Study on Fast Affine Motion Parameter Estimation for Efficient Video Coding

Study on Fast Affine Motion Parameter Estimation for Efficient Video Coding

Antoine Chauvet

February 2019

School of Engineering
Tohoku University

Abstract

The aim of this thesis is to explore new ways to make efficient encoding faster and more practical, through two aspects. First, we look at improving the reconstructed picture through postfiltering, but with a processing speed that is acceptable for realtime uses. Second, we show different and so far unexplored ways of determining affine motion parameters in Affine Motion Prediction for video encoding. The speed improvement comes from a faster parameter estimation algorithm and affine mode skip heuristics to avoid computing affine parameters for every block.

In the review of the literature, we show that existing methods are very slow, which make them hard to include because of the associated cost. Postfiltering methods using Neural Networks take several seconds to process a single frame, which makes real time processing impossible. Most Affine Motion Prediction methods use a gradient-based approach for estimating the parameters. It requires often many iterations to converge towards the best value, and in proposed papers, evaluates affine motion parameters for every block.

Optical flow, designed to find the true motion between images, has been seldom used for video coding, as most papers focus on achieving very high accuracy in their estimation, which comes at the cost of a very slow processing. The frame interpolation application could be used for video coding, but it places a high burden on the decoder which has to compute the optical flow. However, some optical flow methods trade off some accuracy for speed and could be used for video coding.

In the present thesis, we present two improvements for video coding: a fast postfilter that uses shallow neural networks and a fast optical flow-based estimation of affine parameters. The postfilter is able to improve the decoded picture while maintaining real-time processing. The fast estimation is based on segmentation of blocks and classification into three categories: blocks that should use translation, blocks that could benefit from affine prediction, and blocks that should be split up. These heuristics allow avoiding the parameter refinement process when affine prediction is likely to be worse than translation.

Abstract

We found that using the proposed approach, we can achieve a reduction in the affine parameter estimation time by half, while keeping very close to the efficiency of the state of the art methods. When combined with the proposed postfilter, it is able to improve the end result, and on some sequences is more efficient than the existing methods.

Contents

List of Figures

List of Tables

Acknowledgments

This thesis would not have been possible without the support and guidance from many people around me.

I would like to particularly thank the professors in my laboratory that have provided me with a lot of insightful guidance over the years, Professor Shinichiro Omachi, Associate Professor Yoshihiro Sugaya, and Assistant Professor Tomo Miyazaki. During my research presentations, they asked me many challenging questions that made me think about what direction to give my research at times where I was struggling. Their feedback during my five years period in Omachi-Sugaya laboratory was very helpful. Professor Omachi supported my idea to investigate affine motion compensation in video coding, which am I very grateful for.

My deep gratitude also goes to Professor Toshiyuki Yoshida and Mr. Chihiro Tsutake, from University of Fukui, for answering some questions I had about their method and providing me with the source code of their implementation so that I could more easily implement my method and compare it to theirs.

I would also like to thank my thesis committee members, Professors Akinori Ito and Satoshi Shioiri, for their feedback during the last spurt of my work on this thesis and giving me valuable ideas for completing it.

I am thankful to my labmates, especially Kento Tonosaki, who as my tutor helped my a lot when I was starting in the lab. I have learned a lot from all of you, and have been able to have often a great time when coming to the lab and in the events you organized outside. While I have been in the laboratory long enough to have trouble remembering the names of everyone, I have not forgotten about the happy times we shared.

I am very grateful to my family that supported my choice to go to Japan even though it would mean I would rarely be able to see them, their continuous support despite the distance and the many care packages sent from France that helped me through difficult times.

Lastly, I would like to thank my wife, Ai Narita, for her continuous love and support during those three years. I do not think I would have been able to complete with thesis without her by my side.

Glossary

AMP Affine Motion Prediction. vii, 9–13, 16, 17, 19, 34

AMVP Affine Motion Vector Prediction. iv, 34, 35, 54, 55

AVC Advanced Video Coding. 1, 3–5, 10, 19, 24

BD-R Bjøntegaard-Delta Rate. 18–22, 36–38, 46, 47, 51

Block Matching Algorithm for finding similar blocks to the current one for motion estimation purposes. iii, 10, 13, 46

CABAC Context Adaptive Binary Arithmetic Coding. 5, 6, 16

Chroma Cb and Cr components in the YCbCr color space, representing colors. 3

Chroma Subsampling Action of increasing the sample period on the chroma components of a signal to reduce its size. 20

CNN convolutional neural network. 20, 21, 58

CP Control Points. 10–13, 15, 16

DCT Discrete Cosine Transform. 2

HEVC High Efficiency Video Coding. 1, 3–8, 10, 19, 23, 24, 26, 32–34, 36, 37, 58

ITU-T International Telecommunication Union Telecommunication Standardization Sector. 1

JCT-VCT Joint Collaborative Team on Video Coding. 1, 8

JEM Joint Exploration Model. 2, 10–12, 19, 20, 33

JVET Joint Video Exploration Team. 2

JVET Joint Video Experts Team. 2, 10

Luma Y component in the YCbCr color space, representing brightness. 3

Motion Estimation Process of determining motion vectors. 8–10, 12, 14, 15, 17, 18, 29, 33, 42

MPEG Moving Picture Experts Group. 1, 2

Motion Vector Offset between the samples used for prediction in the reference picture and the samples in the current picture. iii, 4–6, 8–17, 19, 20, 22, 26, 32–35, 55, 56

MVD Motion Vector Difference. iii, 8, 9, 16, 55

MVP Motion Vector Prediction. iii, 8–10, 15, 16, 34, 55

OMBC Overlapped Block Motion Compensation. iii, 20

Optical Flow Movement between two frames. iv–vi, 21, 22, 26–29, 31–34, 42, 46, 47, 58, 59

PSNR Peak Signal to Noise Ratio. vii, 7, 36–38, 42–44, 47

PU Prediction Unit. 18

QP Parameter defining the quantization level used for a block. 21, 37, 46

Rate Distortion Optimization Process that evaluates the cost of coding with a given mode by simulating the encoding fully. 17, 18, 22, 53–55

Reference Picture Picture from which samples are sourced for inter-prediction. 4, 15, 26, 27, 29, 30, 58

Residual Error that remains after the prediction step. 2, 4, 6, 9

Reference Picture List List of pictures that can be used as reference by the current picture. 33

SAD Sum of Absolute Differences. 9, 17, 32

SATD Sum of Absolute Transformed Differences. 9, 17, 19

SVM support vector machine. 18

Target Picture Picture that is being predicted. 4, 26, 27, 29, 30

TZSearch Test Zonal Search. iii, 8–10, 14

VCEG Video Coding Experts Group. 1, 2

VVC Versatile Video Coding. 2–4, 10–12, 19, 34, 55, 58

Zoom & Rotation 4-parameter model that represents translation, zooming and translation. 11, 13, 30, 43

1 Introduction

1.1 Context

Video has become ubiquitous in the current world. In a few years, it moved from the analog era with analog TV and VHS tapes to the digital era with digital television, DVDs and now High Definition. Outside of this change in quality, the latest years have seen a change in how people view videos. While before the only sources of video were TV broadcasts and tapes, it is now possible to watch anything we want to using Internet streaming platforms such as Netflix, Hulu, etc. This created a large peak in bandwidth usage for video as instead of one broadcast for many people, now each person gets their own broadcast. Studies show that video already uses over three quarter of the total bandwidth on the Internet, and this ratio is rising. In a white paper [1], Cisco says that in 2017, IP video traffic made 75% of the total IP traffic, and they forecast it will increase to 82% by 2022.

Because of this, encoding systems designed to handle this large increase of data have become more and more important over the latest years.

Because it is important that people agree on the video encoding formats, standardization groups such as the International Telecommunication Union Telecommunication Standardization Sector (ITU-T), assemble experts in video coding to define new standards that everyone will use. Two of the biggest groups of experts are the Moving Picture Experts Group (MPEG) and Video Coding Experts Group (VCEG), that have in recent years joined forces to form the Joint Collaborative Team on Video Coding (JCT-VCT). They have defined the most widely used video coding standards we use now for playing DVDs, watching digital TV or streaming videos over the Internet. The latest standard to have been released by the ITU-T is High Efficiency Video Coding (HEVC) [2]. It was designed to handle bigger video than Advanced Video Coding (AVC), with support for up to 8K video clearly defined into the standard. It also provides a more efficient encoding with an efficiency approximately doubled on average over AVC [3].

In 2015, MPEG and VCEG agreed to work together again on a new video coding standard and formed the Joint Video Exploration Team (JVET). They created the Joint Exploration Model (JEM), a new reference encoder that evaluate proposed changes. They started with a Call of Evidence and followed it with a Call for Proposals[4]. JEM design is modular, allowing to evaluate the efficiency of each proposal independently, as did [5]. For each proposal they evaluated the encoding gains and the added complexity in encoding and decoding time. These proposals formed the basis of the new encoding standard that is currently being developed, Versatile Video Coding (VVC). JVET was renamed Joint Video Experts Team (JVET), to keep the acronym. Not all proposals were accepted, as some put too much burden on hardware implementations, or increase decoding complexity too much. With the number of devices playing video at an all time high, keeping decoding complexity reasonable is critical for easier adoption of newer standards.

1.2 Video Encoding process

To encode a video efficiently, many techniques must be used. This section will first go over the techniques used for any video coding before going into more depth about some specific techniques relevant to this thesis.

A video is composed of frames, or pictures, that are related to each other. Video coding, in contrast to still picture coding, uses this feature to reduce the size of the compressed output while maintaining a good visual quality. Each frame is divided into blocks, and for each block a decision is made. The block is predicted from either a constant value, values of a neighboring block in the same picture or values from a different already coded picture. The first two cases do not require previous frames to be known, so it is always used for the first picture in a sequence, and often at periodic positions to allow moving forward in the video without having to decode every single picture. The latter case is an addition of video coding, called motion compensation or inter (frame) prediction, with the former called in contrast intra prediction. The residual is the error that remains after prediction. Typically, the aim of prediction is to make the residual as small as possible. The next step is a transform, typically something based on the Discrete Cosine Transform (DCT), which aims to put the low frequency components of the residual in the top left corner of the matrix while the high frequency components are moved to the bottom right. The next step is quantization, which introduces errors on purpose to reduce the size of the coding. Instead of storing the precise value, values

are made smaller and lose precision, with high frequency components losing the most precision. The last step is entropy coding, which has to code all the decisions and the values after quantization in a lossless way.

1.2.1 Block division

In older coding standards, such as AVC, there was very little flexibility for block sizes. However, with HEVC, blocks can be of various shapes and sizes, and have their maximum size extended. It is also important to note that while predictions and transforms are both block-based, they can use a different block partitioning. In contrast to earlier video coding standards as well, chroma and luma samples do not have to use the same blocks. VVC goes even further by allowing even larger blocks and more flexibility to splits. This gives a great improvement in efficiency, at the cost of additional complexity for coding the splitting and also requires to evaluate more possibilities when deciding the most efficient encoding of a given picture. Each standard has also a requirement that blocks have to be encoded in a certain order. This is necessary to allow blocks to use neighbors are reference, so blocks on the left and above the current block have to be encoded first. Even when flexibility for block sizes is allowed, only splits inside the largest possible blocks are allowed, and they are all coded in order from top left horizontally. Within the largest possible blocks, a specific order is also required for the subblocks, to ensure the availability of neighbors for prediction.

1.2.2 Intra prediction

When no other blocks to use for reference are available, DC prediction is used. It simply uses one value and sets every pixel in the block to it. Outside of specific cases like black bars in the video, it is not very precise and usually results in a large residual. The other prediction modes use a direction and copy values from already coded blocks in the same frame. Because blocks are coded in a specific order, only blocks on the left and top of the current block can be used. Possible directions are mandated in the standard, and their number has increased in more recent ones. In HEVC, the number of possible directions is 33, which is increased to 65 in VVC.

1.2.3 Inter prediction

Inter prediction has known some very significant changes compared to the earlier standards. For example, H.261 only allowed to predict from a previous frame using an integer displacement. This was very fast since the operation only required to copy bits without any other processing, but was not very precise. Newer standards have allowed for more precise motion vectors (MVs), from full pixel precision to 1/16 sample in VVC. Another improvement is allowing prediction for two reference frames for the same block, also called bi-prediction. In AVC and more recent standards, more pictures are kept in the decoded picture buffer, which allows using different reference frames for each block and choosing the most appropriate. This requires additional memory, which is often an issue for hardware decoders, which made for example using the full 16 reference frames in AVC very costly for embedded hardware.

This prediction can be represented using the following equation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{1.1}$$

where $\begin{bmatrix} x' & y' \end{bmatrix}^t$ represent the coordinates of the points on the reference picture, $\begin{bmatrix} x & y \end{bmatrix}^t$ the coordinates of the target picture, and $\begin{bmatrix} v_x & v_y \end{bmatrix}^t$ the motion vector.

1.2.4 Transform and quantization

In most cases, prediction (either intra or inter) is not perfect, and some errors remain. This error is called residual. The encoder has two options: either ignore the error and code the image using just the prediction, common in low bitrate situations for bi-predicted blocks that are "good enough", or code the error in some way to allow a better reconstruction.

In some coding standards, like HEVC, it is possible to code the residual directly, using a lossless coding. In the most common cases, however, lossy coding is used. The transform aims to preserve to change the representation of the residual from the space domain to the frequency domain. By doing, so this isolates the low and high-frequency components.

The next step is called quantization, and aims at reducing the cost of coding the

residual by reducing the number of possible levels. For example, a parameter could see its range reduced from $(0, 100)$ to $(0, 10)$, reducing precision but requiring less bits to code the value. Quantization typically reduces precision of high frequency components more than low frequency components. It is common for many values to be reduced to zero, reducing greatly the coding cost.

1.2.5 Entropy Coding

The encoding modes, MVs and transformed residual values must be coded losslessly for the decoder to read them and decode the video doing the opposite process as the encoder. This coding must be efficient if one wants to obtain an efficient coding standard. Entropy coding works by attributing a probability to each possibility and defining a code that is efficient for this probability. A coding scheme is efficient if it gets as close as possible to Shannon's source coding theorem defined as follows:

$$N = -\log_2(P) \tag{1.2}$$

where N is the number of bits required for coding a symbol with probability P.

The most common and easy to implement variable length coding is Huffman coding, which has to use an integer number of bits for each symbol, so unless each symbol has a probability that will result in an integer in Equation1.2, it will not be the most efficient.

Arithmetic coding is similar, but can be seen as being able to code symbols with non-integer code lengths, with some limitations for ease of computation. It can be visualized as calculating the Huffman coding of a whole sequence of symbols at once. As the length N of the message grows, the loss from having to keep an integer size decreases.

