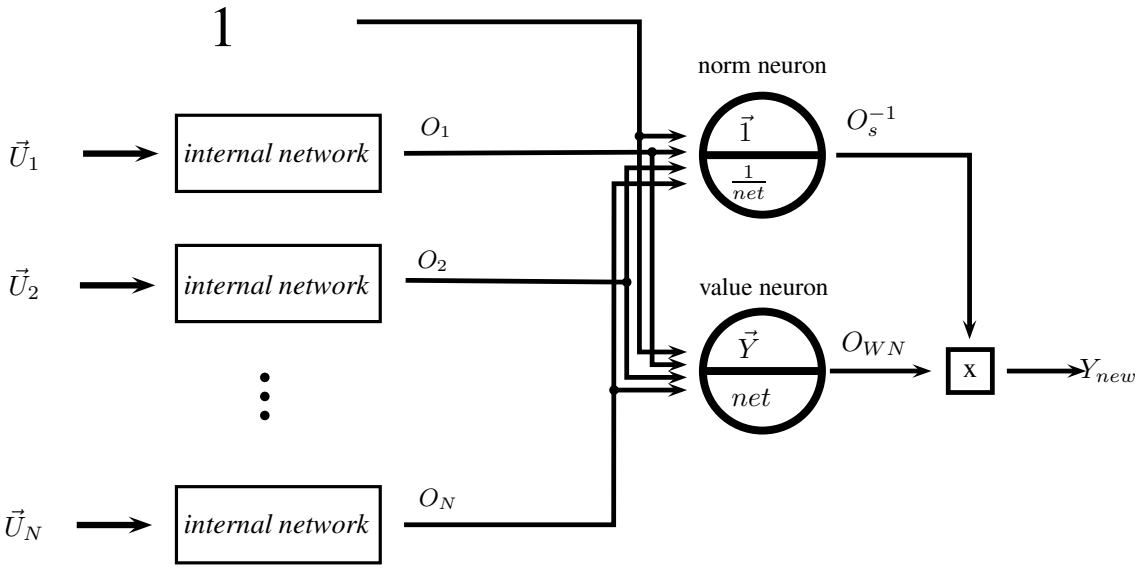


Figure 6.5: General network structure for the filtering of a trajectory. For each node (trajectory pixel) and input vector $\vec{U}_k$ is evaluated by the *internal network*. The resulting output $0 \leq O_k \leq 1$ signifies how much influence the respective luminance value should have on the filtered luminance value Y_{new}.

passes them along

$$O_{WN} = \sum_{i=1}^{N} O_i \cdot Y_i. \tag{6.7}$$

The summation of these weighted decision values corresponds to the averaging process of the TTF. This also constitutes the biggest difference between the *brute-force* solution and the ANN approach: No matter what the state of each *internal network* is, the associated pixel will always be included in the averaging process. However, each pixel is now treated differently depending on its semantic context. For instance, the filter could now decide only to use those parts of a trajectory where the estimated motion is very small.

The norm neuron with an activation function $A = \frac{1}{net}$ computes the inverse of the sum of all individual luminance weights

$$O_s^{-1} = \frac{1}{\sum_{i=1}^{N} O_i}. \tag{6.8}$$

The processed and filtered luminance value is finally calculated by multiplying the output of the value neuron O_{WN} with the norm neuron's output O_s^{-1}.

$$Y_{new} = O_{WN} \cdot O_s^{-1} = \frac{\sum_{i=1}^{N} O_i \cdot Y_i}{\sum_{i=1}^{N} O_i}. \tag{6.9}$$

These two neurons may be found on the right-hand side of Figure 6.5. In this manner the filtering along a trajectory is easily represented by the neural network. The underlying structure of the *internal network*, which is identical for each trajectory node, now has to be defined.

6.1.4 The Internal Network

Since the general structure of the neural network is roughly layed out by the averaging process, an adaptation within the *internal networks* remains. The *internal network* reflects the location where all information from a trajectory node is used for evaluation and whose output controls the selection of certain input values for filtering. At the same time, the *internal network* is responsible for most of the computational complexity. This means that its structure has a significant impact on the total algorithm runtime. Consequently, the following restrictions are placed on the *internal network* structure:

1. Each *internal network* has exactly n input values.
2. Each *internal network* has exactly one output value.
3. The structure of the *internal network* is constituted by neurons with sigmoid activation functions.

In this manner the *internal network* can be treated as an independent neural network that processes an input data set ($\vec{U}_k$) into a decision value O_k. In principle, the structure of the *internal network* could be chosen arbitrarily (see Figure 6.6). Several examples of *internal network* realizations will now discussed.

Luminance Filtering

If the *internal network* has the structure shown in Figure 6.7, only the luminance difference is used for decision making. The *internal network* consists of a single neuron for each trajectory point and determines a threshold for ΔY after successful training. In order for a certain luminance value to be included in the filtering, it has to pass that threshold first. With the aim of evaluating the results of the filtering

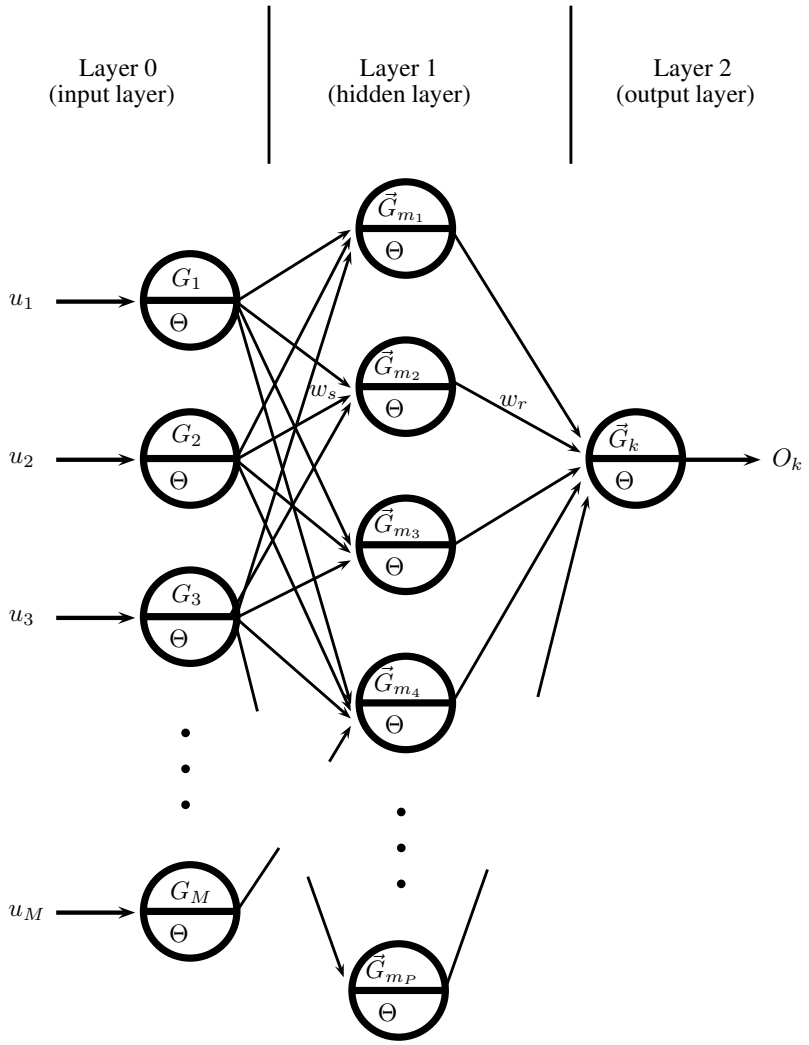


Figure 6.6: An example of a three-layered *internal network*. The vectors $\vec{G}_i$ represent arbitrary weights. The output of neuron N_{m_2} shall be weighted by $w_r \in \vec{G}_k$ and w_s shall link neurons N_2 and N_{m2}

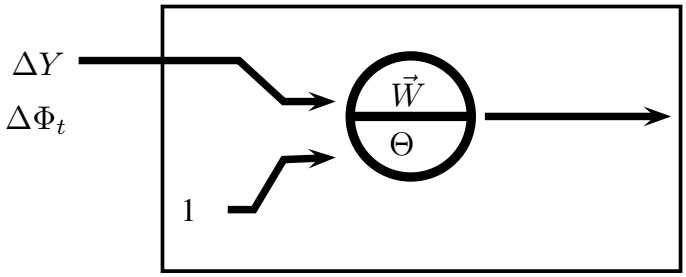


Figure 6.7: Structure of an *internal network* for a simple ΔY threshold calculation. The weight vector $\vec{W}$ consists of two valuesEach neuron has one constant input value which multiplied by its weight constitutes the neuron threshold.

process with an artificial neural network, the mathematical form of the utilized *internal network* is now analyzed:

$$O_i = \Theta(net_i) = \Theta\left(\frac{\Delta Y \cdot w_0 + 1 \cdot w_1}{\beta}\right), \text{ with } net_i = \frac{\Delta Y \cdot w_0 + 1 \cdot w_1}{\beta}. \tag{6.10}$$

$\Theta(net_i)$ has a point of inflection at $net_i = 0$. The corresponding ΔY value shall be referred to as the Θ-threshold Th_Θ:

$$net_i = \frac{\Delta Y \cdot w_0 + 1 \cdot w_1}{\beta} = 0 \Leftrightarrow Th_\Theta := \Delta_Y|_{net_i=0} = -\frac{w_1}{w_0} \tag{6.11}$$

Depending on the choice of β there exists either a transitional interval between the two zones of saturation or the function approximates the *Heaviside function*. In order to describe the width of the transitional interval, the Th_{-3} measure is now defined. It represents the distance from Th_Θ to the point where $\Theta(net_i) = 10^{-3}$. Based on the two values Th_Θ and Th_{-3} predictions can be made concerning filter behavior and the optimal weights. To this end both are formally described by

$$Th_\Theta = -\frac{w_1}{w_0} \tag{6.12}$$

and

$$\begin{aligned} \Theta(Th_{-3} + Th_\Theta) = \Theta(Th_{-3} - \frac{w_1}{w_0}) &= \frac{1}{1 + e^{-\frac{Th_{-3} - \frac{w_1}{w_0}}{\beta}}} \stackrel{!}{=} 10^{-3} \\ 1 + e^{-\frac{Th_{-3} - \frac{w_1}{w_0}}{\beta}} &= 10^3 \\ -\frac{Th_{-3} - \frac{w1}{w0}}{\beta} &= \ln(10^3 - 1) \end{aligned} \tag{6.13}$$

From these the following analytical term is derived:

$$Th_{-3} = \frac{-\beta \cdot \ln(10^3 - 1) - w_1}{w_0} - \frac{w_1}{w_0} = -\beta \cdot \ln(999)\frac{1}{w_0}. \tag{6.14}$$

The Θ-threshold Th_Θ can be seen as a decision threshold which corresponds to the threshold from the *brute-force* filtering algorithm. In the ideal case of a *Heaviside function* with an infinitely small transitional interval ($Th_{-3} = 0$) both algorithms would be identical. When examining Equation 6.11 it becomes obvious that despite the three independent variables β, w_0, and w_1, there only exists two degrees of freedom. Common multiples of the three variables would always result in identical behavior of the *sigmoid function*-

From Equation 6.11 it follows that the absolute value of the Θ-threshold is the ratio of the absolute values of w_0 and w_1. The signs of both parameters influence the output of the *sigmoid function* with respect to ΔY. Figure 6.8 shows the four possible different parameter combinations of w_0 and w_1, where the absolute value of each parameter is fixed to 1 and only the signs change. Here the Θ-threshold also has a value of 1 resulting in a different curvature per combination.

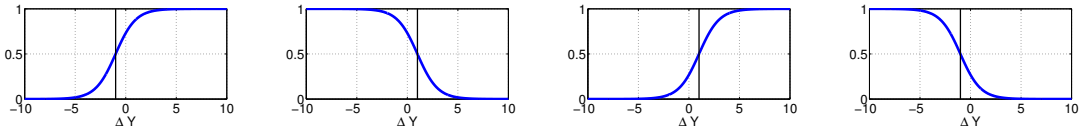
Figure 6.8(a) shows the *sigmoid function* for two positive parameters. In this case, all values (which are non-negative absolute differences) lie above the Θ-threshold and the neuron will always be activated. As the Θ-threshold is negative, every luminance value would be selected. Changing the absolute value of Θ would have no influence on the filter performance.

Figure 6.8(b) shows a different curvature where all ΔY values above a positive threshold should deactivate the neuron. This constitutes the relevant case. In an analogy, to the *brute-force* algorithm only values below a certain threshold are selected and are then processed by the remaining network structure. All values above the threshold are discarded. The remaining two cases are again of no practical use.

In Figure 6.8(c) all values above the threshold would be selected while small differences are ignored. Figure 6.8(d) shows a case with a negative threshold which, in presence of a small transitional zone, would constantly deactivate the neuron.

Temporally Consistant Filtering

A second possible implementation is temporal filtering without any color information, see Figure 6.9 for details. This works exactly in the same manner as the luminance filtering version but with $\Delta\Phi_t$ subject to a threshold. This corresponds to the search for a maximum allowed deviation from the temporal consistency of



(a) $w_0 = 1$, $w_1 = 1$ (b) $w_0 = -1$, $w_1 = 1$ (c) $w_0 = 1$, $w_1 = -1$ (d) $w_0 = -1$, $w_1 = -1$

Figure 6.8: Dependency between the neuron output and the sign of the individual parameters w_0 and w_1.

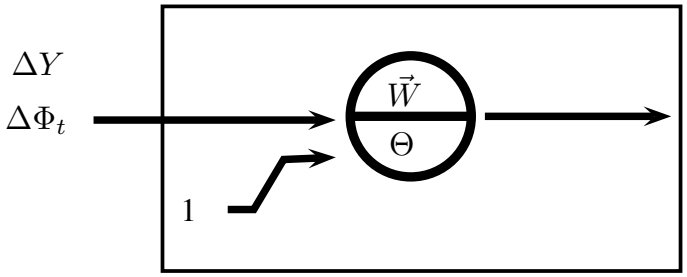


Figure 6.9: Structure of an *internal network* for a simple $\Delta\Phi_t$ threshold calculation. Each neuron has one constant input value which multiplied by its weight constitutes the neuron threshold.

the trajectory. However, it does not take into account the most prominent source of information, namely the luminance difference. As will be shown in Chapter 7.2 disabling the color threshold generally produces very poor results. Thus a combination of both thresholds is much more promising.

6.1.5 A Combination of Luminance Filtering and Temporal Consistency Filtering

Another possible network for filtering based on a combination of two input values is shown in Figure 6.10. There, three neurons are used to combine luminance difference and temporal angular difference. The two neurons of the first layer produce a rough classification of the respective input value. Here, too, two thresholds and the steepness of the *sigmoid function* can be specified. The result of this evaluation is handed over to the output neuron in the form of a value in the interval $[0, 1]$. The weights of this neuron represent the influence of the temporal and luminance differences. A large weight here reflects a strong influence of the associated criterion while a small weight indicates a much weaker correlation. The additional constant input to the output neuron again produces a neuron threshold which both input values have to surpass in order to force the output to 1. In this context the weight

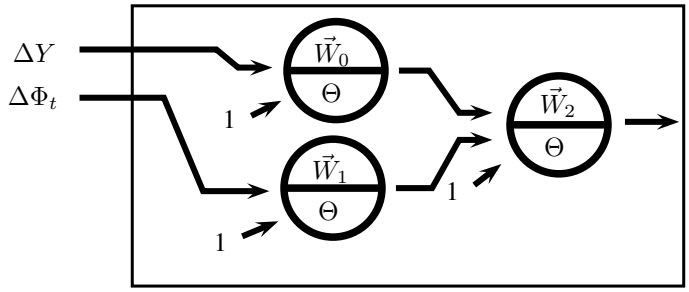


Figure 6.10: Structure of an *internal network* for a combination of ΔY and $\Delta\Phi_t$ threshold calculation. Each neuron has one constant input value which multiplied by its weight constitutes the neuron threshold.

vectors $\vec{W}_0$ and $\vec{W}_1$ each have two components whereas $\vec{W}_2$ consists of another three independent weights. The entire *internal netwok* is, therefore, controlled by a total of seven parameters (w_0 to w_6).

6.1.6 Other Possible Implementations

In principle, a whole set of other implementations is possible as well. Also, the number of input values could be arbitrarily increased if the network structure is also adapted. The previously described implementations are relatively simple and allow for an intuitive interpretation of the threshold values. Based on the general theory of artificial neural networks other forms of neurons, structures with feedback loops, and structures that evaluate input signals more than once are also possible. Especially reevaluating an input value several times allows the network to escape the boundary of linear seperability of individual neurons. The more neurons are available per layer the more easily the network adapts itself to changing input data. However, utilizing too many neurons could possibly overtrain the network which would restrict its general applicability. In addition, each new neuron also increases the computational load. Since the operations of an *internal network* need to be executed for each node of a trajectory, the runtime can increase very quickly. With today's multicore architectures and as each trajectory can be processed individually, such a load could be easily shared between several cores.

6.1.7 The Learning Algorithm

The backpropagation algorithm is a very common way of determining the weights of a forward-oriented neural network. The basic idea is the calculation of individual weights by differentiating the output of the network with respect to a weight. This

method may be chosen since the networks in Figures 6.5 and 6.6 only show linear dependencies (except for the sigmoid function). Consequently, partial derivatives can be calculated for each weight and a gradient-based algorithm can be used to determine the optimal configuration. In order to do so, a neural network can be defined in the following manner:

$$Y_{\text{new}} = \frac{\text{sum of weighted luminance values}}{\text{number of selected luminance values}} \tag{6.15}$$

equivalently

$$\begin{aligned} Y_{new} &= \frac{O_{WN}}{O_s} \\ O_{WN} &= Y_0 + \sum_{k=1}^{N} Y_k \cdot O_k \\ O_s &= 1 + \sum_{k=1}^{N} O_k. \end{aligned} \tag{6.16}$$

The constants (Y_0 and 1) in front of the sums ensure that the decoded luminance value ($Y_0 = Y_{\text{dec}}$) is not discarded and that a division by zero is avoided. The method described here is, of course, not limited to the TTF implementation and is commonly used when neural networks are used for optimization.

