



המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

TECHNION INSTITUTE OF TECHNOLOGY
PROJECT IN IMAGE PROCESSING AND ANALYSIS
234329

Hybrid Video Coding at High Bit-Rates

Students:

Roy Shchory

Ron Gatenio

Supervisor:

Yehuda Dar

July 2017

Table of Contents

1. Theoretical background	5
1.1. Motivation for Hybrid Video Coding	5
1.2. Transform Coding	5
1.2.1. Motivation	5
1.2.2. Karhunen-Loeve Transform – KLT	5
1.2.3. Discrete Cosine Transform – DCT	5
1.3. Motion Estimation and Compensation	6
1.3.1. Motivation	6
1.3.2. Group of Pictures – GOP	6
1.3.3. The Motion Estimation Process	7
1.3.4. Residual Images	7
1.4. HEVC – High Efficiency Video Coding	8
1.4.1. Block partitioning	8
2. Research question and project goals	10
3. Proposed approach	10
4. Part 1 – simplified video encoder	11
4.1. The hybrid video encoder implementation	11
4.1.1. The compressor's configurations	11
4.1.2. Extracted information from the compression process	11
4.2. Testing	12
4.3. Results	13
4.3.1. Test 1 result	13
4.3.2. Test 2 result	13
4.3.3. Test 3 result	14
4.3.4. Test 4 result	14
4.4. Conclusions	14
5. Part 2 – HEVC	15
5.1. JCT-VC's HEVC application	15
5.1.1. Configuration file	15
5.2. Our research tool	16
5.3. Changes to the HEVC application	16
5.3.1. Frames.mat – the TAppDecoder's output	17
5.4. The Analyzer	18
5.4.1. The Viewer	21
5.4.2. Implementation of the Analyzer	22

5.5. The Decompressed Residual Images	23
5.6. More examples from the Analyzer	24
5.7. Testing	25
5.8. Results	26
5.8.1. The credibility of our statistical results	26
5.8.2. Correlation between pixels in the residual image	26
6. Project conclusions	27
7. Project summary	27
8. Appendix - user instructions	28
8.1. HEVC's TAppDecoder and TAppEncoder	28
8.1.1. Compile the applications	28
8.1.2. Possible problems with the applications	28
8.1.3. Run	29
8.2. The Analyzer	29
9. References	30

Abstract

In this project we explore the prevalent hybrid video-coding concept that joins transform-coding and motion-compensation. Specifically, we study the necessity of transforming the motion-compensation prediction residuals for their coding at high bit-rates.

Our research relies on empirical statistics from a simplified motion-compensation procedure implemented in Matlab, and also from the reference software of the state-of-the-art HEVC standard. Our results show that the correlation among the motion-compensation residuals gets lower as the bit-rate increases, supporting the marginal use of transform coding at high bit-rates (i.e., the residuals are directly quantized).

We also developed a research tool that provides data from intermediate stages of the HEVC. The data mainly include motion-compensation residuals, motion vector data, block and frame types, and bit-budget of components. It is formatted in a structure suitable for easy usage in Matlab for future research projects.

1. Theoretical background

1.1. Motivation for Hybrid Video Coding

Modern video coding utilizes the hybrid approach where motion estimation/compensation is utilized together with transform-coding. Instead of coding every frame in the video as a single still image, we predict blocks from past reconstructed frames, quantize and code only the prediction error (residual). Since the prediction error typically has better statistical properties than the original frame pixels, it can be represented with a lower bit-rate (despite the bit-rate overhead of describing motion data).

1.2. Transform Coding

1.2.1. Motivation

The goal of image compression is to achieve a representation that requires fewer bits for encoding than the original image. In images, such a representation is attainable because they mostly contain data that doesn't contribute to the quality of the image. Most images are not a random collection of pixels, similarities exist between neighboring pixels. If we can find a transformation (that can also be easily reversed) that decorrelates the data we can have a more efficient representation of the image. The commonly used approach of lossy compression reduces the representation bit-cost at the expense of allowing some amount of distortion in the reconstructed signal. For example after we apply the DCT transform we receive a subset of coefficients containing most of the signal energy, by saving only that subset instead of all the coefficients we reduce the bit cost at the expense of some distortion.

1.2.2. Karhunen-Loeve Transform – KLT

The Karhunen-Loeve Transform (KLT) is the optimal unitary transform that provides:

- Optimal energy compaction for any number of coefficients.
- Decorrelating the signal components.