Encoding Mode Coding

In recent standards such as AVC and HEVC, Context Adaptive Binary Arithmetic Coding (CABAC) is used. Its main features are as stated in the name, context adaptive and binary arithmetic. Context adaptivity is a very important feature in recent encoders, and a very large source of encoding efficiency gains. Each symbol has an associated probability with it, which can change depending on the actual occurrence of the symbol. For example, if the DC prediction mode is used very often, the encoder will "learn" about it and increase the probability of DC prediction, lowering the cost of coding it.

The learning is a very simple process, so it uses very little additional processing power, as it only has to count the number of times a symbol is used.

Using binary coding helps greatly for the structure of the entropy coder. The encoding mode decisions can be seen as a binary tree, with each branch having a weight representing its probability. In HEVC, the tree structure for blocks can be easily represented as a succession of binary decisions, such as split/no_split, split1/split2, etc.

For values that do not make sense as a binary decision, such as MVs values, the symbol is binarized so that it can be encoded by the CABAC encoder. In most cases, unary coding is used. Probability is still adaptive for those cases, so in the case previous values were big, coding can switch to a different mode representing larger values for less bits. Golomb-Rice codes are very useful for this purpose, as it is possible to change the coding size depending on the average value of the value being coded. However, Golomb-Rice coding only works if the symbol follows a geometric distribution. If the symbol follows a uniform distribution, fixed-length coding would be the most efficient.

Transform Residual Coefficients Coding

Coding the residual coefficients requires a very different approach than the encoding modes. Residual coefficients are encoded from top left, where the most important low frequency components are, to bottom right, where the least important coefficients are. The encoder follows a diagonal, or zigzag pattern with increasing distance from the top left. Coding zeroes can be done very cheaply using run-length encoding, as in the JPEG image format.

Modern coding standards use more advanced coding tricks, that can achieve even higher efficiency. For example, it can adapt the coding according to the probability distribution of the coefficients by using Golomb-Rice code and varying the Rice parameter. Reference [6] shows the details of the implementation of the coefficient coding in HEVC.

1.3 Video Coding Artifacts

In most practical applications, the video encoding is not lossless, since the required bitrate would be too important. Quantization is the step that introduces errors by making coefficients smaller, reducing greatly the coding cost. Some errors will appear

in the reconstructed stream. In most applications, the quality, through Peak Signal to Noise Ratio (PSNR) control, can be maintained high. However, in many situations, such as videoconferencing, the available bitrate can become very low. In those situations, the defects from video coding appear so clearly that everyone notices them, while they were usually impossible to notice even to a trained eye at higher bitrates.

Dividing the picture into blocks has one major disadvantage, as it creates boundaries that can be visible as there are no continuity guarantees between two blocks. For this reason, many encoders and decoders try to maximize visual quality and hide those artifacts. The easier way is to reduce the resolution, either through encoding a smaller size in the first place, or applying blurring to the output at the decoder stage. This is usually not desired because it removes the details of the image. The main way that is being used now is the application of various filters that are trained to remove the encoding artifacts. Some are now in video coding standards like HEVC, which includes a deblocking filter and sample adaptive offset filter. Many other people have proposed filtering techniques to be applied on the reconstructed picture, with many using neural networks in the recent years.

Another approach for reducing blocking artifacts is to limit the amount of block boundaries, which is possible in recent standards as blocks become bigger. However, the motion is not always constant for a larger section of the image, making smaller blocks necessary for accurate prediction. This is where high order motion models shine, representing complex motions that translations are not suited for. For example, using an affine model for motion allows representing camera motions like zooming and rotating objects very accurately, eliminating the need for translation approximations in many small blocks.

2 Related Methods

2.1 Standard Motion Prediction

In video coding standards up to HEVC, the only prediction available was the translation. Finding optimal MVs is already a complex problem with only two dimensions. While performing Motion Estimation (ME) over all possibilities would yield the best solution, it is also very slow, especially as newer methods allow finer level precision, with up to 1/16 sample. For fast computation, various methods have been used.

For efficient coding of the MVs, the standard mandates the use of Motion Vector Prediction (MVP) and Motion Vector Difference (MVD).

The standard does not mandate a specific method as any encoder can find motion parameters the way it wants, but the reference encoder HM [7], released by the JCT-VCT, implements a fast ME method called Test Zonal Search (TZSearch).

2.1.1 Motion Vector Prediction and Motion Vector Difference

The use of MVP is critical for efficient video coding, as it allows entropy coding to work much more efficiently. In most cases, the MV of a block will be similar with a neighboring block, which is already coded. Since we know this MV already, we can use it to allow for a differential coding. In HEVC, MVPs are generated using the neighboring already decoded MVs. Then, we use an index to tell which MV we want to use as predictor. When coding the MV, only the difference between the MVs is coded, $mv_d = mv - mv_p$, where mv is the MV we want to code, mv_p the MV of the block we predict from, and mv_d the coded MVD.

2.1.2 The Test Zonal Search (TZSearch) algorithm

Motion is often similar in the blocks around the current block, which makes MVP very efficient for entropy coding as only MVD are coded, but also for ME, since the algorithm can start with a good initial estimation. A good starting point is to take the median of available MVs from blocks on the left, top, and top-right from the current block. Then, the encoder will compute the transform with this starting estimation and 8 points around it, in a diamond or a square pattern. To evaluate which is the best transform quickly, Sum of Absolute Differences (SAD) is used. Using x for the value of the sample and x' for the predicted value, it can be represented as follows:

$$SAD = \sum |x - x'| \tag{2.1}$$

After the new best estimation is found, the process is repeated, halving the step size when the best value is the same as the previous best. For performance reasons, the ME process always starts with integer-valued MVs, as it does not require interpolation of pixels which would take more time. In the later stages, before deciding on the best transform to add to the list of candidate modes, Sum of Absolute Transformed Differences (SATD) is used. It is similar to SAD but performs a transform on the residual, and is able to measure more accurately the cost of coding the residual.

This process has been tweaked and improved in various ways [8–10]. For example. References [8, 9] proposes to search in 6 directions using an hexagonal pattern instead of the diamond or square pattern. Reference [10] proposes terminating ME early when some conditions are fulfilled, leading to a decrease of half of the encoding time.

2.2 Affine Motion Prediction

Affine Motion Prediction (AMP) has been the subject of significant research. For a long time, it has been apparent that the standard block-matching approaches provide a limited precision for predicting motion in the picture, as it was shown in [11]. However, proposals failed to gain enough support to be included in standards because of the complexity increasing too much. There has been many representations and coding methods for higher motion compensation, with very different complexity and efficiency. The canonical form, coming from mathematics, uses a matrix to represent the coefficients.

However, it is also possible to represent the movement using Control Points (CP), which was demonstrated in [12]. While the most common models use 4 or 6 parameters, [13] shows that there are many models that can be used for prediction, with up to 12 parameters being tested. However, while more parameters can offer greater precision, each additional parameter provides a smaller benefit for representing movement accurately, but incurs the same overhead for coding. Because of this, models with fewer parameters are overall more efficient.

There has been a large increase in papers suggesting some sort of AMP since HEVC. The most likely reason is the increased potential of affine motion prediction compared to previous standards. While older standards like AVC were limited to prediction blocks of 16×16, HEVC allows larger blocks, going up to 64×64. When coding a larger block, the additional cost from coding the affine parameters is relatively smaller and more justifiable when the prediction accuracy improves. Furthermore, while a constant displacement is typically a good approximation for a smaller block, this falls apart with larger block sizes. In [14], the efficiency gains are compared for various block sizes, and at least in this experiment, it is clear that when limited to small block sizes, affine prediction offers limited efficiency gains, which would not be enough to be worth the added complexity. In [13–15], a gradient-based approach is used to find the parameters. In [16], the authors use a different method closer to classical Block Matching (BM) techniques. It is similar to the Test Zonal Search algorithm used for the translation ME.

During the Call for Papers of JEM, several papers were discussed concerning AMP, each bringing something new. Reference [12] proposed the Control Points (CP) representation with 6-parameters with a way to code the MVs with the entropy coding of HEVC. Reference [17] proposed a new merge candidates that also allows using AMP in HEVC. Reference [18] improves on [12], reducing complexity by reducing the parameters to 4, and revising the iteration method for reducing its complexity. Another major change was stopping to compute the MV for each sample, but instead expanding the granularity to 4×4 blocks, reducing the decoder complexity. As it had a lower impact on decoding time than others methods while offering good performance, it was proposed to JVET [19] and was adopted in JEM. When the VVC project started, a modified version of [19] was included in the standard, which will likely still change further until finalizing. Further work has continued, with [20] reintroducing the 6-parameter affine transform and improving the MVP candidates. The 6-parameter affine transform is an additional option, with the encoder being able to choose which transform to use and disable them on a frame basis if it is not useful or wants to skip evaluating the additional modes.

2.2.1 Transform Representation

The two most commonly used models for AMP are the Zoom & Rotation (Z&R) model and the affine model, which use respectively 4 and 6 parameters. The canonical representation of these transforms is given by the following equations:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{2.3}$$

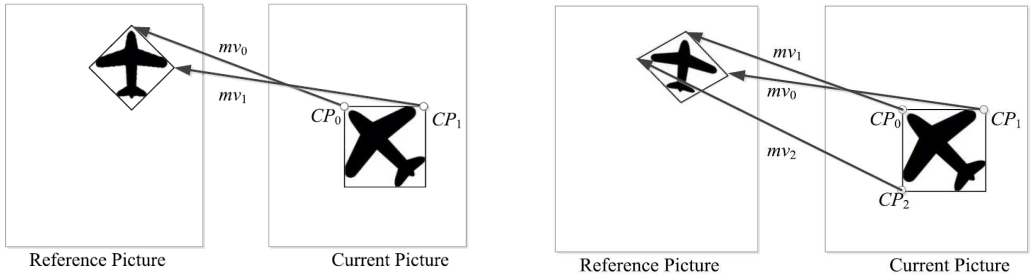
where a, b, c, d are the affine parameters. The Zoom & Rotation (Z&R) model is a simplification of the affine model, a special case where $a = d$ and $b = -c$.

In some papers, such as [12], and in JEM and VVC, Control Points (CP) are used. In this case, the affine parameters a, b, c, d are replaced with motion vectors mv_0, mv_1, mv_2, respectively defined by $\begin{bmatrix} mv_0^x & mv_0^y \end{bmatrix}^t$, $\begin{bmatrix} mv_1^x & mv_1^y \end{bmatrix}^t$, and $\begin{bmatrix} mv_2^x & mv_2^y \end{bmatrix}^t$. The former equations are replaced by the following equivalent form:

$$\begin{bmatrix} mv^x(x,y) \\ mv^y(x,y) \end{bmatrix} = \begin{bmatrix} mv_1^x - mv_0^x & -(mv_1^y - mv_0^y) \\ mv_1^y - mv_0^y & mv_1^x - mv_0^x \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \frac{1}{w} + \begin{bmatrix} mv_0^x \\ mv_0^y \end{bmatrix} \tag{2.4}$$

$$\begin{bmatrix} mv^x(x,y) \\ mv^y(x,y) \end{bmatrix} = \begin{bmatrix} mv_1^x - mv_0^x & -(mv_2^y - mv_0^y) \\ mv_1^y - mv_0^y & mv_2^x - mv_0^x \end{bmatrix} \begin{bmatrix} \frac{x}{w} \\ \frac{y}{h} \end{bmatrix} + \begin{bmatrix} mv_0^x \\ mv_0^y \end{bmatrix} \tag{2.5}$$

CPs can be seen as two (or three) different MVs that represent the motion at a given point. In the former equation, mv_0 is the MV for the top left of the block, mv_1 the MV for the top right, and mv_2 the MV for the bottom left. The two images in Figure 2.1 from [20] show how the CPs work.



(a) 4-parameter affine model with two CPs. (b) 6-parameter affine model with three CPs.

Figure 2.1: Overview of CP-based affine transforms

2.2.2 Transform Implementation

While all methods may use, in essence, the same formula to compute the displacement of each sample, as the coordinates for the reference samples are real-valued, interpolation is necessary. In [14, 16], the quarter-pel interpolation from HEVC followed by bilinear interpolation between the quarter-pel positions is used. This has the advantage of not requiring an additional interpolating filter that could be expensive to implement, as bilinear is the fastest interpolation outside of nearest neighbor. In [13, 15], more expensive interpolation filters are used to deliver a 1/16-pel precision. For JEM and VVC, all MVs, even in translation mode, use 1/16-pel precision. However, in contrast to [13, 15], where a MV for each sample is used, one MV is used for a block of 4×4 samples. In other words, in this case, AMP is only a way to specify many translation MVs for small blocks at once, with one additional feature: filtering is applied at the blocks boundaries, which avoids the blocking noise that would happen with independent MVs. While the interpolation filters from VVC provide a better precision, they are limited to 1/16 for each sample, while the bilinear filter used by the other methods can provide samples at arbitrary positions without rounding. However, in practice, further precision offers limited benefit.

2.2.3 Gradient-based MV estimation

While the various methods using gradient for ME differ, the base algorithm is the same. This subsection will start with introducing the common aspects of those methods, and the following will focus on the differences.

Each method starts with an initial MV. In the most common case, the translation

parameters have been estimated as they serve as a comparison point to decide if the mode should be used. Affine parameters can be predicted from neighboring blocks, and in case they are not available, they are set to zero. Additionally, when two models are available like in [20], the 6-parameter transform can reuse the result of the 4-parameter transform estimation. For methods using Control Points, this means all MVs are equal. See Subsection 2.2.5 for details on starting values for the gradient process. If multiple starting candidates are available, the best among the candidates will be selected for further iteration.

2.2.4 BM-based MV estimation

While the gradient-based methods have proven to be quite good at finding good affine MVs, they require a very different type of processing compared to the classical BM approaches. This means that for a hardware implementation, new silicon is required for these computations, and their cost could mean that many hardware encoders could decide to not use AMP entirely. In [16], the proposed BM algorithm is very similar to existing ones, which would greatly help making implementation easier. The proposed method can be seen as an extension of the square pattern search to more dimensions. Because the number of operations required at each step scales exponentially with the amount of parameters in the MV, some modifications are made to limit the complexity.

3-parameter models

As more parameters make estimation too costly, the authors decided to reduce the number of parameters from the most commonly used Z&R model. In many sequences, there is zooming or rotation but not both at the same time. So instead of allowing both the zooming and the rotation parameters to vary at the same time, the authors propose two 3-parameter models, one that represents zooming combined with translation, and the other rotation combined with translation. The two models can be represented using the following equations:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1+s & 0 \\ 0 & 1+s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{2.6}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & -r \\ r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{2.7}$$

In these equations, s represents the zooming factor and r the rotation factor. As the maximal rotation angle is small, the approximation in Equations 2.6 and 2.7 is good enough.

Expanding TZSearch to more parameters

As with gradient-based methods, the initial starting value is based on the result of the translation ME. Starting from the best translation-only MV, values for the affine parameter (either r or s) are tested. To limit the search space, only 32 possible values for the affine parameter are allowed. To avoid an exhaustive search, the first step searches every possible parameter using a step size $4.\Delta$, where Δ is the step between quantized values, with value 1/256 in the paper. After the best value is found, the two neighbors with a step size of $2.\Delta$ are tested. Then, the translation MV is refined by looking at all nine neighboring candidates with a quarter pel distance. Refinement of r or s with a step size of Δ follows, then the former two steps are repeated until either a maximum number of iterations is attained or no further improvement is achieved at a given step.