For any trajectory the partial derivatives with respect to a weight w_i are

$$\begin{aligned} \frac{\partial Y_{new}}{\partial w_i} &= \frac{\partial O_{WN}}{\partial w_i} \cdot \frac{1}{O_s} - \frac{O_{WN}}{{O_s}^2} \cdot \frac{\partial O_s}{\partial w_i} \\ \frac{\partial O_{WN}}{\partial w_i} &= \sum_{k=1}^{N} Y_k \cdot \frac{\partial O_k}{\partial w_i} \\ \frac{\partial O_s}{\partial w_i} &= \sum_{k=1}^{N} \frac{\partial O_k}{\partial w_i}. \end{aligned} \tag{6.17}$$

(6.18)

With these a new expression for $\partial Y_{new}/\partial w_i$ can be found:

$$\begin{aligned} \frac{\partial Y_{new}}{\partial w_i} &= \frac{1}{O_s} \cdot \sum_{k=1}^{N} Y_k \cdot \frac{\partial O_k}{\partial w_i} - \frac{O_{WN}}{{O_s}^2} \cdot \sum_{k=1}^{N} \frac{\partial O_k}{\partial w_i} \\ \frac{\partial Y_{new}}{\partial w_i} &= \frac{1}{O_s} \cdot \sum_{k=1}^{N} \left(Y_k - \frac{O_{WN}}{O_s} \right) \frac{\partial O_k}{\partial w_i} = \frac{1}{O_s} \cdot \sum_{k=1}^{N} \left(Y_k - Y_{new} \right) \frac{\partial O_k}{\partial w_i} \end{aligned} \tag{6.19}$$

Finally, only the derivative $\frac{\partial O_k}{\partial w_i}$, which is derived from the output of the *internal network*, remains. In order to calculate the derivative, the structure from Figure 6.6 is used. The basic idea of backpropagation is now explained for an arbitrary weight w_i. As mentioned above, all *internal networks* shall use the same set of weight parameters w_1 to w_M. Therefore, the output O_k of neuron N_k is given by

$$O_k = \Theta(net_k) = \frac{1}{1 + e^{-\beta^{-1} \cdot net_k}} \text{ , with } net_k = \sum_{i=0}^{P} w_i \cdot u_{i,k}. \tag{6.20}$$

Where w_i is the i-th weight, $u_{i,k}$ is the i-th input value of neuron N_k. As an activation function $\Theta(net_k)$ the sigmoid function was chosen here. The advantage of this function lies in its derivative

$$\frac{\partial \Theta(net_k)}{\partial net_k} = -\frac{1}{\beta}\Theta(net_k) \cdot (\Theta(net_k) - 1)\,. \tag{6.21}$$

This means that the derivative at location net_k can be calculated directly from the previously determined value at location net_k itself. This greatly reduces the computational complexity since only a multplication and a subtraction need to be carried out. For the dependency of a neuron output with respect to the individual weights the following relation can be found

$$\frac{\partial O}{\partial w_i} = \frac{\partial \Theta(net_k)}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_i} = -\Theta(net_k) \cdot (\Theta(net_k) - 1) \cdot u_{i,k} \tag{6.22}$$

This relationship only applies to the last neuron in an *internal network*. For all other weights calculating the derivative requires application of the chain-rule. If w_s links neurons N_2 and N_{m2}, w_s links neurons N_2 and N_{m2} and P is the number of nodes in the hidden layer (see Figure 6.6) then

$$\frac{\partial O_k}{\partial w_s} = \frac{\partial O_k}{\partial net_k} \cdot \sum_{l=1}^{P} \left(w_l \cdot \frac{\partial u_{l,k}}{\partial w_s} \right) = \frac{\partial O_k}{\partial net_k} \cdot w_r \cdot \frac{\partial O_{m2}}{\partial w_s}. \tag{6.23}$$

It is to be noted that only O_{m2} depends on w_s, whereas all other input values depend on different weights. With the simplified notation $\Theta(net_k) = \Theta_k$ this can be rewritten as

$$\frac{\partial O_k}{\partial w_s} = \frac{1}{\beta^2}\Theta_k \cdot (\Theta_k - 1) \cdot w_r \cdot \Theta_{m2} \cdot (\Theta_{m2} - 1) \cdot O_1, \tag{6.24}$$

where O_1 is introduced since it is a constant input value to the neuron m_2 from the hidden layer and thus the inner derivative of $\frac{\partial O_{m2}}{\partial w_s}$ becomes $\frac{\partial \Theta_{m2}}{\partial w_s} = O_1$. The above equation, as well, only contains additions and multiplications since the Θ-values have all previously been calculated during the update-step. For all other weights the procedure is similar. The backpropagation algorithm now consists of the following steps of determining a derivative $\frac{\partial O_k}{\partial w_i}$:

1. Calculate $\rho_k = \Theta'_k$ for the last neuron.
2. Traverse all layers from back to front.
3. Calculate $\rho_j = \Theta'_j \cdot \sum_q \rho_q \cdot w_r$ for each neuron in a layer.

Where q as an index adresses all output values of neuron j and w_q is the weight between neuron j and its successor neuron q. With ρ_i the derivative can be calculated

$$\frac{\partial O_k}{\partial w_i} = \rho_i \cdot u_{i,k}. \tag{6.25}$$

u_i is the input value connected to the weight w_i. If the MSE is used as an error measure E, its partial derivative is

$$\begin{aligned} E &= (Y_{new} - Y_{Orig})^2 \\ \frac{\partial E}{\partial w_i} &= \frac{\partial E}{\partial Y_{new}} \cdot \frac{\partial Y_{new}}{\partial w_i} \\ \frac{\partial E}{\partial w_i} &= 2\,(Y_{new} - Y_{Orig}) \cdot \frac{\partial Y_{new}}{\partial w_i} \end{aligned} \tag{6.26}$$

The resulting gradient ∇E points, of course, in the direction of the steepest ascent. However, since the aim of this step is finding the minimum of the error function, the opposite direction is chosen.

$$w_i^* = w_i^\dagger - \frac{\partial E}{\partial w_i} \tag{6.27}$$

where w_i^* is the new weight and $w_i^\dagger$ is the value of the same weight from the previous calculation step. All descriptions so far have only focused on one single trajectory. The optimization of the network, however, is conducted for the entire network on frame level. Subsequently, the total error is given by

$$E = \sum_{t=0}^{N} E_t = \sum_{t=0}^{W \cdot H} (Y_{new,t} - Y_{Orig,t})^2. \tag{6.28}$$

Where W is the width and H is the height of the frame. $W \cdot H$ then corresponds to the total number of available trajectories. The partial derivatives are modified in the same way:

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \sum_{t=0}^{W \cdot H} \frac{\partial E_t}{\partial Y_{new,t}} \cdot \frac{\partial Y_{new,t}}{\partial w_i} \\ \frac{\partial E}{\partial w_i} &= 2 \sum_{t=0}^{W \cdot H} (Y_{new,t} - Y_{Orig,t}) \cdot \frac{\partial Y_{new,t}}{\partial w_i}. \end{aligned} \tag{6.29}$$

iRprop+

The actual algorithm is based on an improved version of the "resilient back-propagation algorithm" (Rprop) as described in [23]. In addition to the description in [23] boundaries for the weights are introduced inside of which the optimal weight is to be searched for. The two values UpperBound and LowerBound define the maximum and minimum value for each weight.

A second boundary condition describes a temporal aspect: A moving bound is defined that increases with the number of iterations of the iRprop+. It ensures that small weights are selected during the early phase of the algorithm. Only if no fitting configuration has been found after several cycles, larger weights are considered as well. A detailed description of all parameters of the algorithm may be found in Appendix B.

Simulated Annealing Rprop (SARprop)

As the iRprop+ algorithm showed difficulties to find a minimum in some test cases, the Rprop was extended with the ability to perform Simulated Annealing [47]. The term comes from the area of thermodynamics where particles of strongly heated material are brought into an optimal mesh structure through gradual annealing. The optimal structure corresponds to a global energy minimum. During the annealing process, particles in a minimal energy state can switch to a worse configuration in order to later reach the global minimum. The simulated annealing algorithm here treats the weights of the network as these particles. This, subsequently, means that the algorithm will not terminate once a minimum of the error function E has been found. Instead, random modifications of individual weights are allowed to search for an even better minimum. The amplitude of the random fluctuations is controlled by a "temperature" parameter T, which is decreased step by step. One disadvantage of this process is the increased runtime the algorithm requires to find a minimum. However, the runtime can be controlled by the choice of T. The respective parameters and their influence are described in Appendix B. Nevertheless, no guarantee can be given that a global minimum is found with the SARprop method.

6.2 A Reference-Free Post-Filtering Approach

The main purpose of a reference-free post-filtering approach is to develop a post-filter or a post-filter configuration that operates in such a way as to allow its application without the transmission of additional filter parameters. With this setup the filter can then be utilized outside the strict protocol of the codec that generates the compressed bit stream. Moreover, such a filter does not necessarily need to be linked to a compression scheme at all. Instead a temporal filter could also be used to improve the visual quality of noisy uncompressed video sequences. The basic structure of the filter investigated here remains identical to the one described in Section 6.1. However, the generation of the filter control parameters can now no longer be based on data available only at the encoder. Therefore, the filter needs to adapt itself to the limited information available at the decoder side. Before describing the reference-free filter optimization, a number of definitions need to be made: For any given frame or block the *training data set* shall contain all the pixels which the filter uses to adapt itself to. The *filter data set* (see Figure 6.11) contains all pixels to be filtered and their trajectories. Thus the filter no longer operates on the entire frame but is trained on a small number of pixels. The optimized filter is then applied to the remaining image or block content. In Section 5.1 *filter*

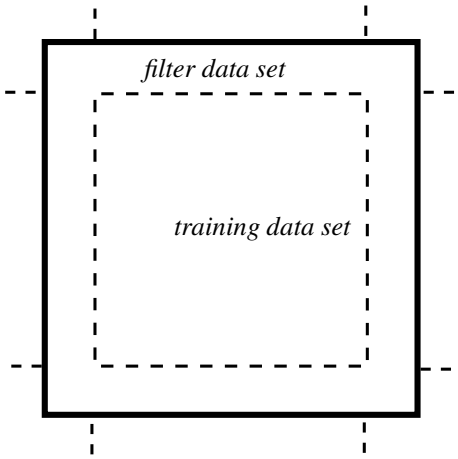


Figure 6.11: A macroblock containing 16×16 pixels is devided into a *training data set* consisting of all boundary pixels within a certain distance and the *filter data set* comprising all other pixels of the block.

data set and *training data set* were identical since the filter was optimized on the entire frame. However, the method required the transmission of filter parameters as side-information. If no side-information is transmitted and both optimization and filtering are done in the same frame and if the *training data set* and the *filter data set* are disjunct sets, then the following conclusion can be drawn: The smaller the *filter*

data set, the smaller the achievable gain. A first possibility to conquer this problem would be to use fixed universally applicable filter parameters for all video sequences, i.e. a *training data set* of size zero. In this case, all pixels from each frame could still be filtered. Such a filter would, however, not take into account the content and temporal characteristics of different sequences. Even for the short sequences used here, temporal adaptability has been the greates strength of the TTF. Based on the experiments described in the first part of Chapter 5, it is safe to assume that no such filter configuration exists. In order to provide good results for a large variety of sequences, it would be possible to optimize the filter parameters per sequence. In the case of a post-filter, this would require the receiver to buffer the entire sequence before displaying it. Because of this disadvantage a frame-based optimization is used instead. With the aim to devide a frame into *training data set* and *filter data set* certain rules are required. This separation scheme is now investigated in detail.

6.2.1 Local Separation with *Brute-Force* Filtering

A local separation scheme shall use the properties of motion estimation (ME) and motion compensation (MC) to separate well-predicted and badly-predicted spatial areas from each other. ME tends to choose larger blocks (16×16 in H.264/AVC) so that the RD-cost for transmitting such a block becomes smaller. It is now assumed that pixels at the center of such a block generally correspond better to the original frame than those at the block boundary. In order to support this theory, 250 frames from each of the videos listed in Table 6.1 are now analyzed. These were chosen since they reflect the spectrum of MPEG test sequences quite well and also, because the set includes both sequences that are and sequences that are not suitable for temporal filtering.

Ideally, the reference-free post-filter should, of course, never decrease the quality of a decoded video. To this end the original sequences were encoded with the H.264/AVC

Sequence	fps Hz	width	height
BasketballDrive	50	1920	1080
BlowingBubbles	50	416	240
BQMall	60	832	480
BQSquare	60	416	240
BQTerrace	60	1920	1080
RaceHorses	30	416	240
Waterfall	25	704	480

Table 6.1: Sequences analyzed concerning the residual distribution.

reference software JM 16 KTA 2.4 [44] and the encoder settings were modified so that the ME algorithm only uses 16×16 macroblocks. The selected H.264/AVC profile is baseline with an IDE level of 4.0. The encoded sequences are, of course, noisy and thus allow for an examination of the error distribution on 16×16 block level. However, especially when close to the frame boundary, certain trajectories point to areas outside of the frame. In order to suppress effects introduced by inaccurate ME, all pixels within a 16 pixel radius around the frame boundary are not examined. Motion vectors in this area are ignored for all subsequent post-filtering applications. For reasons of comparability the same four QP parameters (22 to 37) as in all other experiments are used.

6.2.2 16×16 only, Motion Compensated for the Brute-Force Filter

A first analysis' objective is to evaluate where in a 16×16 motion compensated block pixels occur that are identical to the original frame. These would signify locations where the MC is generally succesful in describing the true pixel-wise motion. Based on the assumption that motion changes only slowly from frame to frame and that foreground objects will distort the block boundary first, motion compensated pixels with a zero residual are expected to occur mostly in the macroblock center.

Zero-Errors

As can be seen in Figure 6.12 this is indeed the case for three out of six sequences (*BlowingBubbles*, *BQMall* and *RaceHorses*). The sequences *BasketballDrive* and *BQTerrace* are the only tested HD-sequences and show a significantly different distribution. In the case of *BQTerrace* a pattern of vertical stripes appears, which correspond to columns of pixels with an above-average number of small errors. In order to explain this anomaly, Figure 6.13 shows the average Y-component distribution per 8×8-block of the uncompressed first frame of the *BQTerrace* sequence. Here, too, the vertical stripes are apparent. These are, in fact, not caused by the codec and its resulting artifacts, but are a property of the original sequence.

Apparently, the photo chip of the camera the video was captured with produced a small offset for each even column of pixels. When MC is conducted the overall texture of the sequence will, of course, be the dominant feature and the slight vertical lines will be ignored. They will, however, frequently reappear in the the residual signal when the difference between prediction signal and original sequence is calculated. This effect is clearly visible in the analyzed residual signals from the *BQTerrace* sequence (see Figure 6.12). When the distortion added by the camera's

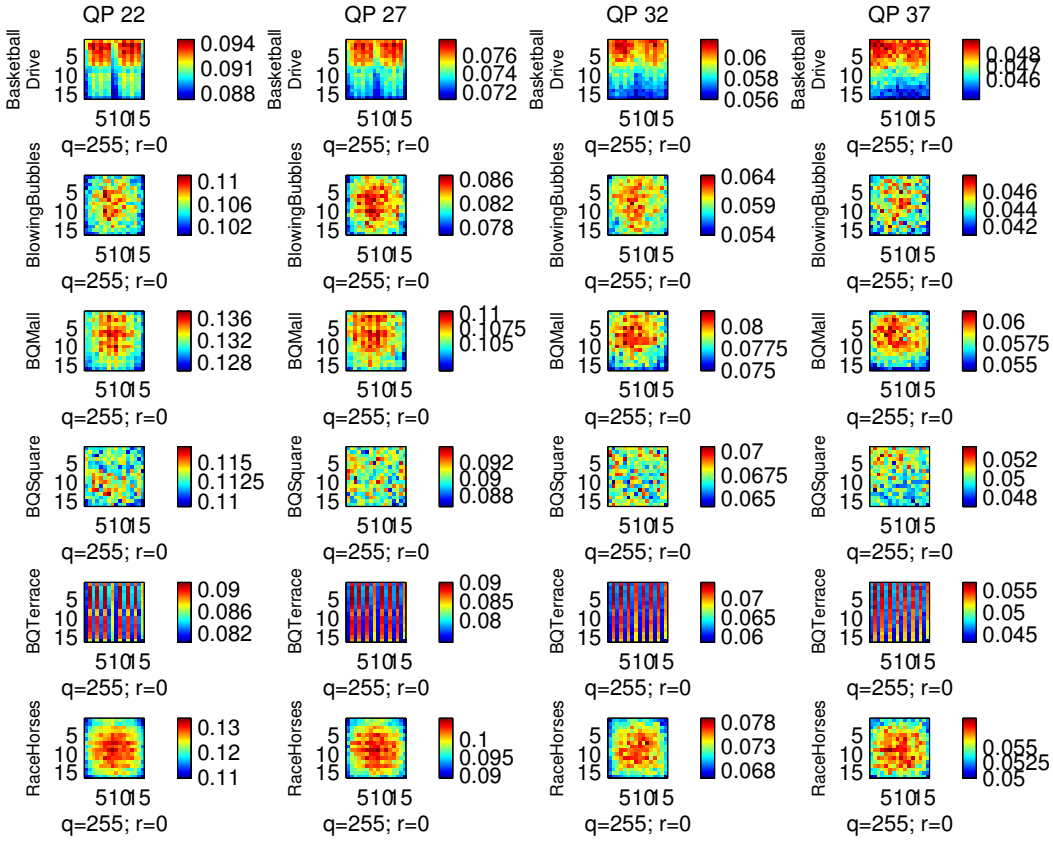


Figure 6.12: Frequency of occurence of errors equal to zero between motion-compensated frame Y_{MC} and the original frame Y_{Orig}. The residual error was discarded in this experiment.

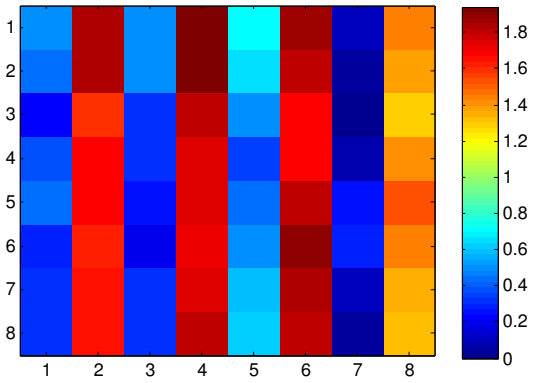


Figure 6.13: Average Y-component distribution per 8×8-block of the first frame of *BQTerrace*. The average minimum luminance value was subtracted from all pixels first.