The KLT matrix is formed from the orthonormal eigenvectors of the autocorrelation matrix $R_X = E[xx^T]$ where x is the signal represented here in its column-vector form.

We calculate the KLT by:

$$U^* R_X U = \Lambda, \text{ where } \Lambda = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \Rightarrow T_{kl} = U^* T, T_{kl}^{-1} = U$$

The KLT diagonalizes the autocorrelation matrix, therefore we get a representation with uncorrelated components.

1.2.3. Discrete Cosine Transform – DCT

The $N \times N$ DCT, T_{DCT} matrix elements are defined as:

$$T_{DCT}(k, n) \begin{cases} \frac{1}{\sqrt{N}} & \text{for } k = 0, 0 \leq n \leq N - 1 \\ \sqrt{\frac{2}{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right) & \text{for } 1 \leq k \leq N - 1, 0 \leq n \leq N - 1 \end{cases}$$

where k is the matrix row (from 0 to $N-1$) and n is the matrix column (from 0 to $N-1$).

The DCT matrix is real-valued unitary matrix. In general, the $N \times N$ autocorrelation matrix of the form:

$$Q = \begin{bmatrix} 1 - \alpha & -\alpha & 0 & \dots & 0 \\ -\alpha & 1 & -\alpha & 0 & \vdots \\ 0 & -\alpha & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & 1 & -\alpha \\ 0 & \dots & 0 & -\alpha & 1 - \alpha \end{bmatrix}$$

has the $N \times N$ DCT matrix as its KLT.

Furthermore, the DCT approximates the KLT for the first-order Markov processes:

Let φ be a signal with R_φ as its autocorrelation matrix. If φ is a first order Markov process, then we get

$$R_\varphi = \begin{bmatrix} 1 & \rho & \dots & \rho^{N-1} \\ \rho & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \rho \\ \rho^{N-1} & \dots & \rho & 1 \end{bmatrix}$$

For $\beta^2 = (1 - \rho^2)(1 + \rho^2)$ and $\alpha = \frac{\rho}{1 + \rho^2}$, we get:

$$\beta^2 R_\varphi^{-1} = \begin{bmatrix} 1 - \rho\alpha & -\alpha & 0 & \dots & 0 \\ -\alpha & 1 & -\alpha & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1 & -\alpha \\ 0 & \dots & 0 & -\alpha & 1 - \rho\alpha \end{bmatrix}$$

Since $\beta^2 R_\varphi^{-1} \approx Q$ for ρ close to 1, and that R_φ^{-1} and R_φ have the same eigenvectors, we have that the rows in DCT are eigenvectors of R_φ implying that it approximates the KLT for such signals.

1.3. Motion Estimation and Compensation

1.3.1. Motivation

The motion estimation process looks for the best prediction macroblock in the reference frame. For compression, we use redundancy between adjacent frames by defining a decompressed frame as a reference and subsequent frames are predicted from the reference using motion estimation. The motion estimation process examines reference frames to identify blocks that have not changed. Motion vectors and prediction residual blocks are coded instead of blocks of pixels.

1.3.2. Group of Pictures – GOP

We divide the sequence of frames to groups of pictures of a fixed size and define each frame as one of three.

The first frame is of type *I* (not exclusively the first frame) which is encoded using spatial prediction and transform coding of the residual.

The two types of frames that get predicted are of type B and type P.

Type P- the frame gets divided into macroblocks, for each macroblock we estimate the Motion Vector per past frames in the decoded frame storage. We then get a predicted frame per the past frame and the MV. We subtract the estimated frame from the original frame and get the residual. We then proceed to transform and quantize the error, in the end we encode the error and the motion vector. The error gets dequantized, inverse transformed added to the predicted P frame and saved in the frame storage.

Type B-the same as type P accept the prediction uses past and future frames (that were already decompressed).

1.3.3. The Motion Estimation Process

We apply prediction among pixels in adjacent frames. We divide the frame to non-overlapping macroblocks, we then define a search region for each macroblock. The search region is defined by a search parameter determining how far from the edges of the macroblock the search is conducted.

The search is done as follows – we look for the best suiting macroblock in the search region by checking the MSE (or sum of squared errors, or sometimes using another metric such as sum of absolute values of the error) and choosing the best one (in the sense of minimal error).

1.3.4. Residual Images

As mentioned earlier in video compression we are working with a residual image, an image which describes the difference between the predicted image and the original one.

Let us remember the next definitions:

1.3.4.1. Stationary Image

Given a zero-mean random vector x , the property:

$$r(k) \triangleq E[x_i x_{i+k}] = E[x_j x_{j+k}], \forall i, j$$

Is termed stationary.