By separating the refinement into two steps, the exponential complexity is avoided. For each step, only 8 estimations for the translation and 2 for the affine parameter are required, while using a 3-dimensional diamond pattern would result in 26 ($3^3 - 1$) operations per step. However, as the number of affine predictions tested is still much higher than gradient-based approaches, pixel interpolation should be faster to compensate.

Pixel interpolation

To reduce the need for expensive pixel interpolation using the 1/16 pel precision computed for every estimation as in other methods [13, 15, 19–21], the image is interpolated entirely and stored in a buffer. Because storing an entire image interpolated with such precision would use too much memory, it is restricted to quarter pixel.

Positives and Drawbacks compared to gradient-based methods

Using a simpler model makes ME easier, but it also requires doing it twice. While it may be faster than a gradient-based approach done twice for selecting the best model, with a software implementation it takes a similar time or is slower compared to a four-parameter gradient-based estimation. While the lower accuracy in the prediction is good for lowering decoding complexity, it also limits the accuracy of the prediction, so it will not be able to achieve the same performance if processing power is sufficient for the higher complexity methods.

2.2.5 Motion Vector Prediction (MVP)

For good starting points in the iteration process and for efficient entropy coding, predicting the MVs is very important. For methods using translation and affine values, the same prediction is used for the translation MV as with normal prediction. The affine parameters can be predicted from a neighboring block that also uses affine prediction, if available. The major drawback from this approach is that if no neighbor uses affine prediction, no prediction can be done and the parameters are simply set to zero.

For the CP based methods, having neighbors that use affine prediction is not required. Instead, each MV can be predicted from the neighboring block closest to it. To predict mv_0, the block on the upper left is used, for mv_1 the top right and for mv_2 the bottom left. If one is not available, the missing one can be deduced from the other two, which will result in a 4-parameter transform. CPs also make scaling transforms using different reference picture easier, as they can use the same scaling as regular MVs.

Both [13] and [16] code the affine parameters using differential coding. However, this does not work well if the reference frames are not identical, limiting its use. In [21], the authors propose a way to implement motion scaling with the affine parameters by decomposing the 4-parameter transform into three different transforms that can each be scaled: a translation, a rotation and a scaling transform. For the 6-parameter affine transform, they decompose it using an additional transform representing shearing, and use 2 parameters for the scaling transform. However [16], who uses 3-parameter transforms that are easier to decompose and scale, does not implement this optimization in their method.

2.2.6 Entropy Coding

As these methods use additional parameters compared to the regular translation-based prediction, these parameters need to be coded, preferably efficiently so that the improvement in prediction accuracy is bigger than the increased cost in signaling the prediction.

In CP-based methods like [18, 20], the encoder codes Motion Vector Difference (MVD) like with regular motion compensation, except that instead of coding a single MV, it will code two or three. Like with normal prediction, an index is used to signal which MVP is used and then codes MVD defined as follows:

$$mvd_i = mv_i - mvp_i \tag{2.8}$$

where mvd_i is the coded MVD, mvp_i the predicted MVP and mv_i the MV for the current block, with i representing the two or three MVs.

In [20], further improvement was made on the coding as it appears that there is a high correlation between the mvd_i. They proposed to code alternative differences defined as:

$$\begin{cases} mvd'_1 = mvd_1 - mvd_0 \\ mvd'_2 = mvd_2 - mvd_0 \end{cases} \tag{2.9}$$

As mvd'_1 and mvd'_2 are usually smaller than mvd_1 and mvd_2, this allows for a gain in efficiency for very little additional processing.

For [13, 15, 16, 21], the encoder codes the translation parameters as usual and code the difference in affine parameters if a block using AMP is available. While [13, 15, 21] give little detail as to how they code the parameters and decided on the quantization, [16] shows results using different step size for parameters and investigated the parameters probability. The probability distribution they obtained showed that truncated unary code was the most efficient coding. They also use additional CABAC symbols for the sign of the affine motion parameters when not coding in differential mode, as they tend to often be similar throughout the picture.

In Table2.1, a summary of the differences between the various methods described so far is presented to highlight the differences.

Table 2.1: Overview of related Affine Motion Prediction methods

Method	Model	Parameter representation	Parameter coding	Iteration init	Iteration process
JEM [18]	4	2 MVs	MVD	MVP + MV_t	Gradient
VVC [20]	4+6	2/3 MVs	MVD′	MVP + MV_t + MV_{parent}	Gradient
Heithausen [15]	4/6	T:MV + Affine	T:MVD A:diff	MV_t	Gradient
Tsutake [16]	4* (3+3)	T:MV + Affine	T:MVD A:diff	MV_t	BM

2.3 Fast Mode and Block Partitioning Estimation

One of the major tasks of an encoder is to find the optimal way to partition blocks and chose the mode (intra, inter, merge, skip) to use for each block. The sure way to find it is to perform an exhaustive search. However, it requires an incredible amount of time, especially for newer encoding standards as the number of options increases. Motion Estimation is far from the only time consuming operation in the encoding process. Performing Rate Distortion Optimization (RDO) over all possible modes to find out which is the most efficient takes a very long time. It has to predict the values, transform then, quantize the residual and entropy code everything, leading to the cost function

$$J_{mode} = SSE + \lambda_{mode} B_{mode} \tag{2.10}$$

where SSE is the squared error from quantizing the residual, B_{mode} the number of bits to code the current block, and λ_{mode} a Lagrange multiplier to optimize for a certain trade-off between quality and bitrate.

To limit the time requirements, some faster estimators were designed. For example, when trying to find the best transform, comparing MVs based on the prediction error is much faster than performing full RDO. The first steps will use SAD and the finer steps SATD. After getting the best transforms for each mode, RDO is desired to pick the best accurately. To limit the number of evaluations required, it is possible to estimate with SATD the top X candidates, and only perform RDO on them. The higher the number

of candidates, the more likely the encoder will pick the right transform, but the more time it will take to do that.

There has been a lot of research in choosing modes optimally. A common way to limit complexity is avoiding splitting blocks as it requires less ME evaluations. Some simple heuristics, as demonstrated in [10, 22, 23], show improvements around a reduction in half of the encoding time for a minor in encoding efficiency. To evaluate encoding efficiency objectively, almost every paper uses Bjøntegaard-Delta Rate (BD-R) as defined in [24]. Reference [10] offers 56.75% for a 1.3% increase in bitrate,while reference [22] shows 40% for 0.05dB, and reference [23] demonstrates 45% for 0.1dB loss.

More recent work often uses machine learning, with Bayesian Decision Rule [25], support vector machine (SVM) [26–29], Adaptive Decision Trees [30], and also Neural Networks [31].

In [25], the encoder is able to update its weights online and adapt with the current sequence being encoded. It reduces encoding time by 54% for the Random Access mode for a 0.71% increase in bitrate. In SVM-based methods, the block-splitting decision is seen as a binary decision problem. Reference [26] achieves 44.7% complexity reduction on average with only 1.35% BD-R increase. In [27], the classifiers are trained for each block size, and some online learning is also proposed. It offers a 51.5% reduction for just below 2% coding efficiency loss. Reference [28] expands SVM decision-making to Prediction Unit (PU) partitioning. It is able to offer even encoding speed, with a 65.6% decrease for a cost of 3.7% BD-R increase.

Reference [29] is very thorough in analyzing the various features and which are relevant for accurate mode decision. They have found features that are good predictors that had not been used so far. The training focused on RDO instead of classification accuracy. This means that misclassifications matter much more when they incur a large efficiency cost. They also devised a threshold to adjust the bias of the encoder towards trying more splits or skipping more often. This allows getting different speed/efficiency tradeoffs points that one can select according to their needs. With a cautious bias, they achieve a 40.4% time save for 0.18% loss in efficiency or 48% time save for 0.48% BD-R increase.

Reference [31] shows a hardware encoder that uses a convolution neural network to make mode decisions. One advantage of neural networks is they are extremely parallel, which allows for a fast processing without requiring a high frequency which would incur a large power cost. The proposed design saves 61% of encoding time for an increase

in BD-R of 2.67%. For a 1080p sequence, the proposed encoder is able to encode 55 frames/s with a power consumption of 16.2mW.

Reference [30] uses decisions trees trained with a few videos to determine the optimal thresholds for each decision. They use only two features, the SATD and the entropy of the current block. If the SATD is small, it means the current block already offers a good prediction. Conversely, if it is large, smaller different transform could be more accurate. They show a high level of correlation between SATD and actual splits in the reference encoder.

A different way to solve the splitting problem could be to compute object boundaries and segment blocks around. In [32], the authors propose segmenting blocks arbitrarily with inexpensive signaling as it is based on samples values in a reference block. The resulting two blocks use two MVs, one also being used for computing the boundary between the two blocks.

2.4 Filtering of Coding Artifacts

With standard translation-based prediction, and even in the proposed AMP implementation in JEM and VVC, there are many discontinuities in the MV that cause discontinuities in the reconstructed pictures. As they are high-frequency components, they are the most affected by quantization, and can be most visible at lower bitrates that uses more aggressive quantization.

2.4.1 In-loop filters of AVC and HEVC

AVC and HEVC, include a deblocking filter [33, 34] that aims to reduce the artifacts created by block boundaries. HEVC also introduced a new Sample Adaptive Offset (SAO) filter [35]. The deblocking filter shows a large improvement in removing block artifacts, with however the downside of sometimes filtering too much and removing details, as there is no advanced heuristic to know when there is a real edge or an artifact, other than some threshold that depends on the quantization level. The SAO filter is very useful at reducing ringing artifacts, which are a consequence of the quantization of the high frequency parameters of the residual.

These filters are in-loop filters, mandated by the standard and their output is used for

the reference picture. This means that not only they improve the current picture, they can also improve the other pictures by having a better picture to predict from.

2.4.2 Overlapped Block Motion Compensation (OMBC)

Because the affine prediction in JEM is actually a lie and is just a way to describe 4×4 MVs more efficiently, blocking artifacts will appear at all these boundaries. To avoid this issue, Overlapped Block Motion Compensation (OMBC) can be used.

OMBC is not a new concept, as there were papers proposing it (and coining the term) as far back as 1992 [36]. Because it can compensate for the irregularities that arise from discontinuity in the prediction, it was adopted in JEM and proved it could give large gains in [5]. However, it also requires more time to decode, which is a significant problem.

While the implementation is different than the deblocking filter and the SAO filter, its is quite similar in principle. The biggest difference is that it is applied during the prediction phase, instead of after everything else like the two other filters. Its main objective is to smooth out the transition between the predicted blocks.

2.4.3 Postfiltering

Adding a filter in a coding standard is no easy task, and it can only provide a benefit when people adopt it years later. The only way to increase video quality right now is to do processing on an already encoded video.

Postprocessing video is not new and is in fact part of any practical video player, as it will have to handle resizing of the video to match a given size. Outside the very uncommon case of no chroma subsampling, the chroma components will have to be scaled to the size of the luma component.

In the recent years, there have been numerous proposals of filters that perform post processing to improve video quality, such as [37–39]. They all use convolutional neural networks (CNNs), which have become popular in recent years because of the great results they achieve in computer vision competitions.

Reference [37] uses a very deep network of 20 layers to achieve an overall reduction in bitrate of 1.6% with the BD-R metric. It adapts for different quality settings by training

for various Quantization Parameter (QP) values. This could require more processing to transmit the QP information to the filter. It works only on the luma component. Paper is unclear but it looks like the same sequences are used for both training and testing, which could boost the results.

Reference [38] work is based on super resolution CNN (SRCNN), but replaces the low resolution images with the degraded images from the encoding process. Same as [37], they train for several QP values. They show results for All Intra, Random Access and Low Delay P, with up to 4.8% BD-R improvement for the All Intra mode. They manage to filter images faster than [37], but it is still quite slow as 1.2 seconds for one frame of size 832×480. It also uses the same training and testing sequences.

Reference [39] shows a 4.6% reduction in bitrate over many sequences. It proposes a filter than also works on the chroma components, however it is even slower than the others, processing a 176×144 picture in almost a second. They present a GPU implementation that is twice as fast, but that is still very slow, as for a FullHD (1080p) picture, time (assuming linear scaling), would go over a minute per frame.

In previous work [40, 41], we have studied more efficient ways to provide similar improvements to videos, but while keeping real time processing in mind. While we achieved only 3.4% improvement in the best case, it was possible to filter several frames per second on a laptop.

2.5 Optical flow

Estimating the movement between two images has been a subject of research for a long time, as it has numerous applications. For video coding, it is necessary for finding motion vectors, and is often done through computationally expensive methods that check the error for each possible motion vector, with more recent methods improving the search algorithms to keep the encoding time reasonable. In those cases, only the cost for the whole block is considered, so the movement estimation is often not accurate at a more granular level.

However, in many applications, the movement for each pixel is desired. This is typically referred to as optical flow. A recent application that also shows potential for video coding is frame interpolation, where by computing the movement for each pixel between the two frames, it is possible to estimate the missing frame with remarkable accuracy,

which was demonstrated in [42]. It is even possible to use the MVs that were computed during encoding in the construction of the optical flow, increasing the speed of computation greatly [43]. While the computed interpolated frame could be used in encoding with a new kind of prediction, [43] is still too slow, especially as it puts a large burden not only on the encoder but the decoder as well. While methods that increase encoder complexity will easily be included in the standard if they offer a good improvement in BD-R, decoder complexity increases the complexity of decoder hardware that has to support it, while hardware encoders can choose to not use specific modes if they want. For example, a fast encoder can skip RDO and modes that are used rarely, decreasing efficiency but increasing speed a lot.

One of the most famous and popular methods for estimating optical flow is the Lucas-Kanade method [44]. It has been used a lot and gives satisfying results for simple movements. It is quite fast, which is one of the reasons for its popularity. On the other hand, there are some methods that achieve much higher accuracy, like the state-of-the-art EPICFlow [42]. However, they rely on finding common points in the two images, and this takes over 15 seconds in a 1024×436 pixels image, even with a fast CPU. This is too slow for practical encoding applications, which aim to achieve real time processing or at least process several frames per second.

Ce Liu's method [45] offers results that are better than Lucas-Kanade, but is not too slow to be impractical. It is based on [46] and [47]. The increased speed is achieved by replacing the costly solving of large linear systems by a conjugation gradient method. This means using several filtering and weighting steps, which are easy to parallelize and process with vectorized instructions. On a similar CPU as used in the EPICFlow test, processing takes less than a second without multi-threading. The number of steps and the weighting can be adjusted. Reducing the number of steps decrease accuracy but improves the processing speed greatly.