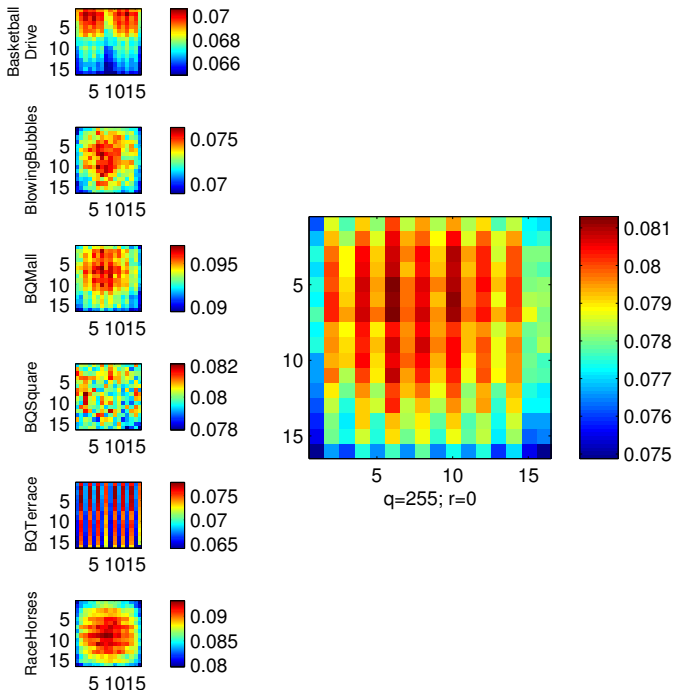


Figure 6.14: *Left:* average over the probability distributions showing an error of zero for all four QPs. *Right:* average over all tested videos.

chip is removed, however, the original distribution of clustered zero errors around the block center reappears.

In order to precisely define an area with above-average reconstruction quality, the number of zero-error pixels should be relatively high. The more often a certain pixel is predicted without distortion, the better its trajectory can be used to train the filter parameters. As a training reference the motion-compensated luminance value Y_{MC} would be used as this value will probably match the original quite frequently. If, in addition, the trajectory is forced to consist of two pixels at least, then the resulting filter parameters will offer a good configuration to calculate pixel trajectories for the remaining pixels of a frame (i.e. the *filter data set*) as well. However, taking into account the actual values shown on the right-hand side of Figure 6.14 only values between 7.5% and 8.1% occur. In more then 90% of all tested cases, a pixel is noisy no matter where in a 16×16 block it lies. Should, however, the remaining 90% of the pixels only have a small residual error, then a removal of certain extreme outliers after training the filter would still be possible. To this end, the distribution of the average luminance error is investigated next.

Average Error

The distribution of the average error shown in Figures A.1 and A.2 in the Appendix are supposed to demonstrate that smaller errors tend to occur more frequently at the center of a 16×16 block. Compared with the previously shown zero-error distributions no tendency towards spatial separation is visible. Again *BQTerrace* displays the pattern of horizontal stripes. The remaining sequences show random distributions. In addition, the average values range from -0.1 to -0.2 for *Blowingbubbles* at QP 22 and between 0 and -1 for *BQTerrace* at QP 22. The residual is obviously biased. This observation was one motivation for the development of the SAO filter described in [19]. In addition, it is obvious that *BQTerrace* with its distinct error residual has a strong impact on the average error distribution. Concerning the separation of this distribution into a *training data set* and a *filter data set* no clear decision can be made based on the test material.

Mean Squared Error

In order to support the theory of a better motion prediction at the macro-block center, the MSE is now measured. This also corresponds to the power of the difference signal $Y_{\text{orig}} - Y_{MC}$ between original and motion-compensated frame. The distributions in Figures A.3 and A.4 (see Appendix) show that for all smaller sequences larger errors occur at the macroblock boundary. This obviously supports

the formulated theory but it probably needs to be restricted to lower resolutions. The deviating behavior of the HD-sequences is very likely due to the HD-cameras' CMOS chips. In addition, for a low-resolution sequence a moving foreground object will distort the boundary of only one 16×16 block. The same image content will be distributed over many more 16×16 blocks at higher resolutions. In this case the distortion will not only affect the block boundary but several 16×16 blocks at a time. Neighboring blocks will not be distorted, however, and can be transmitted without additional boundary distortions. Despite the asymmetric error distributions of *BasketballDrive* and *BQTerrace* a clear 1 to 2 pixel boundary can be distinguished in Figure A.4, which shows that the MSE at a block boundary is generally higher.

Results

From the distributions described above it is possible to separate a pixel boundary of width 1 as *filter data set* from the remaining 14×14 pixels that constitute the *training data set* within a macro block. For each 16×16 block this results in 60 pixels that can be filtered and 196 pixels to train the filter. With a ratio of $\frac{60}{256} \approx 0.24$ only around a quarter of pixels with a trajectory can be improved, which is the general drawback of this approach. The more trajectories are available for the *training data set* the better the filter performance, while less pixels can actually be filtered, thus reducing the achievable gain. If, on the other hand, more pixels are used for filtering than for training, more pixels can actually be denoised while the filter will have poorer quality.

Due to these implications, three different settings will be tested in the subsequent experiments: A *filter data set* with a width of 1 pixel is chosen due to the previously described tests. A second setting with a boundary of 2 pixels corresponds to a ratio of 0.77 and a third with a boundary of 4 pixels even has a ratio of 3. Since the motion-compensated luminance values without residual Y_{MC} are used for training, each trajectory must encompass at least one motion vector and two pixels: the motion-compensated luminance value Y_{MC} itself, which corresponds to a pixel from the previous frame, and the first referenced pixel Y_2. With these two, it is possible to define both a luminance difference and an angular vector difference, which are associated with the first trajectory node. Training and filtering are only done for all pixels from the second node (Y_2) onwards and the parameters are chosen so that the difference to Y_{MC} (within the *training data set*) becomes minimal.

The motivation for this approach is the following: If trajectories within the *training data set* can be optimized to produce filtered values close to the original frame (here represented by Y_{MC}), the same thresholds should work well on all other pixels,

too. These thresholds are subsequently applied to the *filter data set*. The results of all three settings tested within the H.264/AVC baseline profile are shown in Table 6.2. Unfortunately, the results do not reflect the expected gain. For a boundary

Sequence	1 Pixel Boundary		2 Pixel Boundary		4 Pixel Boundary	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballDrive	-1.00	47.57	-1.22	72.42	-1.64	144.56
BlowingBubbles	-0.02	0.45	-0.04	0.76	-0.07	1.50
BQMall	0.00	0.06	0.00	0.09	-0.01	0.14
BQSquare	-0.07	1.45	-0.13	2.70	-0.25	5.46
RaceHorses	0.00	0.04	-0.01	0.15	-0.04	0.77
Waterfall	0.00	0.00	0.00	0.01	0.00	0.05
average	**-0.18**	**8.26**	**-0.23**	**12.69**	**-0.33**	**25.41**

Table 6.2: BD-rate and BD-PSNR for *brute-force* filtering of videos encoded with a fixed block-size of 16×16. Y_{MC} is used as a reference.

of one pixel an average loss of 0.18 dB is observed. For two pixels and four pixels a BD-PSNR of -0.23 dB and -0.33 dB is produced. As a result, filtering with Y_{MC} as reference does not offer a viable opportunity to improve the video quality. Especially *BasketballDrive* with a loss of more than 1 dB and a BD-rate of up to 144.56% is an extreme outlier. This behavior corresponds well to the asymmetric error distribution shown Figure A.3 (see Appendix). The distribution for *BasketballDrive* clearly highlights, that the concept of a training area in the middle of the motion-compensated block is not justified for this sequence and the resulting trajectories are of poor quality. Subsequently, the reference Bjøntegaard bit rate at identical quality is increased by more than 100% since the TTF introduces new artifacts which increase the MSE. An explanation applicable to all sequences is the fact that Y_{dec} is excluded from the filtering process. This value could substantially improve the filter since it generally has a smaller distance to the original than the motion-compensated version Y_{MC} due to the added residual.

6.2.3 16×16 only, Motion-Compensated with Residual

Since post-filtering with the motion-compensated frame Y_{MC} as a reference has not provided the expected quality improvements, the decoded, reconstructed luminance values Y_{dec} are now used as a reference instead.

Zero-Errors

The zero-error distributions, if the test videos are again encoded with a fixed block-size of 16×16, may be found in Figure 6.15. The distributions show the probability with which a certain pixel within a 16×16 block of the reconstructed frame exactly matches the original. Compared with the previously conducted tests on the motion-compensated frame Y_{MC}, no clear separation is possible here. However, some sequences, such as *BQSquare* and *RaceHorses*, show a 1 pixel boundary within which a larger number of errors occures compared with the rest of the block. Other sequences, such as *BasketballDrive* and *BQTerrace* again show a deviating distribution. As in Section 6.2.2 these sequences have an asymmetric error distribution. But most of the pixels with a high residual error probability are located at the block boundary. The average distribution in Figure 6.16 again highlights the 1 pixel boundary as a suitable choice for a separation criterion. A boundary of width 2 also can be justified by the experiments. In order to make the following results comparable with the previously conducted experiments for the motion-compensated reference, a boundary of width 4 is also investigated. Nevertheless, it is expected that smaller boundaries will again provide better results.

Average Error

As in Section 6.2.2 the error distributions in Figures A.5 and A.6 in the Appendix show no distinct separation criteria. However, the average errors are significantly reduced due to the added residual. In addition, the influence of the 4×4 DCT is visible for some QPs (*RaceHorses*, QP 22 and *BQMall*, QP 22). The strong impact of the *BQTerrace* sequence on the average distribution in Figure A.6 has remained the same. The average error has been decreased from values between -0.6 and -0.9 for the motion-cmpensated case to values from -0.45 to -0.75. The effect of this change is twofold. Firstly, the robustness of the training is increased. Secondly, the maximum achievable filter gain is decreased due to the reduced maximum error.

Mean Squared Error

In contrast to the results in presented in Section 6.2.2, no clear separation between *training data set* and *filter data set* for the smaller sequences is possible here. Due to the added residual the power of the observed error is reduced. Nevertheless, a boundary of width 1 remains whose error statistic differs slightly from those of the inner pixels. The summarized representation on the right-hand side of Figure A.8 (see Appendix) as well as the per-sequence histograms provide at least justification for a separate examination of the block boundary. Again the same three boundary

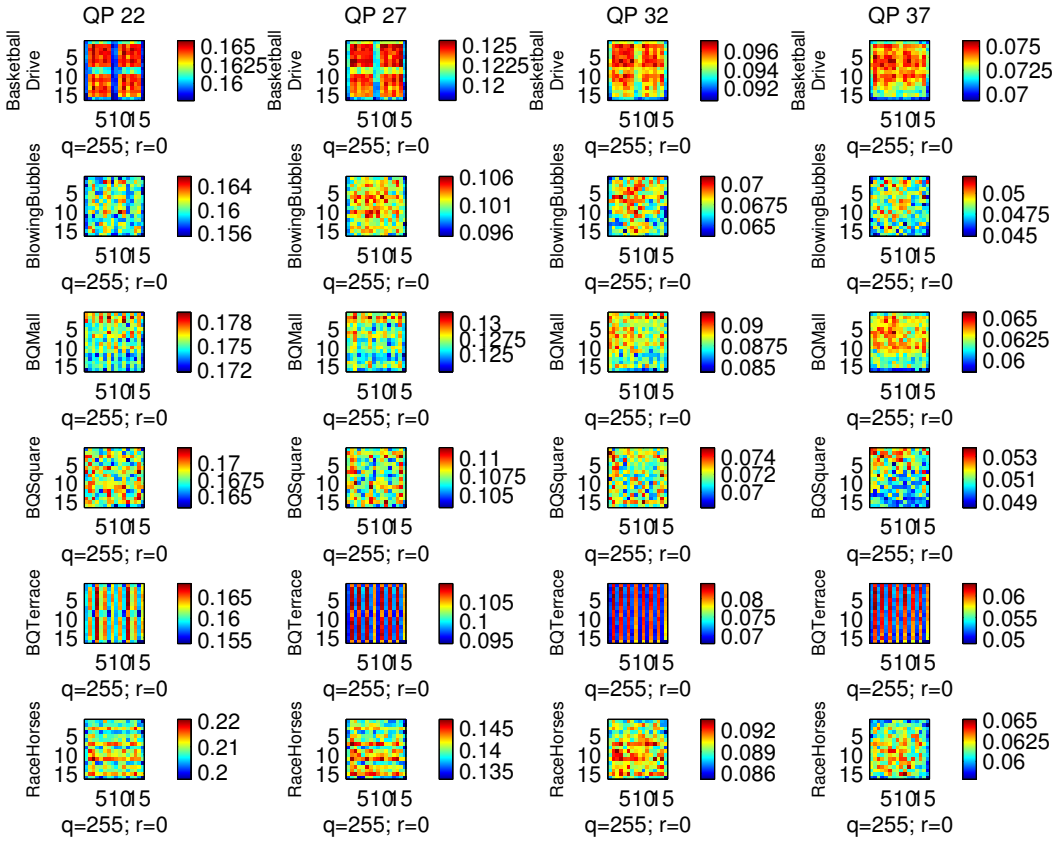


Figure 6.15: Frequency of occurence of error values equal to zero between the reconstructed frame Y_{dec} and the original frame Y_{orig}.

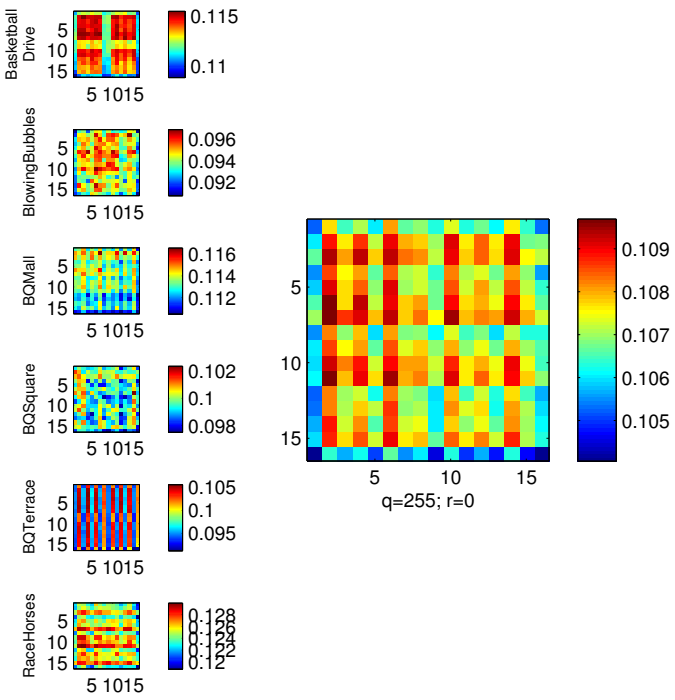


Figure 6.16: *Left:* Average of the distributions over all four tested QPs showing the frequency of occurence of zero-errors. *Right:* Average over all tested videos.

Sequence	1 Pixel Boundary		2 Pixel Boundary		4 Pixel Boundary	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballDrive	-0.97	47.93	-1.20	71.78	-1.55	111.16
BlowingBubbles	0.00	0.08	-0.01	0.12	-0.01	0.31
BQMall	0.00	0.09	0.00	0.11	-0.01	0.18
BQSquare	-0.05	1.04	-0.09	1.96	-0.18	3.91
RaceHorses	0.00	0.03	0.00	0.06	-0.02	0.26
Waterfall	0.00	0.03	0.00	0.09	-0.01	0.15
average	**-0.17**	**8.20**	**-0.22**	**12.35**	**-0.30**	**19.33**

Table 6.3: BD-rate and BD-PSNR for *brute-force* filtering of videos with a fixed block-size of 16×16. Y_{dec} was chosen as a reference.

settings (1 pixel, 2 pixels, and 4 pixels) are used. The decoded frame Y_{dec} is now used as a reference.

Results

The summarized results of the experimental evaluation are shown in Table 6.3. Unfortunately, a boundary of width 1 still produces a loss $-0.17\,\text{dB}$ on average. The improvement of $0.01\,\text{dB}$ compared to the Y_{MC} reference is negligible. Even for boundaries of size 2 with $-0.22\,\text{dB}$ ($+0.01\,\text{dB}$) and 4 with $-0.20\,\text{dB}$ ($+0.03\,\text{dB}$) no actual filter gain can be observed. It remains to be stated that with an increasing boundary width the quality of the filtered frame is decreased while the required bit rate for transmission with the same quality is increased. The results are improved, however, when the decoded value Y_{dec} is included in the filtering process. Only *BQMall* and *Waterfall* show a different behavior: *BQMall* always produces the same quality despite the variation of the boundary width and requires a higher bit rate when the decoded frame is used as a reference. The same is true for *Waterfall* where additionally the quality is decreased with a four-pixel boundary. Based on the results of Sections 6.2.2 and 6.2.3 it can be concluded that a pure post-filtering approach with a boundary-separated version of Y_{MC} or Y_{dec} as reference cannot produce satisfactory quality improvements. Even if the results for *BasketballDrive* are excluded from the average, no filter gain can be observed.

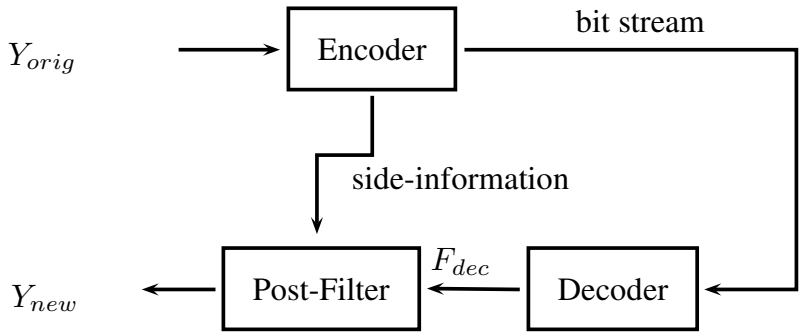


Figure 6.17: A post-filter with additional side-information for optimal quality control.

6.3 *Brute-Force* Filtering With Side-Information

As all post-filtering attempts so far have not provided acceptable filtering gain, the boundary of the theoretically possible maximum gain is now explored. To this end all trajectories are used both as the *training data set* and the *filter data set*, where Y_{orig} is the reference and a parameter combination yielding the minimal MSE is selected and transmitted to the decoder (see Figure 6.17). The side-information consists of the thresholds for ΔY and Φ_t and can be transmitted with $2 \cdot 8 = 16$ additional bits per frame. With respect to the number of frames per second of each sequence, not only the PSNR values are changed but also the bit rates.

16×16 Blocks only

Columns 2 and 3 of Table 6.4 list the filtering results for all sequences when a fixed block size of 16×16 is used. The best results are obtained for *BQSquare* and *Waterfall*, where both the BD-rate is decreased and the image quality is improved by values from 0.03 dB to 0.05 dB. All other sequences do not show significant quality improvements, but the bit rate for transmitting these sequences at a predefined quality is decreased. Only for the *RaceHorses* sequence the bit rate is slightly increased, as side-information is added to the bit stream without improving the visual or objective quality. Despite these improvements such small changes are generally not visible for a human observer. Since it is possible that the 16×16 block-based motion estimation inhibits the filter's effectiveness, other block sizes are now examined as well.