Specifically, and the implied form of the autocorrelation matrix is:

$$R_x = \begin{bmatrix} r(0) & r(1) & \dots & r(N-1) \\ r(1) & r(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & r(1) \\ r(N-1) & \dots & r(1) & r(0) \end{bmatrix}$$

Noting that the covariance and autocorrelation matrices of any stationary sequence are Toeplitz.

1.3.4.2. The First-Order Markov Process

$$r(k) = \sigma^2 \rho^{|k|}, 0 < \rho < 1$$

where σ^2 is the variance.

- A widely-used model for adjacent-pixel correlation in images.
- The autocorrelation matrix:

$$R_x = \begin{bmatrix} 1 & \rho & \dots & \rho^{N-1} \\ \rho & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \rho \\ \rho^{N-1} & \dots & \rho & 1 \end{bmatrix}$$

In our previous work we explored the second order statistics of motion-compensation residuals and empirically noticed that they follow the first order markov model where ρ is relatively low.

1.4. HEVC – High Efficiency Video Coding

High Efficiency Video Coding (HEVC or H.265 or MPEG-H) is the newest video coding standard of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). Its main goal is to significantly improve compression performance relative to existing standards [1]. The standard was approved in January 2013.

HEVC introduced an enhanced hybrid spatial-temporal prediction model, with flexible partitioning (Coding Tree Units), and 35 directional modes for intra prediction (vs the 9 directional modes in H.264) [2]. Furthermore, HEVC has improved variable-block-size segmentation, improved motion vector prediction, and improved motion compensation filtering.

On the down side, HEVC is more computationally expensive due to the larger prediction units and expensive motion estimation.

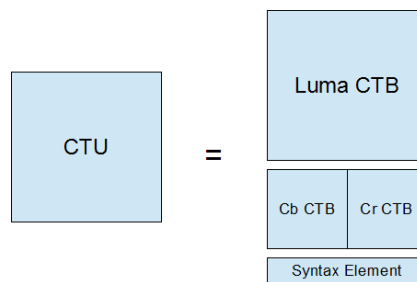
1.4.1. Block partitioning

Due to the increasing frame sizes in modern videos, there was a need for larger macroblocks to encode their motion vectors efficiently. On the other hand, small details are still important and sometimes there is a need to perform prediction and transformation at the granularity of 4x4.

To overcome this problem, the HEVC replaced the macroblock and sub-macroblocks (the fixed sized blocks from H.264) with a quadtree structure for the block partitioning. This structure allows a more flexible block partitioning.

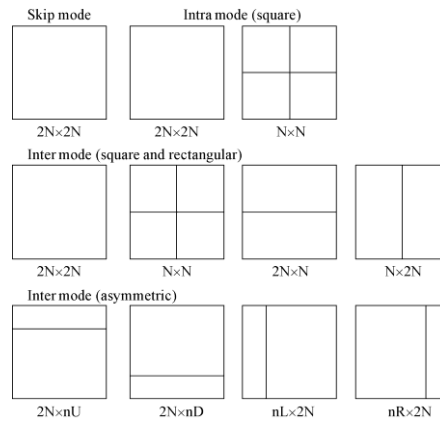
The HEVC's block partitioning structure introduces four different block concepts: CTU (Coding Tree Unit), CU (Coding Unit), PU (Prediction Unit), and TU (Transform Unit).

The Units are logical coding units that are encoded into an HEVC bit-stream. Each Unit usually consists of three blocks (Luma, Cb, Cr) that are portions of a video frame buffer, and of an associated syntax elements. The blocks are noted as CTB, CB, PB, and TB respectively [3].



HEVC divides the picture into CTUs of the same size (defined in the sequence parameter set): 64x64, 32x32, or 16x16. However, in some parts of the video frame, the CTU may be too big to decide whether to perform inter-picture prediction or intra-picture prediction. Thus, each CTU can be differently split into multiple CUs and each CU becomes the decision-making point of prediction type. A CU's size can be all the way from the same size as the CTU to as small as 8x8.

The CU is good enough for prediction type decision, but it could still be too large to store motion vectors (inter prediction) or intra prediction mode. Thus, each CU can be split to PUs depending on the temporal and/or spatial predictability [4].



After the prediction is made, the residual is coded with a DCT-like integer transformation. The CUs could be too big for this. They might contain both detailed part (high frequency) and a flat part (low frequency). So they are divided again into TUs (doesn't have to be aligned with the PUs).