3 Reducing the Defects in Video through Postprocessing

This work expands on my master thesis [41], which performed filtering on blocks based on block boundaries. This presents a different filtering approach that targets all blocks.

3.1 Change of the input

In the previous method only blocks around a block boundary were filtered, which reduced its effectiveness with Random Access mode, as it usually uses larger blocks compared to All Intra mode. Moreover, this made learning difficult because fewer blocks were available for training. It was also running the filter in two directions, requiring transposition of the block data to use it with the same filter.

3.1.1 Filtering the whole image

In HEVC, blocks are organized in a quad-tree structure, the largest blocks being 64×64 samples large, and the smallest 4×4, but only some blocks are allowed to be that small. In practice, 8×8 are more common.

Because of this, we decided to center the filtered blocks on the 8×8 boundary. In other words, the center of the blocks can be defined by $\begin{bmatrix} N \times 8 & M \times 8 \end{bmatrix}^t$, where both N and M are strictly positive integers.

Instead of running the filter on only some blocks, the filter runs on every possible block in the image, allowing for more improvement in the Random Access mode.

The previous filter was only focused on the luma component, but with this architecture it is easier to process chroma as well, so it was included. The input blocks are the same for both luma and chroma samples, with filtering parameters trained separately.

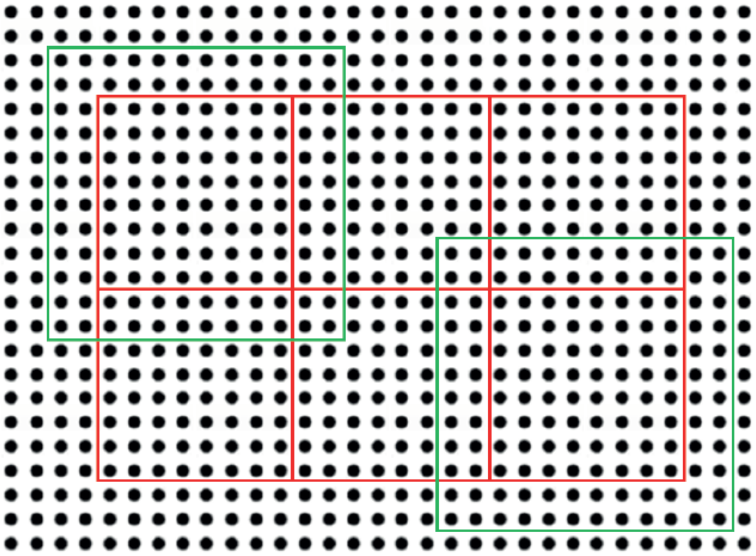


Figure 3.1: Representation of filtered blocks (in red) and input blocks (in green). Only two input blocks are represented for clarity. One overlaps every filtered block

3.1.2 Size of input blocks

When filtering errors out of data, having more context is usually better. However, this also incurs a significant cost. We decided on filtering 8×8 blocks, with a larger 12×12 input for context. The outputs do not overlap, which is critical to allow fast processing. In contrast to the filters in AVC and HEVC, there is no required order when processing the blocks, which helps parallelism greatly. Figure 3.1 shows how the blocks are organized to avoid overlapping.

3.2 Architecture of the proposed method

The architecture is inspired from the one used in my master thesis, with a few key changes that allow better improvement.

3.2.1 Bidirectional filter

In the previous work, the filter was designed to work on only one edge. To this end, it has a 12×4 convolution in the direction of the boundary. As with the present blocks

boundaries can be present on either (or none) of the possible directions, a symmetric structure is preferable.

The first convolution was thus changed to 7×7. The following fully connected layer was adjusted in size to match the different input.

3.2.2 Feature layers

The previous solution used 32 features layers, which already slowed down the processing greatly when processing only a subset of the total amount of possible blocks. To keep processing time reasonable, we decided to reduce the amount of feature layers to 8. This would make near real time processing possible on a standard computer using CPU.

The proposed method as it is described here was published in the journal IIEEJ Transactions on Image Electronics and Visual Computing in the June 2009 issue [48].

3.3 Implementation and measured speed

We measured the speed of the proposed filter with our initial optimization using Matlab and the library MatConvNet [49]. It performed slightly better than the previous method, with 2.5 fps per thread on the test computer.

Using Intel's MKL library [50], we achieved better performance through more optimized routines. With a better implementation, we achieved over 5 fps per thread, with 32 fps when using the whole CPU.

This is not yet completely real time as the processor needs to decode the image as well, but it shows that with either more modern CPUs with more cores or a GPU implementation, real time is quite realistic.

The fully connected layer is very easy to compute efficiently, but we did not find an optimized 7×7 convolution, which is not able to perform as many operations per cycle as the fully connected layer. It could be possible to change the size of the convolution to get more speed, but this was not investigated in the present study.

4 Estimation of Affine Motion Parameters

The key to fast estimation of affine motion parameters lies in two aspects. First, finding good initial estimations that are very close the the optimal MVs, and second an iteration method that converges quickly towards the optimal value without being too slow. The best way to save time is to avoid computing motion parameters when they are not needed, which will be the focus of Section 4.1. The following, Section 4.2 will focus on using optical flow to find good initial estimations. Then, Section 4.3 will describe the iteration algorithm that converges towards the desired MV. The last, Section 4.4, will detail the implementation in HEVC and some compromises made for preserving speed. To allow easier comparison with existing methods on solely parameter estimation over variances in the implementation of the coding and representation of transforms, the computed transform use the two 3-parameter models proposed by [16].

4.1 Optical flow based segmentation and transform selection

4.1.1 Choice of the optical flow method

In this thesis, the optical flow method from Ce Liu [45] was considered because the computation cost varies solely on the size of the input image and the parameters for the number of iterations. This makes it easier to tune for a compromise between time and accuracy.

It also offers very nice properties for hardware implementation, as all the operations are highly parallel in nature, which makes them very easy to implement in hardware.

Computing optical flow requires two images. When using it in an encoding context, the first picture is already known by the encoder/decoder, and further referred to as reference picture, while the second is the picture we are currently encoding or decoding, further referred to as target picture. An example using colors for visualization of the

(a) Reference picture

(b) Target picture

(c) Visualization of the optical flow between 4.1a and 4.1b. Color intensity shows absolute value of the motion vector and tint the direction

Figure 4.1: Overview of the optical flow process

optical flow can be seen on Figure 4.1.

4.1.2 Ignoring some blocks

As computing affine transforms require more processing power than normal ones, a good way to limit the additional processing is to estimate when it will not be able to get good results and avoid the costly affine motion estimation in the first place. To this end, a very simple heuristic, fast to compute is used: the variance of the optical flow.

$$\bar{x} = \sum_{i=0}^{N} \sum_{j=0}^{N} \frac{flow_x(i,j)}{N^2} \tag{4.1}$$

$$\sigma_x^2 = \sum_{i=0}^{N} \sum_{j=0}^{N} \frac{(flow_x(i,j) - \bar{x})^2}{N^2} \tag{4.2}$$

$$\bar{y} = \sum_{i=0}^{N} \sum_{j=0}^{N} \frac{flow_y(i,j)}{N^2} \tag{4.3}$$

$$\sigma_y^2 = \sum_{i=0}^{N} \sum_{j=0}^{N} \frac{(flow_y(i,j) - \bar{y})^2}{N^2} \tag{4.4}$$

$$\sigma_{xy} = \sqrt{\sigma_x^2 + \sigma_y^2} \tag{4.5}$$

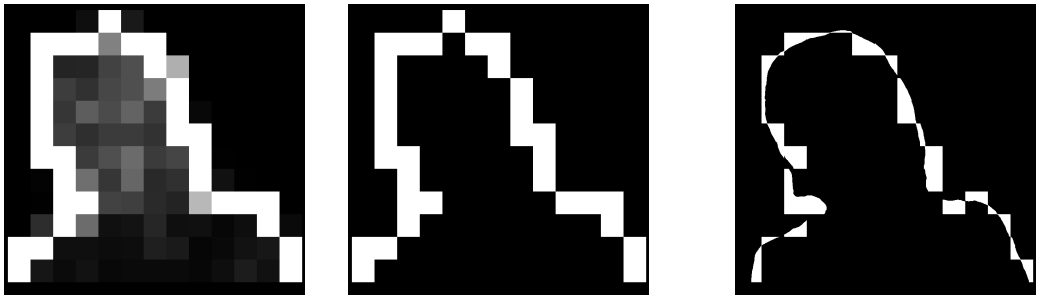
When the variance is below a certain threshold, the optical flow is mostly identical for the whole block, which means translation will work perfectly. Conversely, when it is too high, it usually correlates with object boundaries. Splitting the block would be advised. Lower variance can happen even in case of large discontinuities, but that is most likely to happen if only a few pixels should belong to a different block, but the overhead of coding these pixels separately would not be worth the entropy saved by the reduction in the residual. Between those two extremes, affine motion estimation is likely to offer gains. The estimation of affine motion parameters will be detailed in Section 4.2

In this thesis, the thresholds for σ_{xy} are 0.1 and 4. The mode decision can be summarized with the following:

$$\begin{cases} \text{translation if} & \sigma_{xy} < 0.01 \\ \text{affine if} & 0.01 < \sigma_{xy} < 4 \\ \text{split if} & \sigma_{xy} > 4 \end{cases} \tag{4.6}$$

For the translation case, it is possible to reuse $\bar{x}$ and $\bar{y}$ as initial values and perform refinement with existing methods. Square search was used because it is fast and delivers good results.

When variance is very high, splitting the block becomes necessary, but most encoders require a very expensive search for all possible splits, which require a lot of time. If the splitting threshold is met, segmentation is performed using K-Means. The flows for the x



(a) Variance σ_{xy}, scaled for a maximum level of 4 (b) Blocks with variance over the threshold (c) Result of k-means segmentation on each block from 4.2b

Figure 4.2: Overview of block segmentation algorithm

and y displacements for the given region are combined in an array of (x, y) pairs, then fed into the algorithm, with a number of classes of 2. This results in a partition of the block, with each class belonging to a different object with a different motion. An example of such partition can be seen in Figure 4.2, and a comparison of the prediction results in Figure 4.3. It shows that with the proposed split around the face, the discontinuities in motion that created sharp edges in the left picture are less present in the second one. As both cases still use only translation, the prediction is not perfect even with the correct block segmentation.

4.2 Optical flow based transform estimation

4.2.1 Transform computation

Computing a transform requires three images: a reference picture, the target picture and a picture representing the optical flow. However, instead of using the reference picture after reconstruction by the encoder, the original picture is used. This offers two advantages: first, this allows optical flow to be computed before the picture is encoded, and second, the Motion Estimation is more accurate and follows the real movement better, especially when the quantization parameter is large and the reconstructed image is of lower quality. As the optical flow estimation of movement tracks the actual movement of pixels, it can be thought that the ground truth is identical for both before and after encoding pictures.

After obtaining an approximate displacement for each pixel in the picture, the estimation is performed for each block. The estimation is based on resolving the following

(a) Target picture predicted with a translation for each block

(b) Target picture predicted with two translations for split blocks

Figure 4.3: Comparison of prediction results when splitting or not blocks

linear equation for the 4-parameter (Z&R) model transform:

$$\begin{bmatrix} x'_0 \\ y'_0 \\ x'_1 \\ y'_1 \end{bmatrix} = \begin{bmatrix} a & b & 0 & 0 \\ -b & a & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -b & a \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \\ v_x \\ v_y \end{bmatrix} \tag{4.7}$$

where $\begin{bmatrix} x_0 & y_0 \end{bmatrix}^t$ and $\begin{bmatrix} x_1 & y_1 \end{bmatrix}^t$ represent the coordinates of two points in the target picture and $\begin{bmatrix} x'_0 & y'_0 \end{bmatrix}^t$ and $\begin{bmatrix} x'_1 & y'_1 \end{bmatrix}^t$ the corresponding two points in the reference picture.

For computing the parameters a and b, at least two points are required. Using $x = x_1 - x_0$ as the difference in the input points and $x' = x'_1 - x'_0$ as the difference between the output points, we can estimate a and b with the following equations:

$$\begin{aligned} a = 1 + s &= \frac{xx' + yy'}{x^2 + y^2} \\ b = -r &= \frac{x'y - xy'}{x^2 + y^2} \end{aligned} \tag{4.8}$$

After we get the values of a and b, finding $\begin{bmatrix} v_x & v_y \end{bmatrix}^t$ is easy, by using them in the first two lines of Equation 4.7:

$$\begin{aligned} v_x &= x'_0 - ax_0 - by_0 \\ v_y &= y'_0 + bx_0 - ay_0 \end{aligned} \tag{4.9}$$

To get good results, the points should be far enough apart, so points around the edge of the current block are used. If the points are too close together, cancellation is likely to occur, as the subpixel motion estimation through optical flow is imprecise. To remove the risk of bad estimations from outliers, the values of a and b are estimated for multiple couples of points, and the median value is retained. When the block is on the edges of the image and contains pixels outside the source picture, which happens when the input size is not a multiple of the block size, points that are within the input frame are used.

4.2.2 Transform Selection

When using a 3-parameter transform like in [16], it is necessary to evaluate both models to pick the best. This doubles the required processing and makes finding parameters quite slow. In the proposed method, one transform is selected according to the computed parameters.

To see which 3-parameter model would fit best, the absolute values of s and r are compared, and the model corresponding with the highest value is selected. In case neither is bigger than a small threshold, affine motion estimation is skipped for the current block. In those cases, affine transforms would not give enough additional precision to outweigh the cost of coding them. By skipping those evaluations, time can be saved in the encoding process.

4.3 Transform Iteration

While the parameters obtained from the initial estimation can be good and often superior to the translation parameters, they are rarely optimal, so further refinement is desired.

4.3.1 Initial Iteration

While [16] goes over all possible values for the motion parameter with a step size of 4.Δ,where Δ is the quantized step size, only neighbors of the initial predicted value are investigated. The refinement uses two steps, first of 2.Δ, then of Δ. At each step, SAD is computed for the three values and the best is used for the next. If at the end of the two steps the best parameter is zero, further refinement is aborted and the translation model is used.

4.3.2 Further Refinement

After the initial iteration is completed, the encoder evaluates neighboring translation positions at a quarter-pel distance. The best in the nine MVs is picked as the final transform.

4.3.3 Bad Estimation Detection

Sometimes optical flow gives bad results, especially when the motion was scaled too much. In the event that the predicted value shows a SAD too large compared with the translation estimate, parameters are estimated with the full search as described in [16]. The estimation is performed only for the transform determined by the optical flow, as it showed great accuracy at finding the best transform model. We propose to perform the full search if the initial estimation SAD is larger than the translation SAD.

4.4 Implementation in HEVC

Implementing a new transform in HEVC is a complex process, and reimplementing related methods for comparison purposes adds further additional work. Tsutake et al. [16] agreed to share their code to help with the implementation. They also shared their implementation of [15] and a 3-parameter variant of it. In the proposed method, the entropy coding and transform computations are left identical, with the major changes being the computation of optical flow and a different algorithm for the motion parameters estimation. We published the results of the HEVC implementation in the journal Applied Sciences in January 2020 [51].