Sequence	16×16		all sizes	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballDrive	0.00	-0.10	0.00	-0.07
BlowingBubbles	0.00	-0.02	0.00	-0.07
BQMall	0.00	0.00	0.00	-0.01
BQSquare	0.05	-1.11	0.04	-0.81
RaceHorses	0.00	0.01	0.00	0.00
Waterfall	0.03	-0.74	0.05	-1.08
average	**0.02**	**-0.33**	**0.02**	**-0.34**

Table 6.4: BD-rate and BD-PSNR for *brute-force* filtering. Y_{orig} was used as reference and all trajectories are part of both *training data set* and *filter data set*. Per frame an overhead of $2 \cdot 8$bit is transmitted.

8×8 Blocks only

A smaller block-size was originally selected with the hope of more accurately predicting the motion of individual pixels. However, the first tested videos *BQSquare* and *Waterfall* already showed BD-rate increases of 0.6% and a decreased BD-PSNR. For this reason no further experiments were conducted. Theoretically four 8×8 blocks should more accurately represent the motion of a 16×16 block. However, a smaller block-size also means that the true motion of recurring patterns or structures in a frame cannot precisely be tracked.

All Possible Block-Sizes

Most commonly employed H.264/AVC encoders will of course use the full range of available block-sizes. Theoretically, this should ensure an optimal trade-off between bit rate and achievable quality. The results for this setting are given in Columns 4 and 5 of Table 6.4. Interestingly, the quality improvement for *BQSquare* is reduced in this setting, while *Waterfall* is improved by another 0.02 dB. For all other sequences the quality improvement is still negligible, while the BD-rates are slightly improved.

6.3.1 Local Separation with an Artificial Neural Network

Another variant of a post-filter is the implementation with an artificial neural network. Due to the ANN's ability to learn dependencies between input and output autonomously, these networks are employed whenever such dependencies are not

Sequence	16×16, $2 \cdot 32$ bit		all sizes, $2 \cdot 32$ bit	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BlowingBubbles	-0.01	0.22	-0.01	0.26
BQMall	-0.01	0.11	-0.01	0.13
BQSquare	0.07	-1.32	0.04	-0.72
RaceHorses	-0.01	0.20	-0.01	0.24
Waterfall	0.00	0.03	0.00	-0.02
average	**0.01**	**-0.15**	**0.00**	**-0.02**

Table 6.5: BD-Rate and PSNR-Rate for the artificial neural network. Y_{orig} is used as a reference and all trajectories are both part of the *training data set* and the *filter data set*. For each frame 64 bit of side-information are transmitted.

clearly defined. In addition, they are able to find optimal solutions with the help of adequate learning algorithms. With the help of the back-propagation algorithm described in Section 6.1.7 the MSE of each decoded frame is minimized. In order to keep the computational runtime within reasonable limits the simplest form of the artificial neural network with only ΔY as an input value is used. As will be shown in Chapter 7, adding a temporal threshold to the filter will only produce slight improvements. Y_{orig} is used as the reference for optimization and side-information in the form of neuron weights is again transmitted.

6.3.2 16×16-Blocks only, Artificial Neural Network with the Original as Reference

Similar to the experiments conducted in Section 6.2.1 here, too, compressed videos with a fixed block-size of 16×16 are investigated. The results in columns 2 and 3 of Table 6.5 show, that the BD-rate is increased for all sequences except *BQSquare*. This can be explained by the large amount of side-information per frame. During the search for the optimal PSNR value the highest possible precision (32 bit) was chosen for the two weights w_0 and w_1 each. A floating-point number with a lower precision could possibly yield comparable results. Despite the increased amount of side-information, the BD-rate for *BQSquare* is even better than for the equivalent *brute-force* solution.

6.3.3 Artificial Neural Network Using all Possible Block-Sizes

Since a restriction to 8×8 blocks did not produce any improvements for the *brute-force* solution, it is not again investigated here. The results for utilizing all possible block-sizes are shown in Colums 4 and 5 of Table 6.5. These show a similar behavior to the *brute-force* case with an additional BD-rate reduction for *BQSquare* when only 16×16 blocks are used (compare with Table 6.4). The poorer performance of the neural network here is mostly due to the 64 bits of side-information that are transmitted per frame. The actual PSNR values comparing input and output of the post-filter at a certain QP always show quality improvements.

6.3.4 Neural Network with Quantized Parameters

Since the main disadvantage of the neural network compared to the *brute-force* filter is the increased amount of side-information, a modified version, that uses quantized parameters instead, is now also investigated. The results produced by the modified artificial neural network may be found in Table 6.6. For all tested sequences a quantization of the parameters w_0 and w_1 (represented due to their linear nature by Th_{-3} and Th_{Θ})) to 16 bit and 8 bit respectively only shows slight variations in the resulting PSNR. Subsequently the filter's effectiveness is improved when a smaller amount of side-information is transmitted. However, quantizing the two weights below 8 bit always resulted in drastic losses, so that no further compression of said parameters was investigated. In the case of 8 bit per weight the filter performs similarly to the *brute-force* implementation. Although the highest gain is achieved for *BQSquare* and not for *Waterfall*. Most importantly the post-filter now always improves the BD-PSNR or has no impact on the sequence.

6.3.5 Evaluation of the Filter based on an Artificial Neural Network

The neural network has a huge drawback compared to the *brute-force* solution where the side-information is concerned. Since the weights within the neural network are not restricted to integer numbers, formats with a flexible precision are required. The more bits are available for the filter weights the more accurately the filter can adapt itself. The results produced by the post-filter show small losses for *BlowingBubbles*, *BQMall* and *RaceHorses* even when quanitzed parameters are used. However, the loss is only due to the trade-off between bit rate and PSNR in the Bjøntegaard metric [4].

Sequence	all sizes, $2 \cdot 16$ bit		all sizes, $2 \cdot 8$ bit	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BlowingBubbles	-0.01	0.11	0.00	0.03
BQMall	0.00	0.07	0.00	0.04
BQSquare	0.07	-1.33	0.07	-1.38
RaceHorses	-0.01	0.12	0.00	0.07
Waterfall	0.00	-0.11	0.01	-0.13
average	**0.01**	**-0.23**	**0.02**	**-0.27**

Table 6.6: BD-Rate and PSNR-Rate for the artificial neural network with arbitrary block-sizes and quantized parameters.

If the impact of the side-information is ignored, the neural network actually shows higher quality improvements for *BQSquare* at certain QPs than the *brute-force* solution. The reason for the additional gain lies with the different interpretation of the input values by both filters. The *brute-force* filter simply computes an average luminance value along the trajectory and discards certain outliers. Due to the shape of the *sigmoid function* the neural network uses a larger number of input values but weights them differently. Both implementations here work similarly, since the temporal threshold for *BQSquare* is usually set to the maximum. For the *Waterfall* sequence the impact of the temporal threshold is much more noticable, as the *brute-force* filter will select a different threshold value for each frame. Due to the abscence of this threshold in the neural network implementation, the realised gain for this sequence is far smaller.

6.3.6 Summary

To summarize, the *brute-force* filter with side-information showed the best performance among all post-filter approaches. In all other cases significant losses for some sequences were observed. It appears to be obvious that the current implementation of the TTF always requires control from the encoder. For this reason the *brute-force* filter with side-information is now separately investigated within the HEVC low-delay profile.

6.4 Post-Filtering Approach for HEVC

If a fixed set of parameters is used for an entire sequence, the filter can no longer adapt itself to changing image content or differently moving foreground objects. One way to overcome this difficulty is to calculate filter parameters at the encoder and to transmit them in addition to the actual bit stream describing the video sequence. If the decoder chooses not to use the filter then the bit rate will of course increase slightly. At nine bits per frame for the simple TTF, however, this overhead should be acceptable for most video formats and quality levels. If the filter is applied, the bit rate will be identical but the quality of the decoded frames can be improved. Mainly to emphasize the superiority of the in-loop filter, experiments with a TTF post-filter were also conducted on the HEVC test data set. As in the case of the simple TTF, the QTTF, too, can be employed as a post-filter. Here, however, more problems need to be addressed. Even though the great advantage of the QTTF is its adaptability to differently structured or differently moving image regions, this adaptability is only provided by the underlying quadtree. The quadtree itself can be of arbitrary size and it is only restricted by the RD-optimization which has been described in Section 5.3.3. Even when the *brute-force* method is utilized the quadtree is, of course, only optimal if the QTTF is used at the decoder at all. Otherwise the bit rate will simply be increased by the quadtree information. In this scenario, it is subsequently essential to restrict the maximum size of the quadtree which can be easily accomplished through the introduction of a minimum block size. Bit rate reductions provided by the QTTF as a post-filter were again measured for all HEVC test sequences.

	TTF		TTF post-filter		QTTF brute-force		QTTF brute-force post-filter	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballPass	0.00	−0.04	0.00	0.00	0.01	−0.20	0.01	−0.14
BlowingBubbles	0.03	−0.64	0.00	−0.02	0.04	−0.99	0.02	−0.43
BQSquare	6.06	−0.23	0.06	−1.61	0.26	−6.94	0.13	−3.48
RaceHorses	0.01	−0.13	0.00	−0.03	0.01	−0.20	0.01	−0.22
avg. for class D	**0.07**	**−1.62**	**0.02**	**−0.42**	**0.08**	**−2.08**	**0.04**	**−1.07**
BasketballDrill	0.00	−0.12	0.00	0.00	0.00	−0.15	0.00	−0.01
BQMall	0.00	−0.09	0.00	0.02	0.00	−0.08	0.00	−0.02
PartyScene	0.02	−0.42	0.00	−0.03	0.05	−1.25	0.03	−0.64
RaceHorses	0.00	−0.05	0.00	−0.01	0.00	−0.09	0.00	−0.07
avg. for class C	**0.01**	**−0.17**	**0.00**	**0.00**	**0.01**	**−0.39**	**0.01**	**−0.19**
Vidyo1	0.00	−0.18	0.00	0.03	0.02	−0.52	0.00	0.13
Vidyo3	0.01	−0.27	0.00	0.02	0.01	−0.33	0.00	0.09
Vidyo4	0.01	−0.38	0.00	0.03	0.02	−0.71	0.00	0.16
avg. for class E	**0.01**	**−0.27**	**0.00**	**0.02**	**0.02**	**−0.52**	**0.00**	**0.13**
BasketballDrive	0.00	−0.02	0.00	0.00	0.00	−0.06	0.00	0.02
BQTerrace	0.01	−0.74	0.00	−0.05	0.02	−1.40	0.00	0.15
Cactus	0.00	−0.17	0.00	−0.01	0.01	−0.39	0.00	0.09
Kimono1	0.00	−0.08	0.00	0.02	0.00	−0.13	0.00	0.04
ParkScene	0.01	−0.21	0.00	−0.02	0.02	−0.56	−0.01	0.17
avg. for class B	**0.00**	**−0.24**	**0.00**	**−0.01**	**0.01**	**−0.51**	**0.00**	**0.08**
Waterfall	0.23	−7.61	0.03	−0.85	0.27	−9.05	0.06	−1.98

Table 6.7: BD-rate and average PSNR gain for TTF and QTTF both as post-filter applications and in their original in-loop implementation.

When only the original TTF implementation without quadtree optimization is examined, the post-filter realizes about 20% of the gain achieved by the TTF in-loop filter (see Columns 3 and 5 of Table 6.7). This is mostly due to the missing bit rate reduction which can be induced by improved reference frames. In addition, the TTF's post-filter implementation also produces loss for certain badly performing sequences. For *BasketballPass*, *BQMall* and the *vidyo* sequences the side-information of up to nine bits per frame suffices to produce a BD-loss. For the QTTF-based post-filter the situation is different. Since the QTTF already includes a RD-optimization scheme, no loss can be observed here. Nevertheless, the realized filter gain of the QTTF post-filter is only about 50% of the gain produced by the QTTF in-loop implementation.

6.5 Chapter Summary

In this chapter, several different attempts were described to effectively apply the TTF and its derivatives as a post-filter. Among them were a neural network implementation and an unsuperivsed *brute-force* filter without control information from the original sequence. It has been shown that the TTF only produces bit rate reductions when controlled by additional side-information. Moreover, the in-loop implementation continously outperformed all post-filter algorithms. It is thus safe to state that the TTF is best used within the local decoding loop and almost always needs additional side-information to effectively adapt itself to changing sequence characteristics.

Chapter 7

Assessment of the Theoretical Considerations

If you don't know where you are headed, you'll probably end up someplace else. - Douglas J. Eder

A paper describing the one-dimensional case of the theory and results presented in this section was first published in [16].

The theoretical considerations conducted in Chapter 4 have resulted in predictions concerning the optimal filter length and the achievable filter gain. These predictions (see the solution of Equation 4.32) can only be validated if the variables that the solution depends on are known. Among these variables are the maximum motion estimation errors $q_{x,\max}$, $q_{y,\max}$ as well as the variance of the luminance component σ_Y^2. In addition, the error variance along the trajectory σ_η^2 and the mean horizontal and vertical luma sample correlation coefficents α_x and α_y are measured. For three test sequences out of class D of the HEVC test data set and for one additional sequence (*Waterfall*) highly accurate motion data was obtained with the Human-Assisted Motion Annotation Tool (HAMA) [34]. The respective values may be found in Tables C.1 and C.2 in the Appendix. This data was used in Chapter 3 to analyze the TTF's behavior under ideal conditions. When examining the maximal motion estimation errors $q_{x,\max}$ and $q_{y,\max}$ it becomes instantly apparent, that they may differ significantly from each other, depending on the sequence. In the difference the motion within the scene is reflected: Should a sequence contain more horizontal than vertical motion, for instance, then the errors in x-direction will, in general, be larger. This behavior can be explained by the simple fact, that a motion-compensated block containing a majority of pixels from a foreground object and a few pixels from the

image background will produce an error almost equal to the foreground motion for all covered background pixels. Since a dominant motion in x-direction is assumed, the vertical error will be much smaller for said pixels. This effect is most prominent for the *Waterfall* sequence, which contains a zoom out and a downward track of the camera. Subsequently, $q_{y,\max}$ is larger than $q_{x,\max}$ for this sequence.

7.1 Comparison of Predicted and Realized Filter Lengths

In Chapter 4 formulae were derived that allow for the prediction of the best filter length for the TTF and the resulting maximum theoretically achievable gain if motion prediction errors and noisy images are present. The expected error variance after filtering was given in Equation 4.33.

$$E\left[\left(\hat{Y}-Y\right)^2\right] = \frac{\sigma_Y^2}{m} - 2(1+m(q_{x,\max}\cdot\log\alpha_x, q_{y,\max}\cdot\log\alpha_y))\cdot\sigma_Y^2 + \sigma_Y^2 + \frac{\sigma_\eta^2}{m} \tag{7.1}$$

In order to evaluate the accuracy of the above equation, two tests can be performed:

- By finding the minimum of Equation 7.1 with the method described in Chapter 4, an optimum filter length can be calculated. This value can then be compared with the realized filter lengths observed during the experiments.
- The measured filter gain $\frac{E\left[(Y-Y')^2\right]}{E\left[(Y-\hat{Y})^2\right]}$ can be set into relation with the observed PSNR or MSE gain.

In both cases, the free variables of Equation 7.1 need to be determined first: The image correlation coefficients α_x and α_y can easily be measured by calcualting the autocorrelation function of each image row and image column in a video frame respectively. The individual α-value then corresponds to the value of the normalized autocorrelation function $\rho(1)$ next to the maximum of the function at $\rho(0) = 1$. Here, the median correlation values over all frames are used. The spatial variance can equally be estimated by examining one image at a time

$$\sigma^2 = \frac{1}{W\cdot H-1}\sum_{x=0}^{W\cdot H-1}(Y(x)-\bar{Y})^2, \tag{7.2}$$

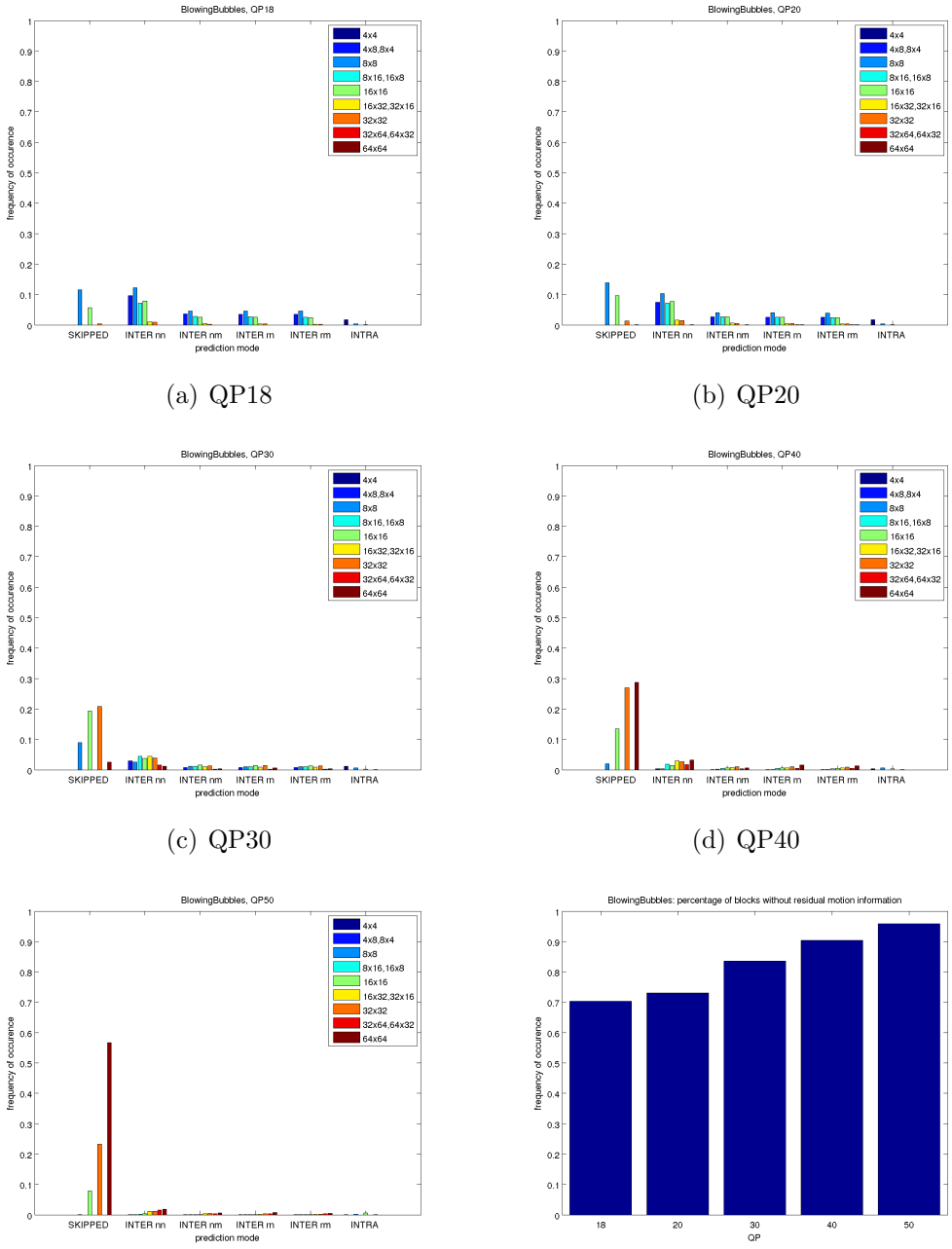
where $\bar{Y}$ is the mean luminance of the current image and W and H are the width and height of the image in pel. In temporal direction the path described by the ground

truth motion vectors is used as the trajectory. Along this trajectory the difference between the encountered luminance values and the original value before compression is measured. The variance of the error can then be computed along the temporal trajectory. Tables C.1 and C.2 in the Appendix give the observed variances for the temporal noise σ_η^2 and for the spatial variance σ_Y^2 of the luminance component. The medians of the maximum endpoint error components $q_{x,\max}$, $q_{y,\max}$ over all frames are also given. In addition, columns 5 and 6 give the mean luminance correlation values in horizontal and vertical direction. The luminance variance is, of course, always measured in the original uncompressed sequence and is thus constant. The temporal error variance changes, as expected, with the used QP. However, a lower bit rate does not necessarily mean a higher σ_η^2 value. Instead, at high QPs the error variance can become smaller, even if its mean value is increased. This effect can be explained by strongly quantized residuals and the more frequent usage of *skip-blocks*. When the *skip* prediction mode is used, the color values along the trajectory are not changed for several frames thus reducing the temporal variability and subsequently the temporal error variance σ_η^2. The luminance correlation values α_x and α_y also only reflect properties of the uncompressed sequence. The maximum motion vector error components $q_{x,\max}$, $q_{y,\max}$ depend on the utilized QP, too.

At low QPs and thus high bit rates the encoder will choose basically random motion vectors in order to reconstruct the best image in an RD-sense. In that range, longer motion vectors can be used if they yield a small residual error. Moreover, the encoder will usually also select smaller blocks for MC in which case not much information is available for finding a good match. At low bit rates, however, *skip-blocks* are chosen more frequently and only very small motion vectors, such as they occur in real video sequences at high frame rates, are transmitted in the bit stream. Subsequently, the maximum motion vector error frequently decreases with a rising QP. A few larger errors will, of course, be introduced by differently moving regions inside the same block or by simple mismatches. On average, however, the ME error is mostly influenced by the RD-optimization scheme depending on the QP. These assumptions are supported by Figure 7.1 showing the number of *skip-blocks* per QP for the *BlowingBubbles* sequence.

7.1.1 Analysis of the Predicted Filter Lengths

Based on the numbers reported in Tables C.1 and C.2 a prediction concerning the expected optimum filter length can be made. In order to compare the prediction with real-world data from the QTTF implementation, one has to consider the limitations of the HEVC encoder. Due to the utilized IBBB coding structure, the trajectory is split at every B-frame. Thus a trajectory spanning n frames can include up to



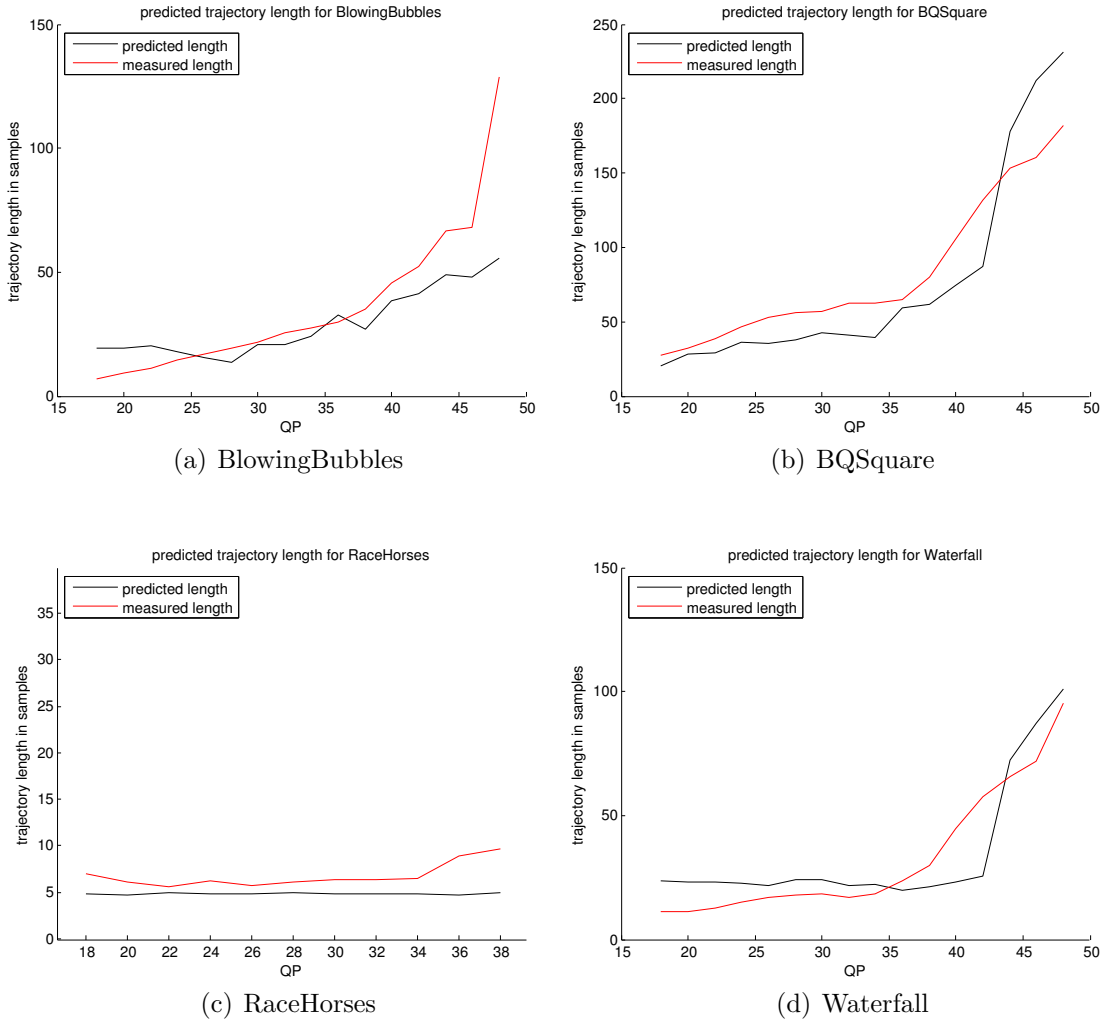
(a) QP18 (b) QP20

(c) QP30 (d) QP40

(e) QP50 (f) summary

Figure 7.1: Subfigures a) to e) show the block prediction mode distribution for the *BlowingBubbles* sequence encoded with the HM 3.0 low-delay high efficiency setting. The *inter*-modes are divided into modes with (m) and without motion vector residual (n) and those with (r) and without (n) a DCT-residual. Subfigure f) summarizes the percentages of blocks utilizing the *skip* mode.

2^{n-1} samples. For comparability a predicted filter length x is consequently replaced by 2^{x-1}. The predicted and observed average filter lengths based on the sequence QP can be found in Figure 7.2. There, the predicted optimum filter length shows a



(a) BlowingBubbles (b) BQSquare

(c) RaceHorses (d) Waterfall

Figure 7.2: The black curve shows the expected optimal filter length. The red curve represents the observed average trajectory lengths.

strong dependency on the QP. Nevertheless the relationship is by no means exponential or even linear and shows a different behavior for each tested sequence. In all tested cases the observed and predicted filter lengths match very closely and, even though significant differences sometimes occur at low bit rates, the predicted filter lengths appear to be a very good predictor for the behavior of the real-world implementation.

In the case of the *BQSquare* sequence, predicted and realized filter lengths are very similar to each other. A particularly good match can be observed for low QPs. Since exact measures for the temporal error were obtained, which match the used analytical model quite well, the theoretical predictions are tuned to match the actually used filter lengths very well. In this context, it is to be highlighted that the error term does not only consist of the described white noise component but also of a constant non-zero mean. This mean has no influence on the TTF's performance since a constant mean cannot be removed through temporal filtering. In Chapter 4, this colored noise is represented by the cross-correlation terms $E[Y(x_i, y_i)Y(x_j, y_j)]$. These introduce an additional error, depending on the correlation values α_x, α_y within the frame itself.

For the *RaceHorses* (D) sequence the behavior is identical. At high QPs the fluctuations again become bigger, but the predictions for mid-range and low QPs are highly accurate. It is to be noted, that the predictions for *RaceHorses* and *BQSquare* do not only share their curvature with the measured values but that both even match them quantitatively. The deviations for *RaceHorses* again only become larger for videos of poor quality. For *RaceHorses* no data is shown for QPs higher than 38 since the filter is generally switched off at poor quality for this sequence. Subsequently, no trajectory lengths are available for QPs 40 and above.

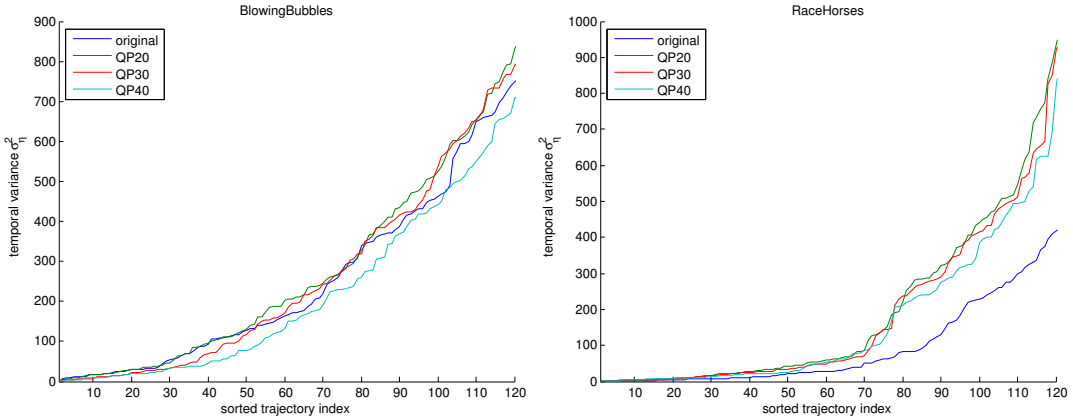
For *BlowingBubbles* a huge difference between theory and experiment only occurs for QP 48. Here, as well as at very high bit rates (low QPs), the TTF is switched off for too many pixels to provide reliable average trajectory lengths.

Interestingly, the predicted and realized filter lengths for *Waterfall* are much smaller than those for *BQSquare*, even though *Waterfall* has always produced a higher gain in the experiments. The reason for this may be found when comparing Tables C.1 and C.2: There the compression-induced error variance for *Waterfall* can be seen to be significantly higher than the variance for *BQSquare*. Subsequently, even short trajectories can provide acceptable gain for this sequence. In summary, all four videos demonstrate the applicability of the formulae derived in Chapter 4 very well.

7.1.2 Variance Analysis

In Table C.1 a dependency of the error variance σ_η^2 on the QP can be observed. However, it still remains to be shown that the assumption of image content undergoing only translational motion (see Chapter 4) is actually valid. If this hypothesis is true, the variance of a single pixel motion compensated pixel along the temporal axis should be small. Changes can now only be introduced by occlusion, new image

content, and lighting changes. Figure 7.3 shows the sorted variances of 120 pixels that were succesfully tracked over at least 20 frames in the sequences *RaceHorses* and *BlowingBubbles*.



(a) BlowingBubbles (b) RaceHorses

Figure 7.3: For all tracked pixels the variance was estimated and the results were sorted in ascending order. Due to the thresholds of TTF, all pixels with an extreme variance would be discarded and thus not filtered.

The displayed variances per pixel were sorted ascendingly first to eliminate extreme outliers, that are discarded by the TTF. In the case of the *RaceHorses* sequence the temporal variance of the original sequence is significantly lower than the variances originating from compressed versions of the sequence. This indicates that filtering should always be possible as long as adequate motion information is available. For *BlowingBubbles* the situation is slightly different. Even though the original variance is generally below those of QPs 20 and 30, the temporal stability of the sequence encoded at QP 40 is higher. Subsequently, the temporal variance decreases for higher QPs. Here, this is due to the nature of the sequence with transparent foreground objects which dissappear at low bit rates when large parts of the frame turn into smooth textureless regions. Then the compressed sequence shows, of course, more temporal stability even though the average error between reconstructed and original sequence is large.

7.2 Analysis of the Filter Functionality

In all experiments conducted in this thesis, the TTF was tested with three thresholds, namely a luminance threshold T_Y, a spatial consistency threshold T_{SC}, and

a temporal consistency threshold T_{TC}. As has been mentioned in Section 5.3, the spatial threshold only showed good performance at low bit rates. In order to further validate the usability of these thresholds, experiments have been conducted to analyze the filter performance with and without each of the aforementioned thresholds. Since disabling the luminance threshold generally renders the filter useless, only the following combinations were tested:

1. All three thresholds are used together. This implementation is identical to the one described in Section 5.3 except for the fact that the spatial threshold was originally only used for high QPs. This configuration will be referred to as QTTF_YST.

2. Luminance threshold and temporal threshold together: At high bit rates the performance of this configuration (QTTF_YT) provides the same gain as the original QTTF.

3. Luminance threshold and spatial threshold together (QTTF_YS): This corresponds to the original QTTF implementation described in [12].

4. QTTF with only the luminance threshold (QTTF_Y): The expected advantage of this approach lies in the considerable reduction of the side-information while the most promising threshold is kept.

The QTTF was chosen as a basis for this test, since its performance is generally better than TTF and the observed differences will thus be more significant. Table 7.1 lists the observed results in terms of BD-performance for all configurations described above. As can be seen from the table, the majority of the gain results indeed from the luminance threshold T_Y which produces around 75% of the observed total gain. The temporal consistency threshold T_{TC} accounts for around 20% of the gain and for some sequences the combination of only these two thresholds is even better than the original implementation. Only for high QPs does the spatial consistency threshold T_{SC} further improve the results. Its impact is, however, far smaller. These observations support the conclusion, that a simplified encoder that optimizes only T_Y and T_{TC} will generally produce almost the same ouput quality with a complexity reduction in the area of 70%. This trend led to the ommission of the spatial consistency threshold in all implementations based on HM 8.0. There, the influence of T_{SC} was generally even smaller due to the improved motion vector accuracy. Nevertheless, the selected filter configuration consisting of all three thresholds produced a good gain over all sequences and quality levels on average.

	QTTF_YST		QTTF_Y		QTTF_YT		QTTF_YS	
	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %	Δ_{PSNR} in dB	BD-rate in %
BasketballPass	0.01	−0.20	0.01	−0.15	0.01	−0.23	0.01	−0.15
BlowingBubbles	0.04	−0.99	0.04	−0.97	0.04	−1.00	0.04	−0.87
BQSquare	0.26	−6.94	0.27	−7.09	0.26	−7.00	0.26	−6.94
RaceHorses	0.01	−0.20	0.01	−0.15	0.01	−0.24	0.01	−0.16
avg. for class D	**0.08**	**−2.08**	**0.08**	**−2.09**	**0.08**	**−2.12**	**0.08**	**−2.03**
BasketballDrill	0.01	−0.15	0.00	−0.10	0.01	−0.23	0.00	−0.12
BQMall	0.00	−0.08	0.00	−0.06	0.00	−0.12	0.00	−0.09
PartyScene	0.05	−1.25	0.04	−1.04	0.05	−1.22	0.04	−1.05
RaceHorses	0.00	−0.09	0.00	−0.03	0.00	−0.06	0.00	−0.10
avg. for class C	**0.02**	**−0.39**	**0.01**	**−0.31**	**0.02**	**−0.41**	**0.01**	**−0.34**
Vidyo1	0.02	−0.52	0.00	0.08	0.02	−0.46	0.00	0.10
Vidyo3	0.01	−0.33	0.00	−0.11	0.02	−0.63	0.01	−0.24
Vidyo4	0.02	−0.71	0.00	−0.13	0.03	−0.82	0.00	−0.14
avg. for class E	**0.02**	**−0.52**	**0.00**	**−0.05**	**0.02**	**−0.64**	**0.00**	**−0.09**
BasketballDrive	0.00	−0.04	0.00	−0.01	0.00	−0.05	0.00	0.00
BQTerrace	0.02	−1.46	0.02	−1.16	0.02	−1.53	0.02	−1.16
Cactus	0.00	−0.23	0.00	−0.14	0.01	−0.37	0.00	−0.16
Kimono1	0.00	−0.08	0.00	−0.04	0.00	−0.09	0.00	−0.04
ParkScene	0.02	−0.56	0.01	−0.34	0.02	−0.65	0.01	−0.37
avg. for class B	**0.01**	**−0.47**	**0.01**	**−0.34**	**0.01**	**−0.54**	**0.01**	**−0.35**
Waterfall	0.27	−9.05	0.27	−8.90	0.27	−9.11	0.26	−8.55

Table 7.1: BD-rate and average PSNR gain for QTTF for different threshold combinations for the HEVC low-delay high efficiency setting.

7.3 Filtering of Foreground and Background

In Chapter 1 it was predicted, that the TTF would work equally well on both foreground and background. Even though high gains on both sequences with and without large foreground objects imply that the initial assumption is true, conclusive proof still has to be presented. To this end, three test sequences out of class D (*BlowingBubbles*, *BQSquare*, *RaceHorses*) were reexamined. These were on the one hand chosen because of their relatively fast encoding and decoding times and also because they reflect the most typical combinations of foreground objects and background scenery:

- In the *BlowingBubbles* sequence slowly moving foreground objects (two girls) are present, which occupy about half of each spatial frame on average.
- Large and quickly moving foreground objects (horses) are depicted in the *RaceHorses (D)* sequence, which is especially challenging for temporal filters as the background is mostly occluded and even inter-object occlusion occurs.
- The *BQSquare* sequence, on the other hand, consists mostly of slowly moving background with only a few small foreground objects (people walking over a terrace).

Each of these sequences has been encoded with HM 3.0 with the added QTTF. For each frame the reconstructed unfiltered signal was compared with the output of the QTTF. For each pixel it was determined whether the filtered signal was closer to the original than the unfiltered one. Exemplary results for these experiments for QP 32 may be found in Figure 7.4.