4.4.1 Distance Scaling

To avoid computing optical flow on every picture in the reference picture list (RPL), optical flow is computed for only the first frame in the RPL, and distance scaling is used for other pictures.

To predict values for other frames in the picture reference list, the displacement is scaled proportionally to the number of frames. This approximation is typically accurate enough when the movement stays similar. For example, if the first reference picture is at a distance of 1 and the second at a distance of 2, the displacement values are doubled. This works similarly to the distance scaling of HEVC or the one used in JEM with the multiple MVs, except that it runs on the estimated MVs from the optical flow.

For the translation parameters, experiments showed that using the result from the translation ME yielded better results than the estimation from optical flow, especially when motion scaling is required.

4.4.2 Parallel Processing

The Optical Flow method requires no encoding information and can be performed while other frames are being encoded. In a typical situation, while the first frame, which has to be Intra, is being encoded, there is enough time for the optical flow computation for the second frame, so if enough CPU cores are available, it can be computed before the need for it arises. If a single frame delay is acceptable, this method will allow saving a significant amount of time in the main encoding loop that iterates over all blocks, as it must run in order. Even in the case where this one frame delay is unacceptable, the optical flow method used can be parallelized very well, and as it performs only basic mathematical operations with no branching, can easily run on a GPU or dedicated hardware.

4.4.3 Using gradient for iteration

As the gradient method showed better results than Tsutake's, we decided to try to use it for refinement. We used the 3-parameter gradient code they provided and changed the transform model decision. Instead of computing both transforms and measuring which one gives the best results after finding the optimal values for each, we decide which

transform model to use based on the optical flow estimation.

4.5 Improvement of Entropy Coding

The code we obtained from Mr. Tsutake did not include the differential coding for coding the affine parameters. As it should offer some efficiency gains without making coding longer, I have worked on implementing it on top of the proposed method. The entropy coding as described in [16] can also probably be improved further, using some techniques that are used in VVC as described in [20]. First, we implement the differential coding as proposed by [16], then we propose a Affine Motion Vector Prediction (AMVP) framework for AMP in HEVC.

4.5.1 Differential coding using neighboring affine predicted blocks

As modes tend to be the same in neighboring blocks, we change the signaling to signal more efficiently the transform mode. If at least one of the neighboring blocks use affine prediction, we check if one uses the same transform model as the current block. In this case, the first neighbor, using the MVP candidate list order, is used as reference. Coding uses a sign bit followed by truncated unary coding, as described in [16].

When no neighbor or neighbors use only the other transform type, coding falls back to the original method coding the parameter from scratch.

4.5.2 Virtual Affine Motion Vector Prediction Candidate

When no neighboring blocks use affine prediction, we would still want to predict affine parameters as [12, 20] do by using the differences in neighboring MVs. We compute the affine parameters a and b with Equation 4.8, setting mv_0 as the top left point of the block and mv_1 as the top right point. If either block uses affine prediction, the MV at the edge of the block is estimated by applying the transform. We also use motion scaling for the MVs if the neighboring block uses a different reference picture than the current block. As with the transform mode selection estimation, the biggest between r and s is evaluated and the corresponding transform with the associated parameter is added to the candidate parameter list.

It is possible to simplify Equation 4.8 as follows

$$\begin{aligned} a = 1 + s &= \frac{mv_1^x - mv_0^x}{w} \\ b = -r &= \frac{mv_1^y - mv_0^y}{w} \end{aligned} \tag{4.10}$$

using $x = w$, $y = 0$, $x' = mv_1^x - mv_0^x$ and $y' = mv_1^y - mv_0^y$. w is the width of the block, which is also the distance between the two MVs.

In case neither r or s are different from zero, the virtual candidate is unavailable.

4.5.3 Proposed AMVP Framework

To avoid having to code the affine model to use and the parameters from scratch, the best solution is to predict them accurately. The previous solution using the difference with the neighbor still required to code the affine model. We create a AMVP candidate list that will require coding to only use one symbol for the index in the list and another for the difference, which should be smaller than with the previous methods. We populate the AMVP candidate list with neighboring affine predicted blocks first, similar to [20], then add the virtual AMVP candidate. If the list is not full, we add zero-valued transform of the other type if only a single transform type was present. Like with normal candidates, if one candidate would be equal to another it is skipped and not added to the list.

If no candidates outside the zero-valued candidates are found, coding is performed with the two flags for the affine model and the sign of the parameter as described in [16]. This happens for the first block in the picture, as there are no already coded neighbors to use. Virtual candidates are also unavailable for the top row as only one neighbor is available.

5 Results

5.1 Source Videos

In the following experiments, the videos were sourced from two datasets: the ITE/ARIB Hi-Vision Test Sequence 2nd Edition [52] and the Derf Test Media Collection [53]. The list of sequences and the motions they present is listed in Table 5.1. Sequences from [52] have their sequence number appended to their name for easier referencing.

Table 5.1: Video Sequences

Sequence Name	Motion	Frame Number
Ginko trees (s201)	Zoom	100–159
Truck train (s202)	Zoom	100–159
Cosmos flowers (s203)	Camera tilting	100–159
Fountain dolly (s209)	Dolly + Panning	100–159
Basketball (s214)	Panning	100–159
Twilight scene (s215)	Zoom	100–159
Horse racing (s218)	Fast panning	100–159
Rotating disk (s251)	Rotation	100–159
Colorful world (s259)	Various	100–159
Fountain (s265)	Rotation	100–159
Station	Zoom	1–313
Stockholm	Panning	4–604
Blue Sky	Rotation	1–217
FungusZoom	Zoom	1–240
Tractor	Rotation + Zoom	1–690

5.2 Post filtering of HEVC videos

In this experiment, we will evaluate how much the proposed method can increase PSNR of videos encoded using HEVC. We also compute the BD-R [24] of the proposed method,

symbolizing how much bitrate could be saved by the use of the filter. We did so for every component, and also calculated an overall gain regrouping luma and chroma that is used in a famous paper for comparing encoding efficiency [3]. The formula used to compute the aggregate PSNR $\mathrm{PSNR}_{\mathrm{YUV}}$ is:

$$\mathrm{PSNR}_{\mathrm{YUV}} = (6 \times \mathrm{PSNR}_{\mathrm{Y}} + \mathrm{PSNR}_{\mathrm{U}} + \mathrm{PSNR}_{\mathrm{V}})/8 \tag{5.1}$$

When calculating the BD-R, we use the aggregate PSNR for the last value using the aggregate. We used all sequences from [52], split in four groups for cross-validation. For encoding, we used the open source and freely available x265 encoder [54], which is used a lot in the industry, for example at Netflix where they have studied its performance for their content in depth [55]. We used two encoding modes, inspired by All Intra and Random Access standard modes for the reference HEVC encoder HM. The QP values used mirror the standard test conditions with $22, 27, 32, 37$ used.

5.2.1 All intra mode

The results we obtained when encoding the sequences with the All Intra configuration are presented in Tables 5.2 and 5.3. The former shows the BD-R results and the second the PSNR values obtained from the filter for 2 QP values, 22 and 32. Figure 5.1 shows the Rate-Distortion curve of the proposed filter compared to HM.

With the exception of on sequence (Twilight scene), each sequence was improved, especially the luma component, with over 4% reduction in bitrate for the sequence rotating disk. The chroma component shows less improvement on average, but outside of one outlier, it showed an improvement of at least half a percent, with 5.7% on the Basketball scene. The median shows a much better typical result than the average, showing how much the outlier skews the results. As this sequence is very dark while others are typically quite bright, it is likely the training on the other sequences was inadequate for this one. A larger dataset including similar sequences would be likely to improve the results.

As it could be expected, sequences encoded with a low bitrate, having a lower PSNR, are more improved by the filter.

Figure 5.2 shows the improvement when using the filter for the Basketball and Rotating disk sequences. The visible encoding artifacts from the low-bitrate encoding are less

Table 5.2: Filter results for All Intra mode

Sequence	Y	U	V	YUV
s201 (Ginko trees)	-2.6%	-2.7%	-0.8%	-2.4%
s202 (Truck train)	-1.6%	-2.7%	-2.8%	-1.7%
s203 (Cosmos flowers)	-2.9%	-1.4%	-0.5%	-2.6%
s209 (Fountain)	-1.4%	-1.7%	-0.7%	-1.4%
s214 (Basketball)	-1.9%	-5.7%	-3.4%	-2.5%
s215 (Twilight scene)	3.0%	16.0%	4.2%	4.2%
s218 (Horse racing)	-1.1%	-1.2%	-1.5%	-1.1%
s251 (Rotating disk)	-4.4%	-1.8%	-1.0%	-3.9%
s259 (Colorful World)	-3.5%	-2.9%	-1.2%	-3.2%
s265 (Fountain)	-0.9%	-3.2%	-2.9%	-1.0%
Median	-1.7%	-2.2%	-1.1%	-2.1%
Average	-1.7%	-0.7%	-1.1%	-1.6%

visible and more smoothed out.

5.2.2 Random access mode

The results we obtained when encoding the sequences with the All Intra configuration are presented in Tables 5.4 and 5.5. The former shows the BD-R results and the second the PSNR values obtained from the filter for 2 QP values, 22 and 32. Outside of one sequence (Twilight scene), which suffers from the same issues as with the All Intra case, all other sequences were improved with on overall -1.4% PSNR_{YUV}. BD-R reduction. In most cases, the luma component is improved the most with an average of -1.9% BD-R improvement and up to -4.9%.

Compared to All Intra, there is less improvement on the chroma component. It could be related to the lack of training, as chroma has less blocks to work with in a sequence because of chroma subsampling, and in Random Access the various types of frames and the different levels of degradation make learning harder.

It could be more efficient to train for each frame type, but this would require more sequences to have enough data for sufficient training.

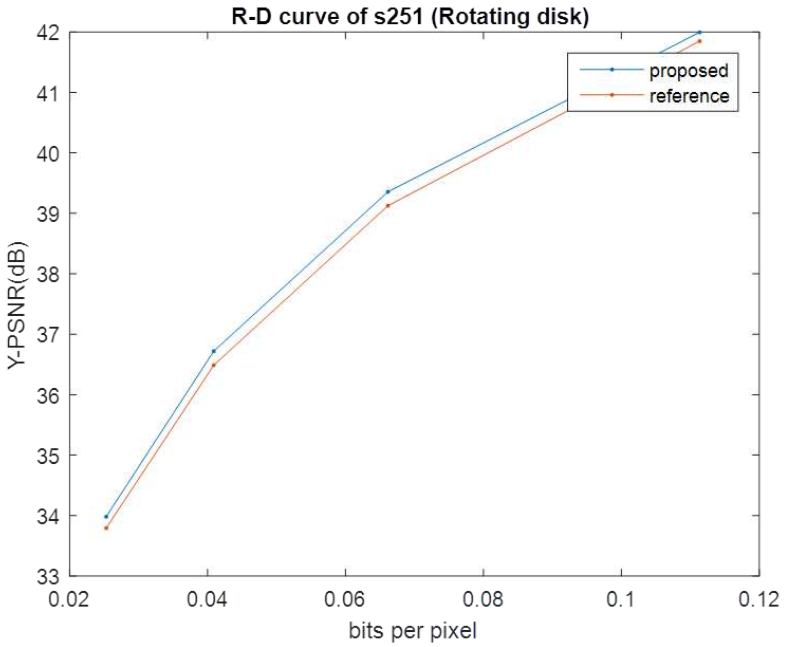


Figure 5.1: Rate-Distortion Curve of the Rotating Disk sequence with the proposed filter in All Intra mode

(a) Picture without filtering

(b) Picture with the proposed filter

Figure 5.2: Comparison of images from Basketball and Rotating Disk sequence with and without applying the proposed filter

Table 5.3: PSNR values for two different QP values on each sequence for the proposed filter and reference HEVC (All Intra)

	QP22						QP32					
	Reference			Proposed Method			Reference			Proposed Method		
	Y	U	V	Y	U	V	Y	U	V	Y	U	V
Ginko trees	40.69	40.92	40.28	**40.84**	**40.97**	**40.30**	31.74	35.32	33.96	**31.97**	**35.46**	**34.00**
Truck train	41.23	41.55	42.63	**41.37**	**41.56**	**42.65**	33.09	38.72	39.43	**33.16**	**38.77**	**39.49**
Cosmos flowers	40.70	40.27	**40.70**	**40.93**	**40.28**	40.69	32.65	35.37	36.01	**32.87**	**35.44**	**36.04**
Fountain	41.25	44.61	43.87	**41.33**	**44.61**	**43.87**	35.99	43.50	40.80	**36.03**	**43.52**	**40.81**
Basketball	42.17	44.87	44.17	**42.22**	**44.99**	**44.21**	37.43	39.83	38.86	**37.54**	**40.16**	**39.06**
Twilight scene	**41.31**	**41.85**	**41.55**	41.24	41.76	41.52	**37.02**	**39.11**	**37.37**	36.91	38.63	37.18
Horse racing	42.25	43.84	46.19	**42.27**	**43.85**	**46.20**	37.69	39.65	41.64	**37.75**	**39.71**	**41.71**
Rotating disk	41.49	42.50	43.32	**41.68**	**42.52**	**43.34**	35.62	39.59	38.57	**35.91**	**39.65**	**38.62**
Colorful World	40.84	40.96	41.14	**41.04**	**41.01**	**41.17**	33.92	35.51	34.58	**34.14**	**35.68**	**34.66**
Fountain	41.42	43.43	44.72	**41.49**	**43.44**	**44.73**	36.18	42.64	42.37	**36.19**	**42.67**	**42.44**
Average	41.33	42.48	42.86	**41.44**	**42.50**	**42.87**	35.13	38.93	38.36	**35.25**	**38.97**	**38.40**

Table 5.4: Filter results for Random Access mode

	Y	U	V	YUV
s201 (Ginko trees)	-2.1%	-2.0%	-1.4%	-2.0%
s202 (Truck train)	-0.8%	-0.3%	-1.1%	-0.8%
s203 (Cosmos flowers)	-4.7%	-0.9%	-0.3%	-4.1%
s209 (Fountain)	-1.5%	-0.8%	-0.3%	-1.4%
s214 (Basketball)	-1.8%	-2.7%	-1.5%	-1.9%
s215 (Twilight scene)	3.2%	17.2%	6.8%	5.1%
s218 (Horse racing)	-1.2%	0.1%	-0.1%	-1.0%
s251 (Rotating disk)	-4.9%	-0.3%	-0.1%	-4.0%
s259 (Colorful World)	-4.4%	-0.2%	1.4%	-3.1%
s265 (Fountain)	-1.2%	-1.4%	-1.0%	-1.2%
Median	-1.6%	-0.5%	-0.3%	-1.6%
Average	-1.9%	0.9%	0.2%	-1.4%

Table 5.5: PSNR values for two different QP values on each sequence for the proposed filter and reference HEVC (Random Access)