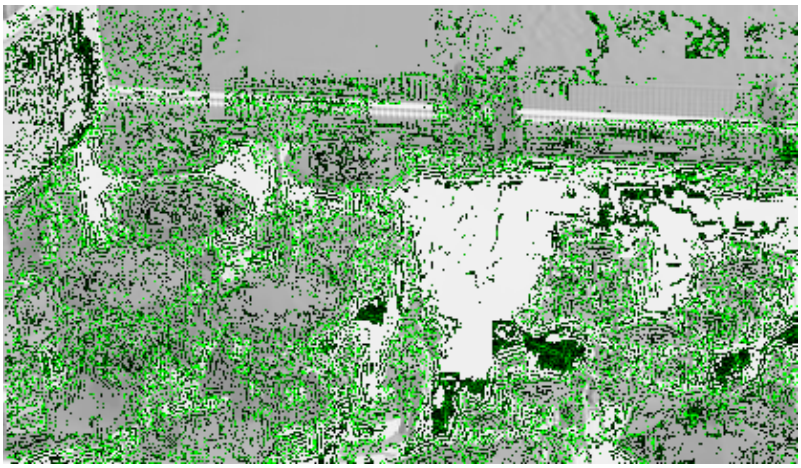
It is firstly to be noted, that the number of filtered pixels per frame corresponds quite well with the achieved gain per sequence: The highest gain among these three test sequences was produced for *BQSquare* where around 40% of the pixels have been filtered. The smallest gain could be observed for *RaceHorses (D)* where the number of filtered pixels is much smaller. Both for *BlowingBubbles* and *BQSquare* the filtered pixels are more or less evenly distributed over the entire frame. In the case of *BQSquare* all foreground objects are also filtered. In *BlowingBubbles* the girl on the left is also completely filtered. While the filter has been switched off for the upper right corner of the displayed frame. The structure of the underlying quadtree can also be deduced from the distribution of the filtered and improved pixels.

The quadtree is responsible for the sharp edges in the pixels marked in green. In the case of the *RaceHorses* sequence only very few of the pixels belonging to the foreground are filtered. However, for this sequence the filter is switched off quite often for the displayed QP. Figure 7.5 shows the same frame at the four tested QPs. As can be seen, large parts of the background are filtered in all cases. The foreground is also widely filtered for QP 37. This corresponds to the decreased motion vector impairment at that QP (see Tables C.1 and C.2). Figure 7.6 shows one exemplary frame at four different QPs for the *BlowingBubbles* sequence. Here, too, the filter is switched on for all QPs, but is most effective for QPs 32 and 37. Moreover, the foreground objects are always part of the filtered area despite their motion relative to the background. For the *Waterfall* sequence the largest gain was always observed regardless of the filter configuration. As can be seen in Figure 7.7, a majority of pixels within that sequence is filtered for all QPs higher than 25. Especially at low bit rates all parts of the frame are successfully modified.

7.4 Chapter Summary

In this Chapter two features of the TTF were investigated. The first part concerned the theoretical considerations conducted in Chapter 4. It has been possible to show that predicted and realized optimal filter lengths strongly correlate for all tested sequences. Even though these all originate from class D of the HEVC test dataset, a similar behavior is to be expected for the high-resolution sequences. This assumption is based on the fact that the filter itself also shows similar characteristics across

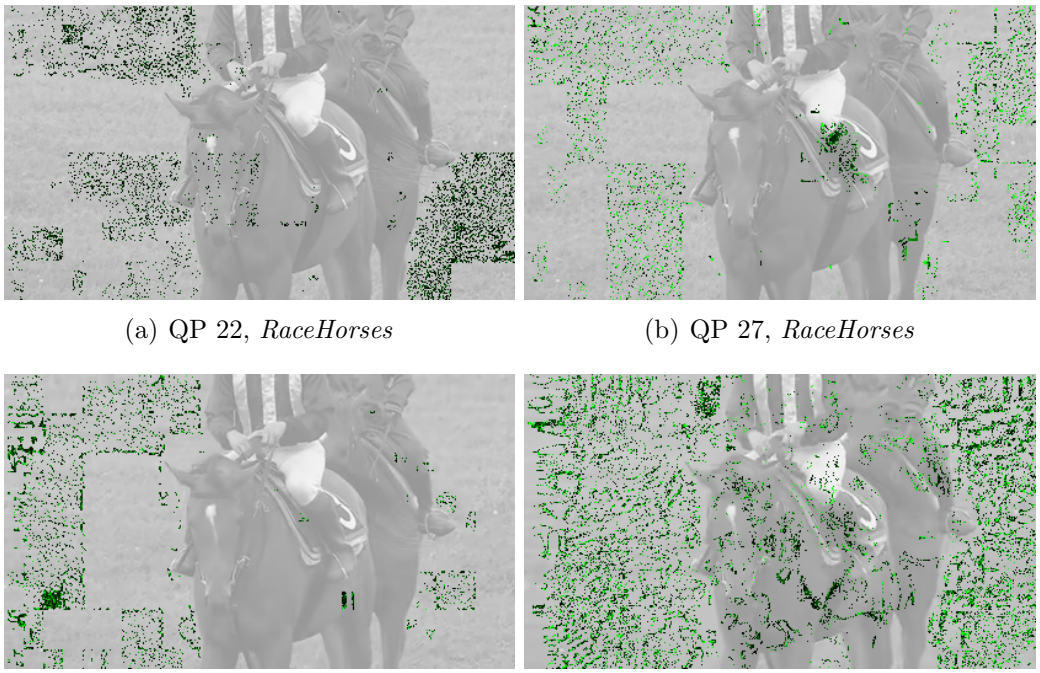
(a) QP 32, *BlowingBubbles*



(b) QP 32, *BQSquare*

(c) QP 32, *RaceHorses*

Figure 7.4: Output frames of the QTTF. Each improved pixel is marked in green. The amplitude of the green color channel (dark to bright green) also reflects the amount of the improvement. All unmarked pixels were not affected by the filter or only minimally influenced.



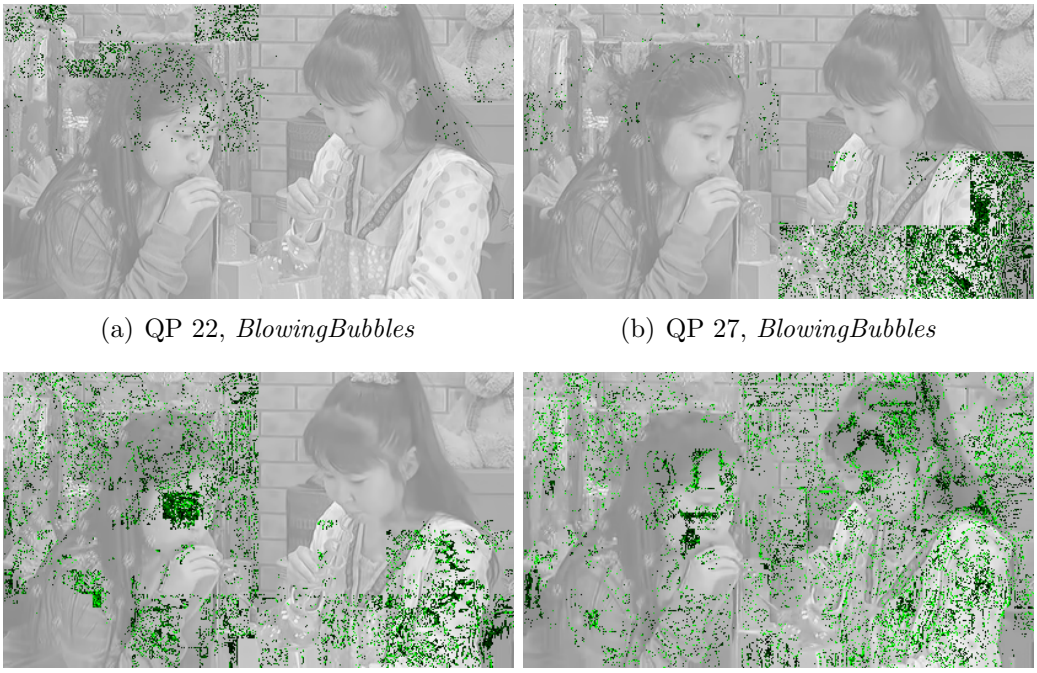
(a) QP 22, *RaceHorses*

(b) QP 27, *RaceHorses*

(c) QP 32, *RaceHorses*

(d) QP 37, *RaceHorses*

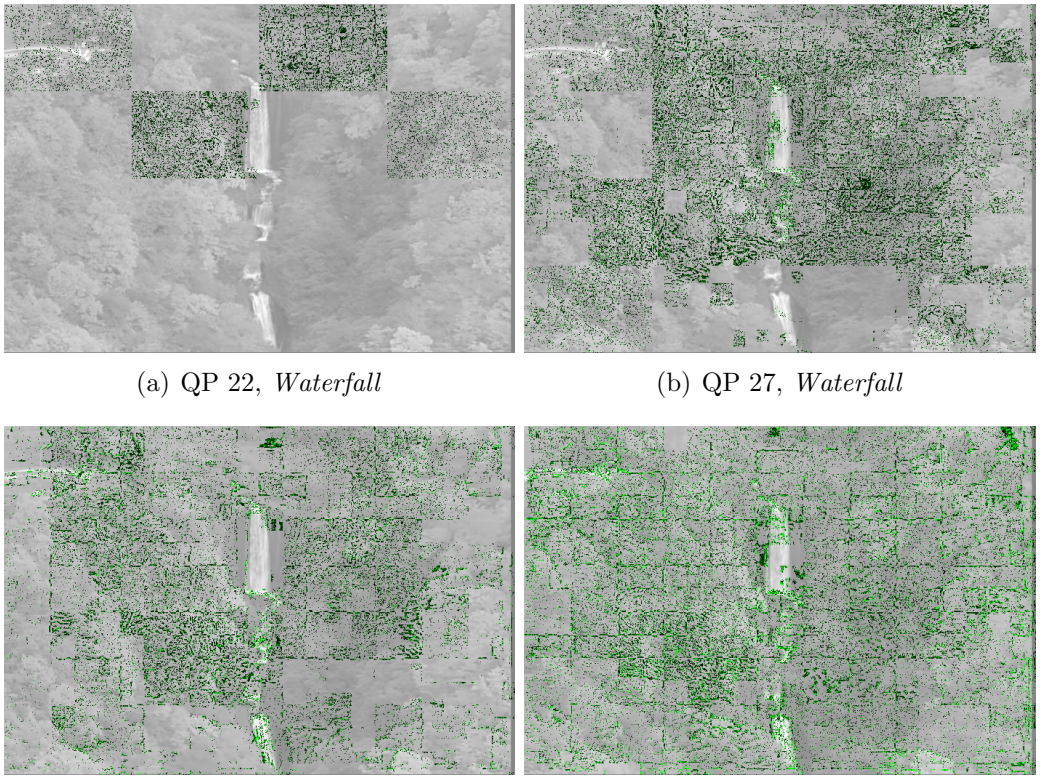
Figure 7.5: Display of filtered and unfiltered pixels for an exemplary frame of the *RaceHorses (D)* sequence for four different QPs.



(a) QP 22, *BlowingBubbles* (b) QP 27, *BlowingBubbles*

(c) QP 32, *BlowingBubbles* (d) QP 37, *BlowingBubbles*

Figure 7.6: Display of filtered and unfiltered pixels for an exemplary frame of the *BlowingBubbles* sequence for four different QPs. The green channel (dark to bright) reflects the achieved improvement per pixel.



(a) QP 22, *Waterfall*

(b) QP 27, *Waterfall*

(c) QP 32, *Waterfall*

(d) QP 37, *Waterfall*

Figure 7.7: Display of filtered and unfiltered pixels for an exemplary frame of the *Waterfall* sequence for four different QPs. The green channel (dark to bright) refelcts the achieved improvement per pixel.

all test classes regardless of spatial or temporal resolution.

Consequently, with the formulae derived in Chapter 4 it is now possible to quickly predict the filters suitability for a certain sequence based on certain spatial and temporal characteristics of a video. Even though the optimal filter lengths are not a parameter that needs to be calculated explicitly, their predictions could for instance be used to build an early termination criterion to speed-up the encoder. Other possible applications include memory management and above all a theoretical assessment of the filter's performance relative to the maximum realizable gain in presence of perfect motion knowledge.

The second part covered the effectiveness of the individual thresholds of the TTF. As expected, the luminance threshold is responsible for most of the realized gain. However, both the temporal and the spatial consistency threshold also justify their existence through their individual contribution to the overall performance of the TTF. As has been mentioned in Section 5.4 the motion vector accuracy of the later HM versions is significantly better than the quality of earlier versions. Since the spatial consistency threshold only performs well with poor motion vector fields, it is subsequently disabled in the final implementation of this thesis based on HM 8.0. Finally, the spatial distribution of the filtered pixels has been investigated for three different sequences. Even though the filter is not switched on for every frame due to RD-optimization, both foreground and background have successfully been filtered in all tested cases.

Chapter 8

Conclusion and Outlook

I think and think for months and years. Ninety-nine times, the conclusion is false. The hundredth time I am right. - Albert Einstein

8.1 Achievements

In this thesis the formation of pixel trajectories from block-based encoded motion vectors as well as their suitability for temporal filtering have been investigated. The temporal trajectory filter has been tested firstly in the H.264/AVC baseline and extended profiles and secondly in several versions of the new standard H.265/MPEG-H. While the results for H.264/AVC were significantly more promising, an acceptable gain could still be produced both for HEVC test models HM 3.0 and HM 8.0.

One achievement, therefore, lies in the proof of concept of a filter that performs temporal filtering on both foreground and background of a sequence and thus both improves the image quality and reduces the required bit rate. In addition, several versions of the filter were implemented and tested in a post-processing scenario.

The second finding in this context is, that the TTF in its current form cannot be used as an unsupervised post-filter. Even though the motion compensated frame shows promising features as a training data set, no gain could be produced without additional side-information. The exact form of the side-information was also examined showing that the amount of side-information is critical when using the TTF as a post-processing filter. The TTF is subsequently best employed in-loop both at encoder and decoder. A quadtree partitioning algorithm, an idea originating from the Wiener-based QALF filter, has also been integrated into the TTF, thus significantly improving its performance.

A third achievement includes the description of an optimal RD-optimized partitioning algorithm for the TTF with an added statistical analysis of the filter flags. The statistical analysis yielded insight into the temporal and spatial behavior of said flags and allowed for the derivation of certain CABAC models to compress the side-information by 47% on average. This compression ratio is quite significant since CABAC usually only produces bit rate reductions in the area of 10 to 20% and underlines the effectiveness of the compression scheme.

In addition, an analytical description of the filter behavior under certain assumptions has been derived. In this context, it has been proven that an optimal filter length does exist for every pixel individually. Even though this length still varies from pixel to pixel, the average optimal trajectory length can reliably be predicted from fundamental video characteristics. Based on these formulae predictions concerning the optimal filter length were made. The accuracy of these predictions could be demonstrated for all tested sequences.

Another achievement here lies in the simple analytical description of the filter based only on three separate input variables: image correlation, temporal noise variance, and maximum motion estimation error. In this context it has been verified that horizontal and vertical correlation in an image are indeed often close to identical, while the motion estimation errors in both directions can differ significantly. Another achievement is the possibility to classify any given video sequence based on the mentioned characteristics. In this manner a powerful encoder could decide whether a sequence is suitable for temporal trajectory filtering or not. In order to reconstruct the true pixel motion, all implementations of the TTF utilize three separate criteria. Despite slight changes over time all three have retained their distinct characteristics: color change and spatial as well as temporal motion consistency.

The introduction and investigation of these three criteria is another achievement of the present thesis. Experiments showed that the color criterion is responsible for around 75% of the filtering gain while the temporal criterion produced much of the remaining improvements. Only for certain sequences the spatial criterion achieved further quality gain.

8.2 Conclusions

The TTF has been extensively studied in this thesis. Its application in both existing and new video coding standards has been tested and validated. Despite the more accurate motion representation available in HEVC compared to H.264/AVC, the filter's performance is decreased when used within the new codec. This loss is mainly due to the increased effectiveness of other prediction and filtering tools in HEVC. For the tested dataset an average BD-rate of -0.4% was observed for the final HEVC reference model thus justifying the investigation of the described filter and possible future extensions.

The filter offers quite good parallelization opportunities even though these have not yet been implemented. For instance, all trajectories could be examined separately to find an optimal *brute-force* solution for the original TTF. In the case of QTTF, a new thread could be started for each subpartition of a given block. Another way to decrease the runtime of the filter would be to store intermediate interpolation results. In the simplest case each frame in the TTF's buffer would be upsampled to a 4-times higher resolution thus making interpolating the same subpel position several times for each new trajectory unnecessary. From the bit rates achieved by HM 3.0 it can be seen, that the TTF works especially well for those sequences that achieve only a less than average compression ratio. For example, both *BQSquare* and *BQTerrace* have significantly higher bit rates than other sequences of the same resolution at comparable quality.

The filter's main drawback, however, is its dependency on accurate side-information that controls the filter behavior. Even with the presented CABAC compression schemes the side-information still restricts the filter's effectiveness. Despite several efforts no temporal or spatial correlation between the three threshlolds T_Y, T_{SC}, and T_{TC} could be established. As has been shown, the *split_flag* for QTTF changes its behavior on a frame-by-frame basis. This is due to foreground objects that move and thus necessitate the recalculation of the entire quadtree for each new frame. If, however, a region-based approach were used, the filter could potentially reuse the *split_flag* structure from previous frames. In this case, however, efficient representations for such regions would be needed.

Another negative aspect of the filter is its sheer computational complexity. Nevertheless, significant advances beyond the state-of-the-art in the areas of temporal filtering of foreground objects, quadtree-compression, and improvement of block-based motion vector fields have been achieved. At low bit rates in particular, the filter consistently produced gain comparable in magnitude to those of the Wiener-based QALF which had previously undergone many years of steady improvements.

In addition, the TTF can now accurately be modeled by a set of analytically derived equations. These will allow future intelligent encoders to only apply the TTF to sequences where long trajectories and thus higher gain can be expected.

8.3 Outlook

A first step to further improve the TTF would, of course, be the reduction of the computational complexity by, for example, using bit shift operations instead of the usual floating point arithmetics. Especially during the recursive search for an optimal parameter combination this would greatly reduce the memory load. Non-exhaustive search methods such as the presented neural network could also be used to find near-optimal settings more quickly. Ideally, such a learning algorithm should be able to derive the TTF's thresholds directly from the image content. As has been shown in Chapter 6, however, the current definition of the thresholds makes finding a good solution with anything other than a *brute-force* search almost impossible. One additional criterion could be the ratio of object size to frame size since smaller objects generally decrease the T_Y value and also require a small (more accurate) temporal consistency threshold T_{TC}.

In addition, asymmetric partitions or even region-based paramter signaling for the QTTF could be investigated. These methods have already been extensively studied during the standardization process of HEVC in the context of motion estimation and motion compensation [31]. They allow the algorithms to better adapt themselves to individually moving image regions of arbitrarty shape. In the context of the TTF these regions have not yet been investigated. It would, however, be possible to use a merging technique to combine square image partitions into shapes of arbitrary size after parameter optimization. However, such a merging step would again need to be controlled by an RD-optimization scheme. Due to runtime restrictions such schemes can only be put into operation when the TTF optimization itself is sped up significantly.