	QP22						QP32					
	Reference			Proposed Method			Reference			Proposed Method		
	Y	U	V	Y	U	V	Y	U	V	Y	U	V
Ginko trees	38.91	41.09	41.00	**38.95**	**41.09**	**41.06**	32.71	36.28	34.88	**32.85**	**36.36**	**34.92**
Truck train	39.77	41.98	43.18	**39.83**	**41.98**	**43.20**	34.70	39.20	39.99	**34.72**	**39.20**	**40.00**
Cosmos flowers	39.22	39.80	40.57	**39.46**	**39.80**	**40.58**	32.04	35.65	36.22	**32.26**	**35.67**	**36.23**
Fountain	40.49	44.50	43.65	**40.56**	**44.50**	**43.65**	35.74	43.68	40.68	**35.77**	**43.68**	**40.68**
Basketball	41.41	44.87	43.92	**41.46**	**44.89**	**43.94**	37.60	40.65	39.47	**37.65**	**40.75**	**39.53**
Twilight scene	**40.27**	**41.51**	**41.23**	40.19	41.40	41.14	**36.89**	**38.73**	**37.20**	36.82	38.43	37.06
Horse racing	41.51	43.91	46.69	**41.53**	**43.91**	**46.70**	37.89	**40.48**	42.52	**37.92**	40.47	**42.52**
Rotating disk	40.54	**42.74**	43.91	**40.68**	42.74	**43.92**	36.36	40.25	**39.53**	**36.52**	**40.25**	39.51
Colorful World	39.57	**41.02**	**41.78**	**39.68**	41.00	41.76	35.21	36.55	**35.81**	**35.38**	**36.58**	35.74
Fountain	40.60	**43.42**	44.65	**40.66**	43.42	**44.66**	35.60	42.63	42.74	**35.66**	**42.63**	**42.75**
Average	40.23	**42.48**	43.06	**40.30**	42.47	**43.06**	35.47	**39.41**	**38.90**	**35.56**	39.40	38.90

5.3 Block segmentation and transform selection

The first experiment with optical flow aims to evaluate the accuracy of the segmentation of blocks using k-means as described in Section 4.1. Using only translations for blocks with splitting around object boundaries, Tables 5.6 and 5.7 show the PSNR improvement of the prediction. No refinement is done on the prediction, the output from ME using optical flow only is used. Outside of the Fountain sequence, which shows a large improvement in the two cases, the other two sequences show a much larger improvement for the 64×64 blocks case. While in the case of larger blocks the reference method which does not split the blocks shows a noticeable and expected decrease in prediction accuracy, the proposed method manages to split blocks correctly and maintains the same prediction accuracy. This proves the optical flow is able to estimate object boundaries accurately enough for encoding purposes. Figure 5.3 shows the results of segmentation on the Truck Train sequence.

Table 5.6: Increase of predicted image PSNR with 32×32 blocks

Sequence	Reference	Proposed	Increase
Truck Train	26.22dB	26.28dB	0.062dB
Fountain	32.33dB	32.85dB	0.522dB
Stockholm	25.27dB	25.55dB	0.288dB

Table 5.7: Increase of predicted image PSNR with 64×64 blocks

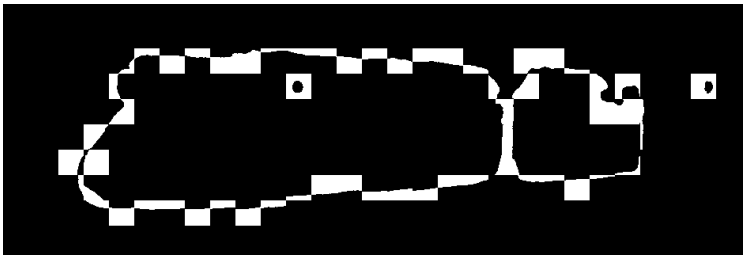
Sequence	Reference	Proposed	Increase
Truck Train	25.95dB	26.26dB	0.312dB
Fountain	32.35dB	32.86dB	0.504dB
Stockholm	24.91dB	25.48dB	0.572dB

5.4 Optical flow estimation of motion parameters

The second experiments aims to evaluate the accuracy of the predictions provided by the optical flow-based estimation and measure the improvement from the iteration process. Only the blocks that were classified as affine in Equation 4.6 are predicted with affine transforms, the rest either used the proposed segmentation with translation for each subblock or translation for the whole block. For comparison purposes, outside of

(a) Reference Picture

(b) Target Picture



(c) Result of k-means segmentation on each 32×32 block for the area around the train

Figure 5.3: Segmentation on the Truck Train sequence

the proposed 3-parameter transform selected automatically, the Z&R model and affine prediction are used to compare the prediction accuracy. For the Z&R transform, the values are computed with Equations 4.8 and 4.9 as for the 3-parameter transform, but without zeroing a parameter. For the affine transform, the OpenCV library [56] function getAffineTransform is used. For refinement, the proposed method uses the 2-step refinement described in Section 4.3. All other methods use refinement on the translation parameters only, keeping the computation time roughly equal for each method. A summary of the transforms used and the refinement methods is presented in Table 5.8.

Table 5.9 shows the prediction accuracy of each transform. Figure 5.4 shows the difference between the translation-based prediction and the proposed 3-parameter transform after refinement on the Rotating Disk sequence. The discontinuity around the rotating disk are much smaller with the proposed transform, which shows in the greatly increased PSNR compared to the translation.

Table 5.8: Overview of the different tested transforms

Transform	Dimension	Model	Parameter Estimation	Parameter Refinement
Translation	2	$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}$	Average of values in block	Square Search
Zoom or Rotation (proposed)	3	$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1+s & 0 \\ 0 & 1+s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}$ $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -r \\ r & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}$	Equation 4.8	Proposed
Zoom and Rotation	4	$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}$	Equation 4.8	Square Search ($\begin{bmatrix} v_x & v_y \end{bmatrix}^t$ only)
Affine	6	$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix}$	OpenCV function	Square Search ($\begin{bmatrix} v_x & v_y \end{bmatrix}^t$ only)

Table 5.9: Increase of predicted image PSNR with 64×64 blocks

Sequence	Proposed	ZR	Affine	Translation	Transform Type	Motion
Fountain	35.18	35.14	34.92	34.80	Scale 3 Rotation 79	Rotation
Station	24.16	24.00	23.92	23.82	Scale 426	Zoom
Rotating disk	24.88	24.76	24.31	24.08	Scale 45 Rotation 273	Rotation
Twilight scene	29.41	29.41	29.40	29.37	Scale 54 Rotation 36	Zoom
Cosmos flowers	22.32	22.35	22.48	22.20	Scale 160 Rotation 183	Camera tilting

(a) Translation

(b) Proposed 3-parameter transform

Figure 5.4: Comparison of prediction result on Rotating Disk sequence

5.5 Evaluation of the implemented method in HEVC

5.5.1 Testing conditions

The HEVC reference encoder HM14 [7] is used as the anchor to estimate the Bjøntegaard-Delta Rate (BD-R) [24] estimated bitrate savings and encoding time to compare the various methods. The compared reference methods are [16], their implementation of [15], and a 3-parameter variant of the gradient method to compare the parameter estimation only as it uses the same entropy coding as [16]. For the encoding settings, the same settings as [16] are used. The encoding mode is set to Low Delay P, and the QP values are $22, 27, 32, 37$.

To compute the encoding time, we used the following formula:

$$\Delta T = \frac{T_{target} - T_{HM_{14}}}{T_{HM_{14}}} \tag{5.2}$$

For results aggregating different encoding quality settings, the extra encoding time ΔT is averaged over all QP values.

5.5.2 Fast mode decision

The first experiment evaluates the mode decision accuracy. In this experiment, the iteration uses the same process as [16], but evaluates only one transform using the optical flow information to decide which is likely better. Table 5.10 shows how the proposed method compares to [16] and [15]. On average, the proposed method loses less than 1% in BD-R, but the required encoding time goes down from over 20% to just over 11%, about half of the time, which is expected from having to evaluate only half the affine encoding modes.

However, when compared to the gradient approach, it fails to be either faster or more efficient. In [16], the gradient method had a similar encoding time as Tsutake's Block Matching approach. It is likely that some of the speed gains of the gradient approach comes from the different computer and compiler. The gradient operations can be vectorized for a huge performance gain with a newer processor and compiler supporting the extensions. In the present thesis, Microsoft's latest version of Visual C++ 2019 (VS14.2) was used with optimizations for the most recent instructions set

extensions.

Table 5.10: Comparison of coding efficiency and encoding speed in terms of BD-Rate and ΔT

Sequence name	Proposed (model only)		Tsutake [16]		Heithausen [15]	
	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]
Station	-23.75	14.54	-24.76	24.60	-25.62	12.81
Fountain	-0.15	7.38	-0.16	16.20	-0.24	10.49
Fungus Zoom	-16.18	19.77	-16.37	26.70	-15.23	17.35
Rotating Disk	-24.07	10.23	-26.77	20.42	-29.28	11.24
Blue Sky	-4.74	8.78	-5.13	19.18	-5.52	9.23
Tractor	-3.71	8.44	-4.59	15.24	-4.82	7.68
Twilight Scene	-0.26	11.05	-0.39	20.48	-0.51	10.15
Average	-10.41	11.46	-11.17	20.40	-11.61	11.28

5.5.3 Fast parameter estimation

While the previous experiment proved that optical flow was able to determine the encoding mode accurately, finding good parameters is a more difficult problem. In this experiment, two approaches will be compared. The first method, estimation only, uses the process defined in Section 4.3, while the second method, mode + hint, performs the bad estimation detection and goes back to [16] when the initial estimated proves inaccurate.

The BD-R estimates and encoding time results are shown in Table 5.11, with mode only representing the previous experiment method. Table 5.12 shows the PSNR (dB) for all components along the bitrate (bits/s) and encoding time (s) for each QP value.

It appears that the initial estimation is often not good enough, as the results show a clear degradation in encoding efficiency, which is not acceptable for the minor increase in speed. For the second experiment, the encoder often ends up trying more values as the initial estimation is not good enough. Better heuristics for switching iteration modes are required. Looking at the detailed results in Table 5.12 with PSNR values for all variants, it appears that using an initial estimation can give an advantage for encoding time. However, it is still too inaccurate to give consistently good results. An overview of the coding efficiency for the sequence Station using Rate-Distortion curves

Table 5.11: Comparison of coding efficiency and encoding speed in terms of BD-Rate and ΔT for the variants of the proposed method

Sequence name	Estimation		Model + hint		Model only	
	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]
Station	-18.03	14.32	-23.75	14.64	-23.75	14.54
Fountain	-0.10	5.28	-0.15	7.45	-0.15	7.38
Fungus Zoom	-11.38	16.77	-16.18	19.88	-16.18	19.77
Rotating Disk	-13.22	6.98	-24.07	10.63	-24.07	10.23
Blue Sky	-3.76	5.90	-4.74	9.10	-4.74	8.78
Tractor	-1.66	5.79	-3.71	8.00	-3.71	8.44
Twilight Scene	-0.27	8.01	-0.26	11.27	-0.26	11.05
Average	-6.92	9.01	-10.41	11.57	-10.41	11.46

for each method tested, except the method with hint that showed similar results as the mode only one can be seen in figure 5.5. All methods give a much higher efficiency that the reference encoder HM, and while the proposed estimation-based method is clearly below the others, the proposed method performs very well at all bitrates compared to the existing methods.

Table 5.12: PSNR and encoding time of the variants of the proposed method

Sequence name	QP	Proposed (estimation)					Proposed (model + hint)					Proposed (model only)				
		bitrate	Y	Cb	Cr	Time	bitrate	Y	Cb	Cr	Time	bitrate	Y	Cb	Cr	Time
Station	22	2401	41.01	44.36	45.01	9185	2242	41.00	44.36	45.01	10036	2141	41.02	44.38	45.04	10332
	27	560	39.18	42.89	43.33	7307	464	39.26	42.91	43.36	8060	441	39.34	42.98	43.44	8332
	32	251	37.02	41.85	42.08	6460	214	37.21	41.83	42.13	8106	208	37.33	41.90	42.18	7458
	37	132	34.75	41.26	41.38	6077	120	34.92	41.24	41.38	6820	118	35.07	41.23	41.47	7086
Fountain	22	24344	40.14	43.33	44.41	3417	24333	40.14	43.33	44.41	3562	24333	40.14	43.33	44.41	3598
	27	12906	37.25	43.01	43.45	2821	12900	37.25	43.02	43.45	2952	12898	37.25	43.02	43.45	3009
	32	5995	33.84	42.56	42.48	2315	5998	33.84	42.58	42.47	2445	5997	33.85	42.58	42.47	2501
	37	2289	30.72	42.40	41.85	1875	2289	30.73	42.35	41.85	1999	2294	30.73	42.39	41.86	2058
Fungus Zoom	22	426.7	38.43	39.63	40.41	454	406.5	38.41	39.61	40.40	496.9	392.4	38.40	39.61	40.39	497.0
	27	118.8	34.54	37.18	38.01	313	106.8	34.53	37.19	38.01	361.7	100.3	34.53	37.20	38.03	372.1
	32	40.0	31.02	35.75	36.62	250	34.1	31.01	35.76	36.64	299.8	31.9	31.01	35.81	36.70	308.9
	37	16.4	28.17	34.86	35.83	216	13.6	28.16	34.95	35.93	266.6	13.1	28.16	35.00	35.92	278.5
Rotating Disk	22	6974	39.92	42.43	43.22	2213	6621	39.89	42.40	43.18	2338	6372	39.87	42.41	43.20	2378
	27	1765	37.65	41.04	40.85	1628	1605	37.70	41.02	40.82	1706	1416	37.75	41.05	40.86	1749
	32	683	35.23	40.00	39.10	1342	592	35.32	39.99	39.09	1454	521	35.41	40.03	39.16	1523
	37	337	32.51	39.14	37.79	1226	280	32.60	39.13	37.79	1338	258	32.76	39.16	37.82	1399
Blue Sky	22	9214	41.83	41.81	43.15	9077	9129	41.82	41.81	43.15	9179	9121	41.82	41.81	43.15	9376
	27	3110	38.89	38.92	40.63	6792	3055	38.92	38.91	40.62	7233	3039	38.93	38.92	40.62	7443
	32	1358	36.04	36.57	38.54	5743	1317	36.12	36.56	38.52	6190	1304	36.14	36.58	38.54	6374
	37	652	33.13	34.77	36.97	5081	627	33.25	34.78	36.96	5498	621	33.26	34.77	36.95	5718
Tractor	22	8280	41.46	42.71	43.43	38730	8246	41.46	42.71	43.43	39823	8151	41.46	42.71	43.44	40495
	27	3164	39.03	40.50	41.07	29761	3132	39.05	40.51	41.08	31486	3090	39.08	40.52	41.10	32113
	32	1450	36.46	38.65	39.13	24598	1431	36.50	38.65	39.13	26309	1415	36.54	38.67	39.14	26892
	37	725	33.81	37.37	37.82	21161	722	33.85	37.38	37.82	22763	716	33.88	37.37	37.82	23324
Twilight Scene	22	9626	40.07	41.36	40.92	2181	9634	40.07	41.37	40.93	2272	9633	40.07	41.36	40.92	2316
	27	2832	38.22	40.00	38.94	1430	2826	38.22	39.99	38.94	1554	2825	38.21	39.99	38.93	1587
	32	998	36.18	38.57	36.94	1206	993	36.18	38.57	36.94	1321	993	36.18	38.57	36.94	1369
	37	428	34.21	37.61	35.60	1111	430	34.23	37.63	35.61	1219	430	34.22	37.62	35.62	1272

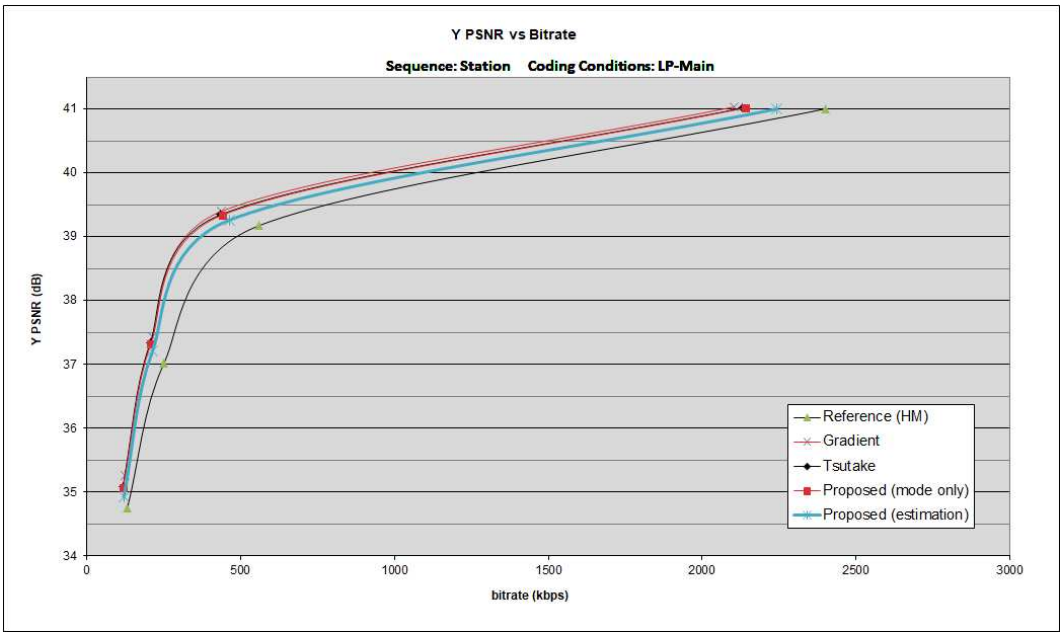


Figure 5.5: Rate-Distortion Curve of the Station sequence with the various tested methods

5.5.4 3-parameter gradient refinement

We also evaluated the proposed mode selection model with the 3-parameter gradient method that was also included in the code provided by Tsutake et al.

This allows us to compare the two types of refinement to see which works best for 3 parameters in the event you only compute one. Table 5.13 shows the obtained BD-R with the various methods. Mode (Tsutake) refers to the proposed method deciding which transform to use then doing refinement with Tsutake's algorithm. Mode (Gradient) refers to the proposed method deciding which transform to use then doing refinement with the gradient approach. We can see that contrarily to [16], where the 3-parameter gradient is always slower than their method, in this case the opposite is true, the gradient approach is always faster.

Using gradient for iteration is faster than Tsutake et almost every sequence, and more efficient in every sequence but two. In the two sequences where it is a bit slower, the efficiency is increased, especially for Rotating Disk, that gains 2% BD-R efficiency. On the Station sequence, efficiency is a bit lower, but the loss is minimal and encoding speed is better. The tractor sequence is highly challenging and the only sequence where gradient shows a lower efficiency than Tsutake, so lower efficiency is to be expected.

Table 5.13: Comparison of coding efficiency and encoding speed in terms of BD-Rate and ΔT for the variants of the proposed method

Sequence name	Model (Tsutake)		Model (Gradient)		Tsutake		Gradient 3	
	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]
Station	-23.75	14.54	-23.45	12.04	-24.76	24.60	-24.61	22.22
Fountain	-0.15	7.38	-0.15	6.29	-0.16	16.20	-0.17	10.07
Fungus Zoom	-16.18	19.77	-16.47	17.31	-16.37	26.70	-16.64	23.15
Rotating Disk	-24.07	10.23	-26.08	11.30	-26.77	20.42	-27.28	17.06
Blue Sky	-4.74	8.78	-5.59	10.91	-5.13	19.18	-5.78	15.52
Tractor	-3.71	8.44	-3.26	7.75	-4.59	15.24	-4.09	11.94
Twilight Scene	-0.26	11.05	-0.50	12.81	-0.39	20.48	-0.65	16.43
Average	-10.41	11.46	-10.79	11.20	-11.17	20.40	-11.32	16.62

Table 5.14: Comparison of Coding Efficiency and Encoding Speed when using affine mode skipping compared to only selecting the transform model

Sequence name	Model only		Model + Affine Skip	
	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]
Station	-23.75	14.54	-23.68	13.25
Fountain	-0.15	7.38	-0.15	5.10
Fungus Zoom	-16.18	19.77	-16.42	19.92
Rotating Disk	-24.07	10.23	-23.74	10.01
Blue Sky	-4.74	8.78	-4.69	8.93
Tractor	-3.71	8.44	-3.38	5.79
Twilight Scene	-0.26	11.05	-0.42	10.65
Average	-10.41	11.46	-10.35	10.52

5.5.5 Affine parameter evaluation skipping

In the previous experiments, unless neither of the estimated parameter was null, evaluation of the best transform was performed, which made the time savings limited to about half of the original time. The thresholds for mode selection are large enough to avoid too much eagerness in the skipping, which would affect greatly efficiency. The results of the fast skipping algorithm are presented in Table 5.14. With the proposed thresholds, the proposed method is able to gain 1% of encoding speed with a negligible change in efficiency. Two sequences were even improved by the proposed method, Station and Fungus Zoom. Tractor was affected most negatively for efficiency, but it was also sped up greatly. The tractor sequence is challenging because of the combined rotation and zooming motion, and the presence of object boundaries that can be hard to estimate well as they change greatly between frames. Rotation is the most challenging because of the wheels rotating a lot between frames. This is illustrated in Figure 5.6

5.5.6 Accuracy of mode decisions

To understand better the results we obtained, we investigated the differences in mode decisions between our proposed method and Tsutake [16]. For each block that would use affine coding in the reference, we checked what decision would our proposed method take, assuming it had already taken the same decisions as the reference, as the context

(a) Current picture (POC N), wheel angle 136

(b) reference picture 0 (POC N-1), wheel angle 132

(c) reference picture 1 (POC N-4), wheel angle 118

Figure 5.6: Three frames extracted from the Tractor sequence. While the wheel is in a more distant position in 5.6c than 5.6b, an acceptable prediction can be done with 5.6c using the wheel rotational symmetry

Table 5.15: Evaluation of the model prediction accuracy and affine mode skip accuracy of the proposed method compared to Tsutake

Sequence name	Sensitivity		Best block rate		Correct	Affine Skip	
	Rotation[%]	Zoom[%]	Rotation[%]	Zoom[%]	Model[%]	Bad[%]	Missed[%]
Station	14.6	90.8	0.5	33.4	89.7	3.3	70.2
Fountain	71.2	8.0	2.0	0.4	59.8	35.3	59.8
Fungus Zoom	0.0	99.2	0.3	35.6	98.4	0.0	100.0
Rotating Disk	86.9	30.9	21.6	1.3	83.9	4.5	89.7
Blue Sky	92.8	20.4	10.2	0.7	88.2	4.5	93.6
Tractor	27.4	58.6	0.9	16.7	57.0	29.3	48.6
Twilight Scene	30.8	38.6	0.8	1.2	35.5	34.4	83.3

adaptivity of entropy coding makes investigating decisions very difficult. For example, a decision that was different for in a previous block can make the optimal decision different for the current block. So the proposed method could take the right decision but be labelled as incorrect because of the different context.

We only evaluate the final coding decisions, so in the case affine prediction was the best in RDO for a given block size but the split block version, evaluated later performs better, the right decision will be classified as skipping. The results are presented in Table 5.15.

Sensitivity shows if the proposed method can detect when a block should use a given model. It is computed with the following formula:

$$\text{sensitivity}_{model} = \frac{\text{TP}_{model}}{\text{TP}_{model} + \text{FN}_{model}} \tag{5.3}$$

where TP_{model} is the true positives for a given model (predicted and used model are the same), and FN_{model} the false negatives (prediction did not chose this model when the reference used it).

The best block rate is the percentage of blocks that were encoded with this model. The correct model represents how often when either affine model was the correct answer, the proposed method chose the right affine model. It weighs the sensitivity values with the prevalence of each model in the sequence. The skip statistics show when the proposed method skipped while it should not have, and when it did not skip while it should have. While both being as low as possible is desireable, it is hard to decrease one without increasing the other, which creates a trade-off between encoding speed and efficiency.

As the previous results with the model selection show, the proposed method picks between the two models with a high accuracy. Models that are used less often, like rotation in Fungus Zoom, have a very low sensitivity because they typically offer little improvement over the other model. In most cases, the less probable model only uses the smallest possible transform. In some cases, like Twilight Scene, the efficiency was even improved. The limitations of the RDO to estimate the best compromise between quality and bitrate are a likely factor, but it is also possible that the change in occurrence of some transform models reduced the entropy coding cost by influencing the context model probabilities. Everything considered, we think the proposed skipping heuristics offer the best compromise between speed and efficiency. Increasing the lower threshold results in more skipping but also results in skipping affine transform blocks, and decreasing the upper threshold similarly results in a lot of too eager skipping. The best values depend on the sequence, so we have selected values that work acceptably for every sequence in our dataset.

5.6 Improved Entropy Coding

We implemented the differential coding as proposed in [16] to evaluate the gains it offers. We also implemented our proposed affine Affine Motion Vector Prediction (AMVP), to evaluate its gains when compared to the more simple differential coding. Table 5.16 shows the difference between the three entropy coding methods that have been imple-

Table 5.16: Comparison of Coding Efficiency and Encoding Speed with different entropy coding methods

Sequence name	Parameter coding		Differential coding		AMVP	
	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]	BD-R[%]	ΔT[%]
Station	-23.68	13.25	-24.32	12.88	-22.32	11.65
Fountain	-0.15	5.10	-0.12	5.92	-0.12	4.03
Fungus Zoom	-16.42	19.92	-16.58	18.92	-16.49	18.52
Rotating Disk	-23.74	10.01	-24.44	9.41	-23.77	9.20
Blue Sky	-4.69	8.93	-4.96	8.29	-4.54	10.41
Tractor	-3.38	5.79	-3.46	4.95	-3.25	4.46
Twilight Scene	-0.42	10.65	-0.37	9.46	-0.34	9.21
Average	-10.35	10.52	-10.57	9.97	-10.12	9.64

mented: parameter coding, differential coding using a neighbor, and AMVP.

Even without accounting for prediction that could use different reference pictures, the differential coding is able to increase coding efficiency for no additional encoding time. We believe that the variations in encoding time are likely to be from measurement error as decisions are very similar and the process should take a little more time. When using AMVP, we notice that for most sequences, the efficiency decreases compared to the reference entropy coding and is always less efficient than differential coding. The reduction in encoding time is caused by the encoder choosing less often the affine mode after RDO. This suggests that in many cases, using the proposed prediction of motion vectors uses more bits than the simple approach. The reasons for that are not entirely understood, but there are some known possibilities. The first the changes in the context models. In this approach, additional context models are required to store the index of the predictor in the candidate list and the MVD. This increases overhead as the translation part of the current MV already needs to code this information. Increasing the number of context models also makes it harder for it to adapt, as each context is used less. The second is that the model for the probability distribution of the error in the MVD is inaccurate, leading to inefficient coding of the prediction error. Using a unified MVP candidate list for both translation and affine prediction, like VVC does, is likely to be the solution.

5.7 Combining Filtering and Affine Prediction

While both approaches for improving the quality of coding videos described so far have proved their ability when used independently, using them together could lead to lowered efficiency. The postfiltering aims to reduce block noise, which mostly happens because of the discontinuities in the prediction MVs. As affine prediction aims to reduce those discontinuities by replacing them with a gradual change, it can be expected that the filter would have less left to improve.

For this experiment, we used the network that was trained with the Random Access mode in the filtering experiment. The training data excludes the three sequences from [52] that were encoded using the proposed affine prediction: Twilight Scene, Fountain, and Rotating Disk. Table 5.17 shows the improvement of the filter for each sequence.

Table 5.17: Comparison of coding efficiency in terms of BD-Rate when using the postfilter on sequences encoded by the proposed method with the current state of the art

Sequence name	Proposed (no filter) BD-R[%]	Proposed (filter) BD-R[%]	Tsutake [16] BD-R[%]	Heithausen [15] BD-R[%]
Station	-23.75	-25.88	-24.76	-25.62
Fountain	-0.15	-3.37	-0.16	-0.24
Fungus Zoom	-16.18	-13.61	-16.37	-15.23
Rotating Disk	-24.07	-29.90	-26.77	-29.28
Blue Sky	-4.74	-7.87	-5.13	-5.52
Tractor	-3.71	-4.18	-4.59	-4.82
Twilight Scene	-0.26	-0.23	-0.39	-0.51
Average	-10.41	-12.15	-11.17	-11.61

With the exception of Twilight Scene, which already failed to attain improvement in the previous experiments with the existing training dataset, all sequences but one show significant improvement when using the proposed filter, and in multiple cases now surpass in efficiency the current state of the art methods. On average, the efficiency is increased by 1.74%, with almost 1% more average efficiency than Tsutake, and half a percent compared to Heithausen. One sequence shows a large loss in efficiency: Fungus Zoom. It is the only sequence with a different definition, using WQVGA (Wide Quarter VGA format) with a definition of 416×240, while every other sequence uses FullHD (1920×1080). This is likely to be the cause as the training data cannot adapt to such

large changes in picture sizes. Because of the smaller picture size, it is also more difficult to train for these sizes as more sequences a required to provide enough sample blocks for acceptable training. The proposed method with the filter beats the existing state of the art on four sequences: Station, Fountain, Rotating Disk and Blue Sky. While Fountain achieved little gains with the affine prediction, making similar gains to the previous filtering experiments unsurprising, Station and Rotating Disk obtained both over 20% improvement from affine prediction and were still further improved significantly by the filter, by almost 5% for Rotating Disk.