While a post-filter implementation will quite probably not be a likely realizable scenario, other uses within a video encoder are possible. For instance, the TTF could be used before the calculation of the frame residual, which would constitute a combination of loop filter and prediction mode. There are several advantages to such a mode of operation: Firstly, a smaller residual can be expected on average. Additionally, the new prediction mode could potentially even perform better than the regular B-frame as has been pointed out in Section 5.1. Due to the temporal memory of such a prediction mode flickering artifacts could be reduced further, resulting in a much more stable output image. Moreover, the temporal consistency criterion

could be used to predict the location of certain trajectories in future frames. With this ability a temporally smoothed prediction mode similar to the one described by Ohm [40] would be possible. A further improvement of the TTF would be a decoder-side motion estimation scheme to improve the quality of the decoded motion vectors where necessary. Similar to the already mentioned filtering method by Wige et al. [56] the filter could also be applied after a frame has been displayed. In this manner the frames used for ME and MC would probably be better suited for prediction while no unwanted artifacts are introduced in the decoded sequence.

Additionally, combinations with the Global Motion Temporal Filter (GMTF) [20] are also possible. In this scenario, the GMTF tool would be used to process the background of an image and the TTF would be applied to the foreground only. In such a case, where global motion models are available at the decoder, the TTF's vector field could again be improved, while the homographies calculated by the GMTF could also be used to segment a frame into foreground, background and other independently moving regions. In this case, no additional side-information would need to be transmitted to selectively apply the filters to certain regions. Since both filters would now only optimized on certain regions of a frame, the optimization time for TTF and GMTF would also be reduced. One disadvantage of such an approach would, however, be the introduction of new artifacts at the boundaries between the two filtering areas. Everywhere where the segmentation achieves poor results new errors would be introduced unintentionally.

Bibliography

[1] Ce5: 4:4:4 coding. *VT-S305, 19th JVT Meeting, Geneva*, Apr 2006.

[2] M. A. Agostini and M. Antonini. Theoretical model of the coding error in mcwt video coders. In *Proceedings of the IEEE International Conference on Image Processing*, 2006.

[3] J. Barron, D. Fleet, and S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12:43 – 77, 1994.

[4] G. Bjøntegaard. Calculation of average PSNR differences between RD-curves. *ITU-T SG16/Q.6 VCEG document VCEG-M33*, Mar 2001.

[5] C.-F. Chen and K. K. Pang. The optimal transform of motion-compensated frame difference images in a hybrid coder. *IEEE Transactions on Circuits and Systems for Video Technology*, 40(6):393–397, 1993.

[6] T. Chujoh, N. Wada, T. Watanabe, G. Yasuda, and T. Yamakage. Specification and experimental results of quadtree-based adaptive loop filter. *ITU-T SG16/Q.6 VCEG document VCEG-AK22*, Apr 2009.

[7] T. Chujoh, G. Yasuda, N. Wada, T. Watanabe, and T. Yamakage. Block-based adaptive loop filter. *ITU-T SG16/Q.6 VCEG document VCEG-AI18*, Jul 2008.

[8] G. Dane and T. Nguyen. The effect of global motion parameter accuracies on the efficiency of video coding. In *Proceedings of the 11th IEEE International Conference on Image Processing (ICIP)*, 2004.

[9] M. Esche, A. Glantz, A. Krutz, and T. Sikora. Adaptive temporal trajectory filtering for video compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(5):659–670, May 2012.

[10] M. Esche, A. Glantz, A. Krutz, M. Tok, and T. Sikora. Quadtree-based temporal trajectory filtering. *Proceedings of the 19th IEEE International Conference on Image Processing (ICIP)*, 2012.

[11] M. Esche, A. Glantz, A. Krutz, M. Tok, and T. Sikora. Weighted temporal long trajectory filtering for video compression. *Proceedings of the 29th IEEE Picture Coding Symposium (PCS 2012)*, May 2012.

[12] M. Esche, A. Krutz, A. Glantz, and T. Sikora. A novel in-loop filter for video-compression based on temporal pixel trajectories. *Proceedings of the 28th PCS*, pages 514–517, Dec 2010.

[13] M. Esche, A. Krutz, A. Glantz, and T. Sikora. Temporal trajectory filtering for bi-directional predicted frames. In *Proceedings of the 18th IEEE International Conference on Image Processing (IEEE ICIP2011)*, pages 1669–1672, Brussels, Belgium, Sept. 2011. IEEE catalog number: CFP11CIP-USB ISBN: 978-1-4577-1302-6.

[14] M. Esche, M. Tok, A. Glantz, A. Krutz, and T. Sikora. Efficient quadtree compression for temporal trajectory filtering. In *Data Compression Conference*, Snowbird, Utah, Mar. 2013.

[15] M. Esche, M. Tok, and T. Sikora. Adaptive dense vector field interpolation for temporal filtering. In *20th IEEE International Conference on Image Processing*, Melbourne, Australia, Sept. 2013.

[16] M. Esche, M. Tok, and T. Sikora. Theoretical considerations concerning pixelwise temporal filtering. In *Data Compression Conference*, Snowbird, Utah, Mar. 2014.

[17] M. Flierl, S. Member, T. Wiegand, and B. Girod. Rate-constrained multihypothesis prediction for motion compensated video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 12:957–969, 2002.

[18] M. Flierl, T. Wiegand, and B. Girod. Multihypothesis pictures for h.26l. In *Proceedings of the 8th IEEE International Conference on Image Processing (ICIP)*, pages 526–529, 2001.

[19] C.-M. Fu, C.-Y. Chen, Y.-W. Huang, and S. Lei. Sample adaptive offset for hevc. *IEEE 13th Intl. Workshop on Multimedia Signal Processing (MMSP)*, pages 1–5, 2011.

[20] A. Glantz, A. Krutz, M. Haller, and T. Sikora. Video coding using global motion temporal filtering. *Proceedings of the 16th IEEE International Conference on Image Processing (ICIP)*, pages 1053–1056, Nov 2009.

[21] B. Glocker, H. Heibel, N. Navab, P. Kohli, and C. Rother. Triangleflow: Optical flow with triangulation-based higher-order likelihoods. In *11th European Conference on Computer Vision (ECCV)*, Crete, Greece, September 2010.

[22] M. Hazewinkel. *Encylopedia of Mathematics*. Springer, 2010.

[23] C. Igel and M. Hüsken. Empirical evaluation of the improved rprop learning algorithms. *Neurocomputing*, 50:105–123, 2003.

[24] International Telecommunications Union. ITU-T Recommendation H.120. Technical report, 1988.

[25] International Telecommunications Union. ITU-T Recommendation H.261. Technical report, 1993.

[26] International Telecommunications Union. ITU-T Recommendation H.262. Technical report, 1995.

[27] International Telecommunications Union. ITU-T Recommendation H.263. Technical report, ITU-T Study Group 16, March 1996.

[28] International Telecommunications Union. Draft ITU-T recommendation and final draft inernational standard of joint video specification (ITU-T Tec. H.264/ICO/IEC 14496-10 AVC). *Joint Video Team (JVT) of ICO/IEC MPEG and ITU-T VCEG, JVT-G050*, 2003.

[29] A. K. Jain. Advances in mathematical models for image processing. *Proceedings of the IEEE*, 69(5):502–534, 1981.

[30] N. Jayant and P. Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video*. Prentice Hall, Englewood Cliffs, NJ, 1984.

[31] I.-K. Kim, S. Lee, M.-S. Cheon, T. Lee, and J.-H. Park. Coding efficiency improvement of hevc using asymmetric motion partitioning. In *IEEE International Symposium on Brandband Multimedia Systems and Broadcasting*, 2012.

[32] S. Knorr and T. Sikora. An image-based rendering (ibr) approach for realistic stereo view synthesis of tv broadcast based on structure from motion. In *Proceedings of the IEEE International Conference on Image Processing*, volume 6, pages 572 – 575, 2007.

[33] P. List, A. Joch, J. Lainema, G. Bjøntegaard, and M. Karczewicz. Adaptive deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 13(7):614–619, Jul 2003.

[34] C. Liu, W. T. Freeman, E. H. Adelson, and Y. Weiss. Human-assisted motion annotation. *Prodeedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2008.

[35] Y. Liu. Unified loop filter for video compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(10):1378 – 1382, October 2010.

[36] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7):620–636, Jul 2003.

[37] K. McCann, T. Wiegand, B. Bross, W.-J. Han, J.-R. Ohm, J. Ridge, S. Sekiguchi, and G. J. Sullivan. Hevc draft and test model editing. *ITU-T SG16 WP3, ISO/IEC JTC1/SC29/WG11 doc. JCTVC-D500-r1*, Mar 2011.

[38] R. Mester. A system-theoretical view on local motion estimation. In *5th IEEE Southwest Symposium on Image Analysis and Interpretation*, 2002.

[39] A. A. Muhit, M. R. Pickering, M. R. Frater, and J. F. Arnold. Video coding using elastic motion model and larger blocks. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(5):661–672, 2010.

[40] J.-R. Ohm. Three-dimensional subband coding with motion compensation. *IEEE Transactions on Image Processing*, 3(5):559–571, Sep 1994.

[41] J.-R. Ohm. *Multimedia Communication Technology*, chapter 11, 12 and 13. Springer, Heidelberg, Germany, 2004.

[42] K. K. Pang and T. K. Tan. Optimum loop filter in hybrid coders. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(2):158–167, April 1994.

[43] M. M. Sondhi and G. W. Elko. Adaptive optimization of microphone arrays under a nonlinear constraint. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 981 – 984, 1986.

[44] K. Suehring. Key technical area reference modell, http://iphome.hhi.de/suehring/tml/download/.

[45] G. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22:1649–1668, 2012.

[46] A. M. Tourapis. Enhanced predictive zonal search for single and multiple frame motion estimation. *Proceedings of the SPIE Conference on Visual Communications and Image Processing (VCIP)*, pages 1069–1079, Jan 2002.

[47] N. K. Treadgold and T. D. Gedeon. Simulated annealing and weight decay in adaptive learning: the sarprop algorithm. *IEEE Transactions on Neural Networks*, 9(4):662–668, 1998.

[48] D. T. Vo and T. Q. Nguyen. Quality enhancement for motion JPEG using temporal redundancies. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(5):609–619, May 2008.

[49] D. T. Vo and T. Q. Nguyen. Optimal motion compensated spatio-temporal filter for quality enhancement of H.264/AVC compressed video sequences. *Proceedings of the 26th IEEE International Conference on Image Processing (ICIP)*, pages 3173–3176, Nov 2009.

[50] Z. Wang and A. C. Bovik. Mean squared error: Love it or leave it? *IEEE Signal Processing Magazine*, 2009.

[51] 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 – 6012, 2004.

[52] T. Wiegand, M. Flierl, and B. Girod. Entropy-constrained design of quadtree video coding schemes. In *Proceedings of the International Conference on Image Processing and its Applications*, pages 36–40, 1997.

[53] T. Wiegand and B. Girod. Lagrange multiplier selection in hybrid video coder control. *Proceedings of the 8th IEEE International Conference on Image Processing (ICIP)*, 3:542–545, 2001.

[54] T. Wiegand, E. Steinbach, A. Stensrud, and B. Girod. Multiple reference picture video coding using polynomial motion models. *Proceedings of the VCIP*, 3309:134–145, Jan 1998.

[55] T. Wiegand, G. Sullivan, G. Bjøntegaard, 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, Jul 2003.

[56] E. Wige, G. Yammine, P. Amon, A. Hutter, and A. Kaup. Adaptive in-loop noise-filtered prediction for high efficiency video coding. In *IEEE International Workshop on Multimedia Signal Processing*, pages 1–6, 2011.

[57] S. Wittmann and T. Wedi. Transmission of post-filter hints for video coding schemes. *Proceedings of the 25th Picture Coding Symposium (PCS)*, pages 81–84, Sep 2007.

Appendices

Appendix A

Block-Based Error Distribution Analysis

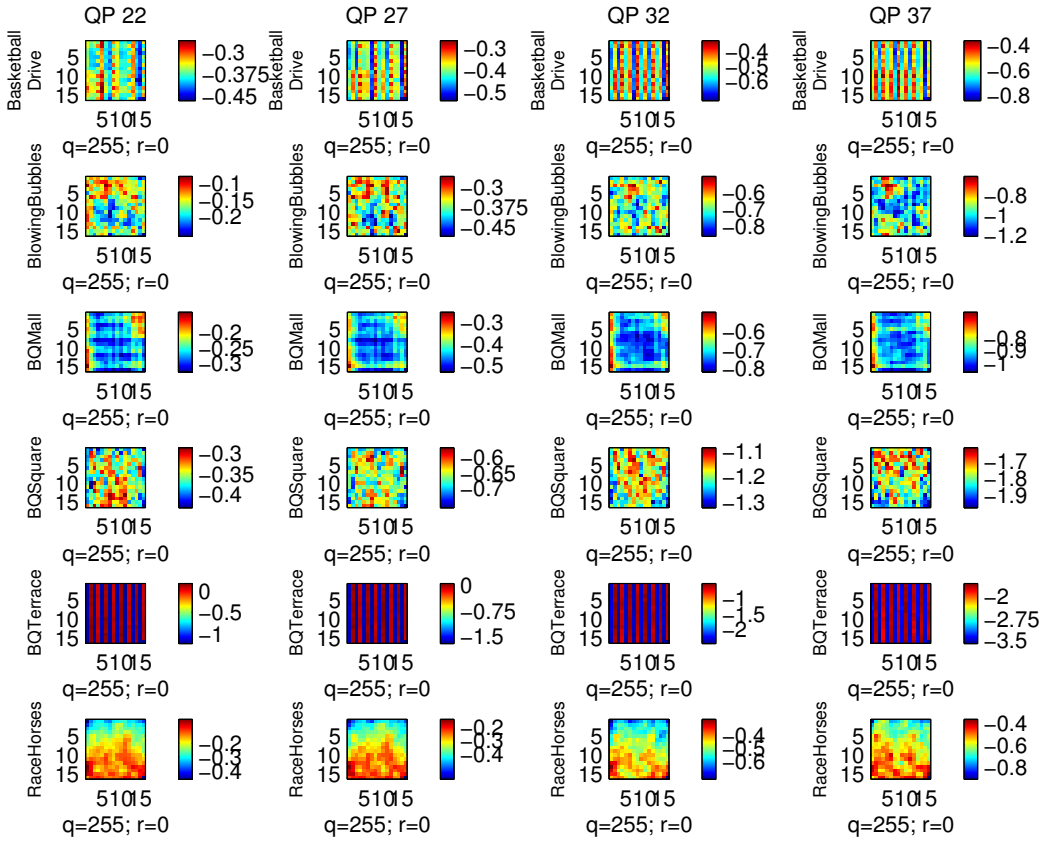


Figure A.1: Average error between the motion-compensated frame Y_{MC} and the original Y_{Orig} within the 16×16 macro blocks.

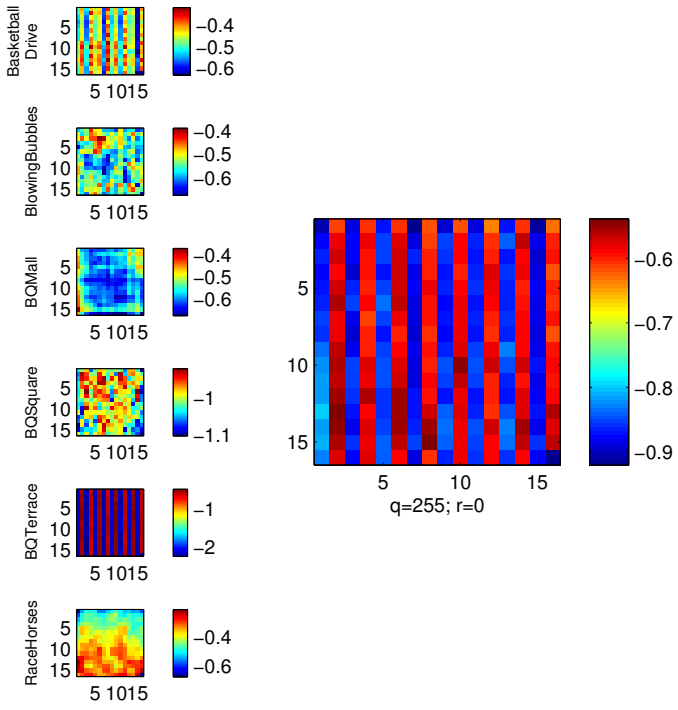


Figure A.2: *Left:* average errors for all four QPs. *Right:* Average over all tested sequences.

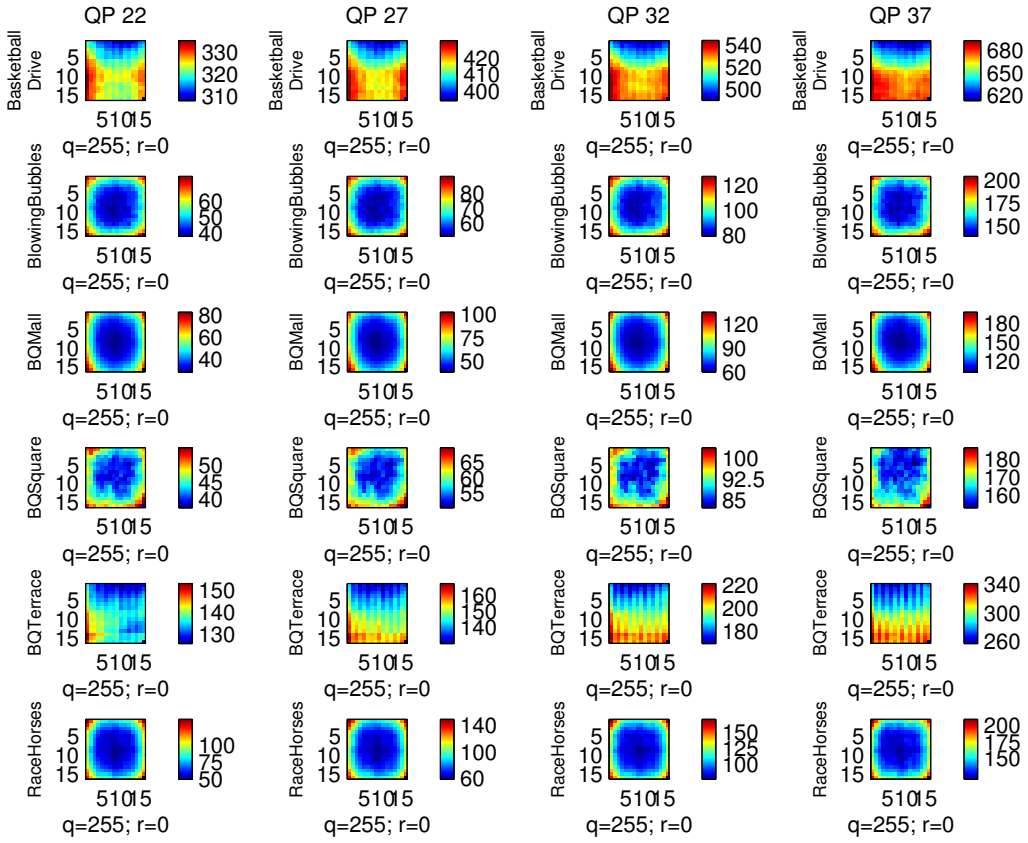


Figure A.3: Power of the error distribution between the motion-compensated frame Y_{MC} and and the original Y_{Orig} within 16×16 blocks.