Combining the two methods shows impressive results, so if a small increase in decoding time is acceptable, it is possible to beat the state of the art methods with a faster encoding time. As it is often the case in video coding, large increases in efficiency require many improvements combined together.

6 Conclusions

In this thesis, we have shown the feasibility of the CNN-based filters for HEVC-encoded video that are able to process high definition content in real time. The proposed method achieves results close to state of the art methods in a time ten to a hundred times shorter.

In the following chapter, we have demonstrated the feasibility of a new way to estimate affine motion parameters. By using values obtained by the optical flow process, we can get an estimate of motion parameters in a very short time.

Optical flow proved good enough for segmenting the image into regions that are likely to use a given transform, and was able to discern object boundaries quite well.

In the HEVC implementation, prediction of the transform type (zooming or rotation) showed very accurate, allowing for a reduction in half of the time required by Tsutake et al. [16], and achieving similar performance with the state of the art gradient methods like [15], with a small decrease in efficiency.

We proposed a method for scaling the optical flow for different reference pictures and predicting the affine parameters values directly, but while there was a gain in encoding speed, the efficiency dropped a lot. The fast optical flow method is not accurate enough for precise parameter estimation.

Using gradient for the iteration algorithm showed great results, getting in most cases better efficiency than with Tsutake's iteration method in a slightly shorter time.

To get larger speed improvements, we used the segmentation information to exclude blocks from the affine parameter search. This reduced the coding efficiency slightly, but it greatly improved the encoding speed, especially on sequences with a lot of translation motion.

We have also tested some variants to the entropy coding, taking inspiration of what was included in VVC [20]. Using the differential coding as suggested by [16] offered a small improvement over all sequences, but the VVC-inspired method was less efficient

than the straightforward parameter coding.

In our last experiment, we have combined both approaches for increasing coding efficiency and measured that their gains stack, allowing us to beat the current state of the art methods for affine prediction coding. As both methods were designed to be fast, the combined approach is able to both reduce the total computing time and increase coding efficiency.

In further work, we want to investigate some ways to improve the parameter estimation when using a different picture than the one for which optical flow was computed. We also want to improve the heuristics to reduce unnecessary estimations of affine parameters while keeping the same coding efficiency. We will also investigate the issues faced by the proposed entropy coding and find ways to improve it.

Achievements

Conferences

A. Chauvet, T. Miyazaki, Y. Sugaya, and S. Omachi, "Adaptive post filter for reducing block artifacts in high efficiency video coding," in *2016 International Conference on Multimedia Systems and Signal Processing (ICMSSP), New Taipei, Taiwan*, pp. 22–25, September 2016.

A. Chauvet, T. Miyazaki, Y. Sugaya, and S. Omachi, "Optical Flow-Based Prediction of Block Splitting," in *ICE2018 Conference, Daegu, South Korea*, November 2018

Journal papers

A. Chauvet, T. Miyazaki, Y. Sugaya, and S. Omachi, "Fast image quality enhancement for HEVC by postfiltering via shallow neural networks," *IIEEJ Transactions on Image Electronics and Visual Computing*, vol. 7, pp. 2–12, June 2019.

A. Chauvet, Y. Sugaya, T. Miyazaki, and S. Omachi, "Optical flow-based fast motion parameters estimation for affine motion compensation," *Applied Sciences*, vol. 10, pp. 729–743, January 2020.

Bibliography

[1] Cisco, "Cisco visual networking index: Forecast and trends, 2017-2022 white paper," Feb. 2019, document ID:1551296909190103.

[2] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[3] J.-R. Ohm, G. Sullivan, H. Schwarz, T. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards—including high efficiency video coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669–1684, Dec. 2012.

[4] C. Segall, V. Baroncini, J. Boyce, J. Chen, and T. Suzuki, "JVET-H1002: joint call for proposals on video compression with capability beyond HEVC," JVET, Tech. Rep., Jul. 2018.

[5] J. Chen, M. Karczewicz, Y.-W. Huang, K. Choi, J.-R. Ohm, and G. Sullivan, "The joint exploration model (JEM) for video compression with capability beyond HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, Oct. 2019.

[6] J. Sole, R. Joshi, N. Nguyen, T. Ji, M. Karczewicz, G. Clare, F. Henry, and A. Duenas, "Transform coefficient coding in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1765–1777, Dec. 2012.

[7] JCT-VC, "High efficiency video coding (HEVC) test model 14 (HM 14.0)," Fraunhofer Heinrich Hertz Institute, 2014. [Online]. Available: https://hevc.hhi.fraunhofer.de/

[8] N. Purnachand, L. N. Alves, and A. Navarro, "Fast motion estimation algorithm for HEVC," in *2012 IEEE Second International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Berlin, Germany, Sep. 2012, pp. 34–37.

[9] X. Li, R. Wang, W. Wang, Z. Wang, and S. Dong, "Fast motion estimation methods for HEVC," in *2014 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, Beijing, China, Jun. 2014, pp. 1–4.

[10] R. Khemiri, N. Bahri, F. Belghith, F. E. Sayadi, M. Atri, and N. Masmoudi, "Fast motion estimation for HEVC video coding," in *2016 International Image Processing, Applications and Systems (IPAS)*, Hammamet, Tunisia, Nov. 2016, pp. 1–4.

[11] G. J. Sullivan and R. L. Baker, "Motion compensation for video compression using control grid interpolation," in *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, Toronto, Ontario, Canada, Apr. 1991, pp. 2713–2716.

[12] H. Huang, J. Woods, Y. Zhao, and H. Bai, "Control-point representation and differential coding affine-motion compensation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 23, pp. 1651–1660, Oct. 2013.

[13] C. Heithausen and J. H. Vorwerk, "Motion compensation with higher order motion models for HEVC," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brisbane, QLD, Australia, Apr. 2015, pp. 1438–1442.

[14] M. Narroschke and R. Swoboda, "Extending HEVC by an affine motion model," in *2013 Picture Coding Symposium (PCS)*, San Jose, CA, USA, Dec. 2013, pp. 321–324.

[15] C. Heithausen, M. Bläser, M. Wien, and J. Ohm, "Improved higher order motion compensation in HEVC with block-to-block translational shift compensation," in *2016 IEEE International Conference on Image Processing (ICIP)*, Phoenix, AZ, USA, Sep. 2016, pp. 2008–2012.

[16] C. Tsutake and T. Yoshida, "Block-matching-based implementation of affine motion estimation for HEVC," *IEICE Transactions on Information and Systems*, vol. E101.D, no. 4, pp. 1151–1158, Apr. 2018.

[17] N. Zhang, X. Fan, D. Zhao, and W. Gao, "Merge mode for deformable block motion information derivation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2437–2449, Nov. 2017.

[18] L. Li, H. Li, D. Liu, Z. Li, H. Yang, S. Lin, H. Chen, and F. Wu, "An efficient four-parameter affine motion model for video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1934–1948, Aug. 2018.

[19] S. Lin, H. Chen, H. Zhang, S. Maxim, H. Yang, and J. Zhou, "Affine transform prediction for next generation video coding," ITU-T, Tech. Rep., Sep. 2015, document ITU-T SG 16.

[20] K. Zhang, Y. Chen, L. Zhang, W. Chien, and M. Karczewicz, "An improved framework of affine motion compensation in video coding," *IEEE Transactions on Image Processing*, vol. 28, no. 3, pp. 1456–1469, Mar. 2019.

[21] C. Heithausen, M. Bläser, and M. Wien, "Distance scaling of higher order motion parameters in an extension of HEVC," in *2016 Picture Coding Symposium (PCS)*, Nuremberg, Germany, Dec. 2016, pp. 1–5.

[22] K. Choi and E. S. Jang, "Fast coding unit decision method based on coding tree pruning for high efficiency video coding," *Optical Engineering*, vol. 51, no. 3, pp. 1 – 4, Mar. 2012. [Online]. Available: https://doi.org/10.1117/1.OE.51.3.030502

[23] M. B. Cassa, M. Naccari, and F. Pereira, "Fast rate distortion optimization for the emerging HEVC standard," in *2012 Picture Coding Symposium*, Krakow, Poland, May 2012, pp. 493–496.

[24] G. Bjøntegaard, "Calculation of average PSNR differences between RD-curves," in *ITU—Telecommunications Standardization Sector STUDY GROUP 16 Video Coding Experts Group (VCEG), 13th Meeting*, Apr. 2001, document VCEG-M33.

[25] H. Kim and R. Park, "Fast cu partitioning algorithm for HEVC using an online-learning-based bayesian decision rule," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 130–138, Jan. 2016.

[26] X. Shen and L. Yu, "Cu splitting early termination based on weighted svm," *EURASIP Journal on Image and Video Processing*, vol. 2013, no. 4, pp. 1–11, Jan. 2013.

[27] Y. Zhang, S. Kwong, X. Wang, H. Yuan, Z. Pan, and L. Xu, "Machine learning-based coding unit depth decisions for flexible complexity allocation in high efficiency video coding," *IEEE Transactions on Image Processing*, vol. 24, no. 7, pp. 2225–2238, Jul. 2015.

[28] L. Zhu, Y. Zhang, Z. Pan, R. Wang, S. Kwong, and Z. Peng, "Binary and multi-class learning based low complexity optimization for HEVC encoding," *IEEE Transactions on Broadcasting*, vol. 63, no. 3, pp. 547–561, Sep. 2017.

[29] M. Grellert, B. Zatt, S. Bampi, and L. A. da Silva Cruz, "Fast coding unit partition decision for HEVC using support vector machines," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 6, pp. 1741–1753, Jun. 2019.

[30] X. Zheng, Y. Zhao, H. Bai, and C. Lin, "Fast algorithm for intra prediction of HEVC using adaptive decision trees," *KSII Transactions on Internet and Information Systems*, vol. 10, pp. 3286–3300, Jul. 2016.

[31] Z. Liu, X. Yu, Y. Gao, S. Chen, X. Ji, and D. Wang, "Cu partition mode decision for HEVC hardwired intra encoder using convolution neural network," *IEEE Transactions on Image Processing*, vol. 25, no. 11, pp. 5088–5103, Nov. 2016.

[32] M. Bläser, C. Heithausen, and M. Wien, "Segmentation-based partitioning for motion compensated prediction in video coding," in *2016 Picture Coding Symposium (PCS)*, Nuremberg, Germany, Dec. 2016, pp. 1–5.

[33] P. List, A. Joch, J. Lainema, G. Bjøntegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 614–619, Jul. 2003.

[34] A. Norkin, G. Bjøntegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, and G. V. D. Auwera, "HEVC deblocking filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1746–1754, Dec. 2012.

[35] C. Fu, E. Alshina, A. Alshin, Y. Huang, C. Chen, C. Tsai, C. Hsu, S. Lei, J. Park, and W. Han, "Sample adaptive offset in the HEVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, Dec. 2012.

[36] S. Nogaki and M. Ohta, "An overlapped block motion compensation for high quality motion picture coding," in *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems*, vol. 1. San Diego, CA, USA: IEEE, May 1992, pp. 184–187.

[37] C. Li, L. Song, R. Xie, and W. Zhang, "Cnn based post-processing to improve HEVC," in *2017 IEEE International Conference on Image Processing (ICIP)*, Beijing, China, Sep. 2017, pp. 4577–4580.

[38] W. Park and M. Kim, "Cnn-based in-loop filtering for coding efficiency improvement," in *2016 IEEE 12th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*, Bordeaux, France, Jul. 2016, pp. 1–5.

[39] Y. Dai, D. Liu, and F. Wu, "A convolutional neural network approach for post-processing in HEVC intra coding," in *MultiMedia Modeling*, L. Amsaleg, G. Guomundsson, C. Gurrin, B. Jónsson, and S. Satoh, Eds. Cham: Springer International Publishing, Jan. 2017, pp. 28–39.

[40] A. Chauvet, T. Miyazaki, Y. Sugaya, and S. Omachi, "Adaptive post filter for reducing block artifacts in high efficiency video coding," in *2016 International Conference on Multimedia Systems and Signal Processing (ICMSSP)*, New Taipei, Taiwan, Sep. 2016, pp. 22–25.

[41] A. Chauvet, "Study on reducing block artefacts in HEVC video," Master's thesis, Department of Electrical Engineering, Tohoku University, Feb. 2017.

[42] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow," in *CVPR - IEEE Conference on Computer Vision & Pattern Recognition*. Boston, NA, USA: IEEE, Jun. 2015, pp. 1164–1172. [Online]. Available: https://hal.inria.fr/hal-01142656

[43] D. Rüfenacht and D. Taubman, "HEVC-EPIC: Fast optical flow estimation from coded video via edge-preserving interpolation," *IEEE Transactions on Image Processing*, vol. 27, no. 6, pp. 3100–3113, Jun. 2018.

[44] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., Apr. 1981, pp. 674–679. [Online]. Available: http://dl.acm.org/citation.cfm?id=1623264.1623280

[45] C. Liu, "Beyond Pixels: Exploring New Representations and Applications for Motion Analysis," Ph.D. Thesis, Massachusetts Institute of Technology, 2009.

[46] A. Bruhn, J. Weickert, and C. Schnörr, "Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods," *International Journal of Computer Vision*, vol. 61, pp. 211–231, Feb. 2005.

[47] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," in *Computer Vision - ECCV 2004*,

T. Pajdla and J. Matas, Eds. Prague, Czech Republic: Springer Berlin Heidelberg, May 2004, pp. 25–36.

[48] A. Chauvet, T. Miyazaki, Y. Sugaya, and S. Omachi, "Fast image quality enhancement for HEVC by postfiltering via shallow neural networks," *IEEJ Transactions on Image Electronics and Visual Computing*, vol. 7, no. 1, pp. 2–12, Jun. 2019.

[49] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," in *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[50] Intel, "Math kernel library," 2017. [Online]. Available: https://software.intel.com/en-us/mkl

[51] A. Chauvet, Y. Sugaya, T. Miyazaki, and S. Omachi, "Optical flow-based fast motion parameters estimation for affine motion compensation," *Applied Sciences*, vol. 10, no. 2, pp. 729–743, Jan. 2020. [Online]. Available: http://dx.doi.org/10.3390/app10020729

[52] Institute of Image Information and Television Engineers, *ITE/ARIB Hi-Vision Test Sequence 2nd Edition*, NHK Engineering Services, Inc., Sep. 2009.

[53] Derf, "Xiph.org video test media [derf's collection]," 2018, [Online; accessed 10-October-2018]. [Online]. Available: https://media.xiph.org/video/derf/

[54] MulticoreWare, Inc., "x265 HEVC encoder version 2.1+69-c97c64ab8b8e," 2016. [Online]. Available: http://x265.org/

[55] I. Katsavounidis and L. Guo, "Video codec comparison using the dynamic optimizer framework," in *Applications of Digital Image Processing XLI*, A. G. Tescher, Ed., vol. 10752, International Society for Optics and Photonics. San Diego, CA, USA: SPIE, Sep. 2018, pp. 266 – 281. [Online]. Available: https://doi.org/10.1117/12.2322118

[56] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.