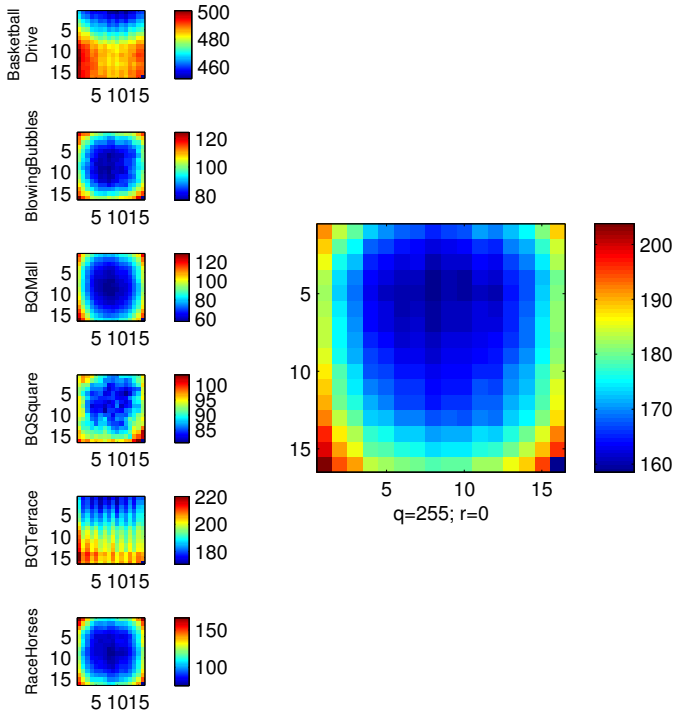


Figure A.4: *Left:* Average over the power distributions over four QPs. *Right:* Average over all sequences.

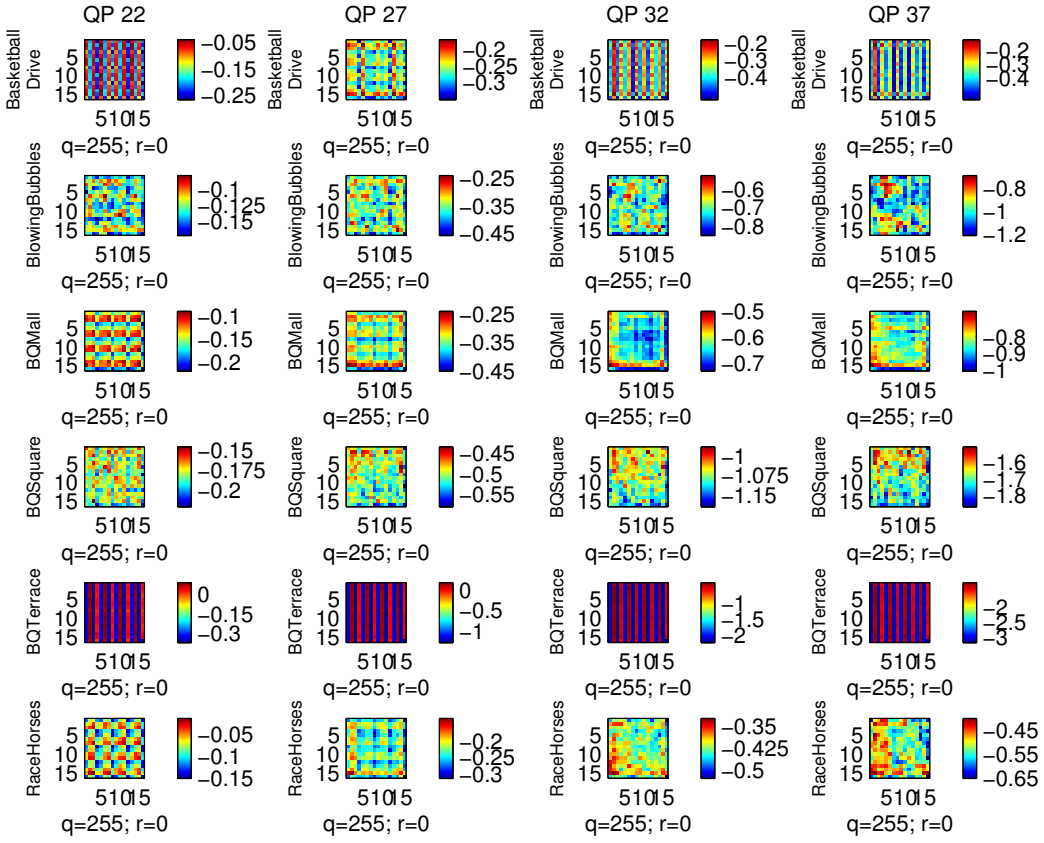


Figure A.5: Average error between the decoded frame Y_{dec} and the original Y_{orig} within a 16×16 block.

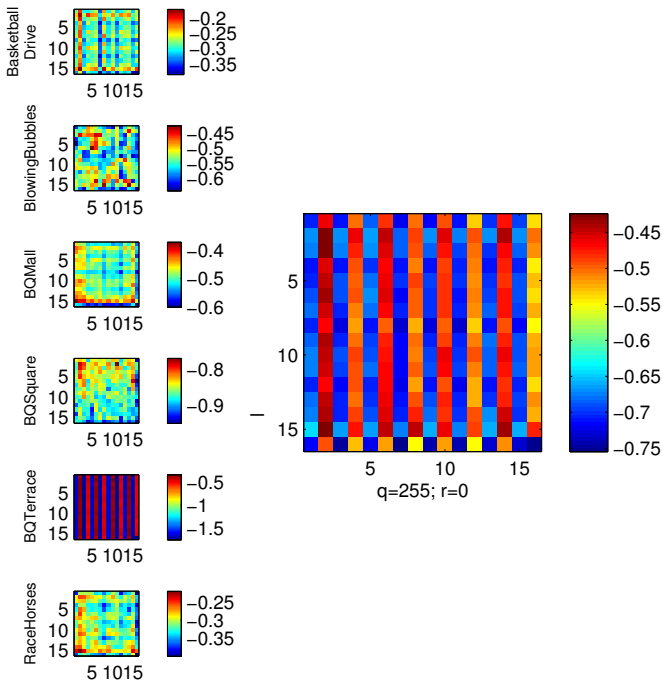


Figure A.6: *Left:* Average over all tested QPs. *Right:* Average over all sequences.

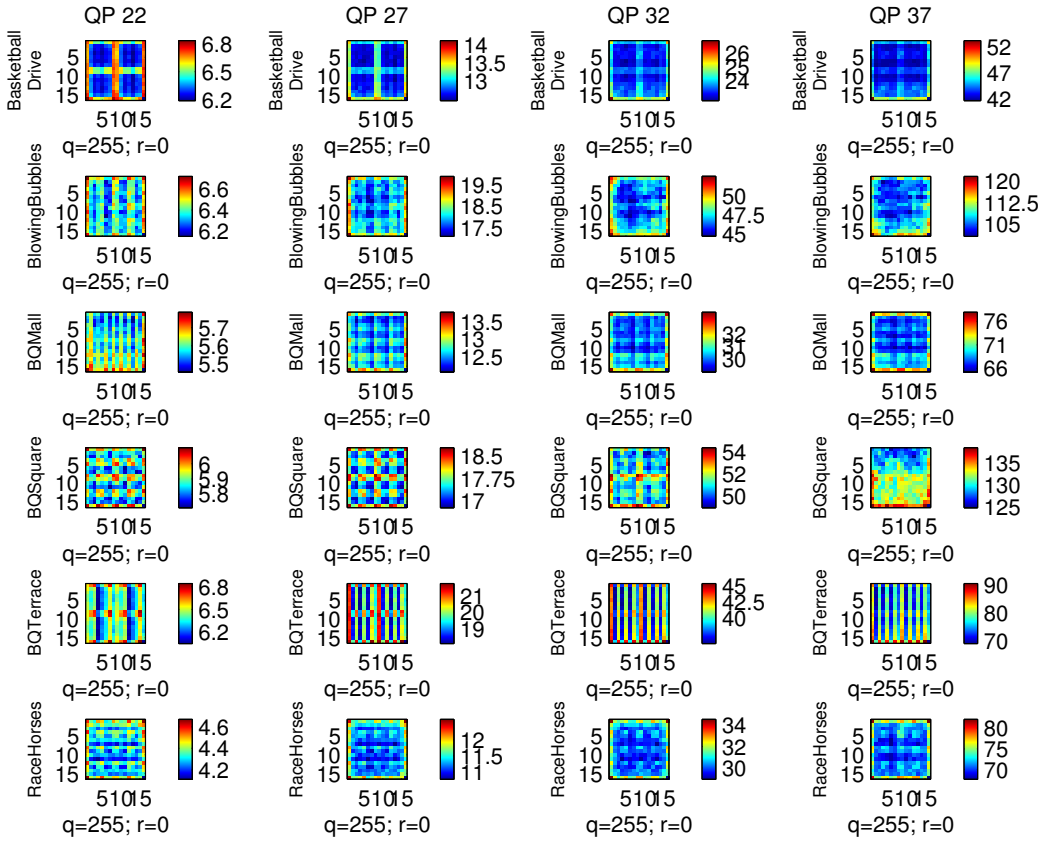


Figure A.7: Distribution of the MSE between the decoded frame Y_{rec} and the original Y_{orig} within all 16×16 blocks.

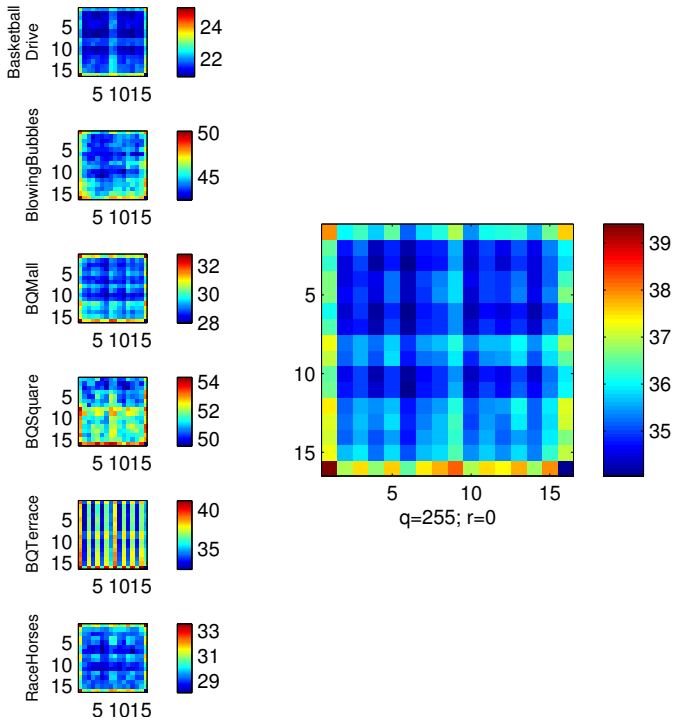


Figure A.8: *Left:* Average over all MSE per sequence over all QPs. *Right:* Average over all sequences.

Appendix B

Implemented Neural Networks

The implemented neural networks and their parameters shall be described here. The configurable network parameters shall be detailed first. The iRprop+ is essentially a gradient-based algorithm whose function to optimize is an error function. The variables of the error function are its weights. As has been described in Section 6.1.2 there are combinations of weights that have no practical use for this application.

In order to reduce the runtime of the algorithm, these combinations are eliminated first. Subsequently, upper limits are imposed on the utilized weights within the ANN. Two variables (NN_UpperBound and NN_LowerBound) are defined for each possible weight of the algorithm. Another bound ensures that small weights tend to be chosen more frequently during the early stages of the algorithm. With an increasing number of iterations higher values can be assigned to the weights. To this end the temperature parameter T from Section 6.1.2 is used. It controls the speed at which a bound moves towards larger weights. A second parameter k_1 individually adjusts the speed of adjustment for each weight whereas k_4 sets the starting poiunt for the boundary. Together these are used to formulate a moving boundary:

$$MB = k_4 \cdot 2^{k_1 \cdot T \cdot n}, \tag{B.1}$$

where n is the number of iterations.

For the SARprop variant of the algorithm the following additional constraints are formulated: Should a parameter λ be smaller than $k_2 \cdot sRMS$ then the λ for the next iteration will be

$$\lambda_{\text{new}} = \lambda \cdot \eta^- + k_3 \cdot r \cdot 2^{-T \cdot n}. \tag{B.2}$$

Here λ is the parameter introduced in [47] that adjusts the weights and r is a random real number between 0 and a. $\eta^- = 0.5$ as in [47]. k_3 is the maximum allowed change

of the weight. k_2 is an additional threshold such that, if the derivative of the weight drops below k_2, only then is a random change of the weight permitted. The $sRMS$ describes a modified version of the root mean squared error:

$$sRMS = \sqrt{\mathrm{MSE}_0} \cdot \frac{\sqrt{\mathrm{MSE}}}{256}. \tag{B.3}$$

The first term MSE_0 is the MSE value measured between the decoded frame and the reference frame. The second term represents a normalized version of the regular MSE. Together both can be interpreted as a normalized error variance which is converted to the standard deviation by taking the square root. The other parameters of the algorithm are its termination criteria:

1. maximum number of iterations
2. minimum absolute value of the error gradient
3. minimum absolute value of a single component of the error gradient

The algorithm stops when the last two criteria are fulfilled.

Appendix C

Spatial and Temporal Properties of the Test Sequences

Sequence	QP	σ_η^2	σ_Y^2	α_x	α_y	$\frac{\sigma_\eta^2}{\sigma_Y^2}$	$q_{x,\max}$	$q_{y,\max}$
BlowingBubbles	18	486.13	1610	0.997	0.998	0.319	4.67	4.67
	20	483.63	1610	0.997	0.998	0.300	4.66	4.74
	22	483.32	1610	0.997	0.998	0.300	4.67	4.32
	24	476.56	1610	0.997	0.998	0.296	5.28	4.28
	26	475.30	1610	0.997	0.998	0.295	5.79	4.59
	28	470.83	1610	0.997	0.998	0.292	6.59	4.45
	30	470.21	1610	0.997	0.998	0.292	4.42	4.53
	32	462.67	1610	0.997	0.998	0.287	4.90	3.81
	34	451.95	1610	0.997	0.998	0.281	4.47	3.53
	36	443.36	1610	0.997	0.998	0.275	3.79	3.10
	38	428.36	1610	0.997	0.998	0.266	4.04	3.46
	40	416.40	1610	0.997	0.998	0.259	3.51	2.74
	42	400.85	1610	0.997	0.998	0.249	3.30	2.75
	44	371.22	1610	0.997	0.998	0.231	3.04	2.48
	46	344.58	1610	0.997	0.998	0.214	2.94	2.60
	48	336.90	1610	0.997	0.998	0.209	2.46	2.86
BQSquare	18	446.13	4465	0.996	0.996	0.100	1.85	2.62
	20	441.53	4465	0.996	0.996	0.099	1.64	2.13
	22	440.04	4465	0.996	0.996	0.099	1.61	2.12
	24	440.01	4465	0.996	0.996	0.097	1.40	1.94
	26	433.45	4465	0.996	0.996	0.095	1.36	2.01
	28	423.31	4465	0.996	0.996	0.094	1.39	1.88
	30	418.11	4465	0.996	0.996	0.096	1.38	1.69
	32	430.63	4465	0.996	0.996	0.093	1.35	1.81
	34	415.36	4465	0.996	0.996	0.088	1.37	1.83
	36	392.83	4465	0.996	0.996	0.083	1.26	1.40
	38	395.86	4465	0.996	0.996	0.089	1.23	1.39
	40	372.38	4465	0.996	0.996	0.083	1.25	1.16
	42	341.58	4465	0.996	0.996	0.077	1.25	1.00
	44	307.61	4465	0.996	0.996	0.069	1.13	0.60
	46	285.22	4465	0.996	0.996	0.064	1.02	0.60
	48	192.68	4465	0.996	0.996	0.043	0.96	0.59

Table C.1: Spatial and temporal properties for *BlowingBubbles* and *BQSquare*.

Sequence	QP	σ_η^2	σ_Y^2	α_x	α_y	$\frac{\sigma_\eta^2}{\sigma_Y^2}$	$q_{x,\max}$	$q_{y,\max}$
RaceHorses (D)	18	583.74	1600	0.995	0.996	0.365	6.96	6.71
	20	671.13	1600	0.995	0.996	0.363	6.96	6.96
	22	670.60	1600	0.995	0.996	0.361	6.60	6.64
	24	667.33	1600	0.995	0.996	0.361	6.96	6.66
	26	666.50	1600	0.995	0.996	0.362	6.60	6.59
	28	663.00	1600	0.995	0.996	0.363	6.96	6.62
	30	665.31	1600	0.995	0.996	0.363	6.60	6.62
	32	664.23	1600	0.995	0.996	0.351	6.60	6.92
	34	652.91	1600	0.995	0.996	0.352	6.96	6.79
	36	640.04	1600	0.995	0.996	0.340	6.96	6.62
	38	627.03	1600	0.995	0.996	0.347	6.96	6.00
Waterfall	18	740.04	735	0.998	0.990	1.007	0.65	3.09
	20	735.94	735	0.998	0.990	1.000	0.66	3.13
	22	735.90	735	0.998	0.990	1.001	0.64	3.13
	24	729.25	735	0.998	0.990	0.992	0.64	3.16
	26	724.18	735	0.998	0.990	0.985	0.63	3.21
	28	721.10	735	0.998	0.990	0.981	0.65	3.01
	30	721.51	735	0.998	0.990	0.982	0.62	3.03
	32	714.65	735	0.998	0.990	0.973	0.67	3.16
	34	716.80	735	0.998	0.990	0.974	0.67	3.13
	36	706.35	735	0.998	0.990	0.960	0.69	3.31
	38	705.38	735	0.998	0.990	0.959	0.73	3.17
	40	691.92	735	0.998	0.990	0.940	0.72	3.00
	42	676.81	735	0.998	0.990	0.921	0.75	2.81
	44	668.28	735	0.998	0.990	0.909	0.85	1.67
	46	656.25	735	0.998	0.990	0.893	0.85	1.53
	48	639.34	735	0.998	0.990	0.869	0.93	1.40

Table C.2: Spatial and temporal properties for *RaceHorses* and *Waterfall*.