2. Research question and project goals

Based on the theoretical background, the main purpose of the transform is to de-correlate the input residual blocks, but what if the residual is already de-correlated?

Our main research question is to determine if the transformation is really needed when coding in high bitrate.

Coding at high bitrates enables us to refine the prediction process by using smaller motion-estimation blocks (coding more motion vectors), by storing better reference frames (because the quantization process can be less damaging to the quality of the frame), and so on. Better prediction should produce i.i.d like residuals, which have very small correlation. This means the transform coding might not be necessary

Another goal of this project is to develop a research tool for providing data from intermediate stages of the HEVC. The data will mainly include motion-compensation residuals, motion vector data, block and frame types, and bit-budget of components. It will be formatted in a structure suitable for easy usage in Matlab.

3. Proposed approach

Initially, we tested our theory on a simplified video encoder that we developed in Matlab, based on the hybrid video coding design. We evaluated the encoder on a video with various parameters affecting the quality of the prediction, then we evaluated the correlation among samples in the residual images.

Later, we will work with a state-off-the-art HEVC encoder. We will modify its source code in order to extract information from its intermediate stages. Then we will analyze the data in Matlab, and test our theory.

We will also develop a research tool in Matlab in order to analyze the HEVC data.

4. Part 1 – simplified video encoder

In the final project for the course "Signal and Image Processing by Computer" (236327) we created our own hybrid video encoder, based on the H.264 algorithm. The purpose of the project was to study the bit-allocation problem on a simple implementation of H.264.

Our goal in this part of the project is to expand our previous implementation, and to use it to test our theory.

The Matlab code for this part is located in the folder OurH264.

4.1. The hybrid video encoder implementation

Our code is structured exactly like the H.264 block diagram. The main class, VideoCompressor, can encode and decode videos, and extract information from the process.

Our code also takes advantage of Matlab's parallel pool in order to run faster. The motion estimation stage can be ran in a multithreaded environment.

4.1.1. The compressor's configurations

Every VideoCompressor object can be configured with different parameters.

- Search algorithm function – the function that will be used to estimate the motion vectors in the motion estimation stage.
- The macroblocks' size – the size of the macroblocks that will be used to divide the frames in the motion estimation and transform stages.
- The search parameter – defines the search region around the macroblocks.
- The cost function – will be used in the motion estimation stage. The motion estimator will choose the reference block that produces the lowest cost. We implemented the following cost functions: MSE, MAD and MMSE. More can be implemented and used.
- The GOP structure – a sequence of 'I's and 'P's that define the GOP. Note that we did not implement B frames support.
- Interpolation factor – defines the scale factor for the reference image in the motion estimation stage. This value can be any power of 2. Increasing this factor allows a more precise prediction.
- Transformation preferences – These preferences include the transformation type and the block size. Note that I frames and P frames can use different transformations.
- The number of terms to keep after the transform process (K-term approximation). Note that I frames and P frames can use different values.
- The number of bits per sample to be used in the quantization stage. This value defines the step size of the quantization. Note that I frames and P frames can use different values.
- QP factor for the reference frames – an option to encode and decode all the reference frames with a specified QP value using the HEVC code (that will be explained in the next part of the project).

4.1.2. Extracted information from the compression process

Our encoder extracts information, from the compression process, for each frame. The extracted information includes:

- The original frame
- The predicted frame – the frame that is predicted from the motion vectors and reference frame.
- The prediction residual – the residual error that remains after the prediction process.
- The transformed residual – the residual after the transformation process.
- The quantized residual – the residual after being transformed and quantized.

- The inversed transformed residual – the transformed residual after being applied the inverse transform.
- The reconstructed frame – the sum of the inversed transformed residual and the predicted frame. This frame is the one that will be created by the decoder.
- The motion vectors for each macroblock in the frame. Note that in our implementation, the reference frame is always the previous frame.
- Information about the transformation – the block size that was used, the number of terms that were kept, and the energy compaction of the transformation stage.

After the compression process, we passed this information to a statistical analyzer that extracted statistical information for the whole video. The extracted statistical parameters include the mean, variance, and the correlation of adjacent pixels in the video. The statistics can be extracted for each image type, but we used it only for the residual images.

4.2. Testing

Our goal was to determine the effects of some of the compression parameters on the correlation between every two adjacent pixels in the residual images. The parameters that we tested were:

- The interpolation factor – this parameter defines the scale given to the reference frame in the motion estimation stage. We assume that by increasing this value, the prediction will improve. Then, based on our theory, the residuals correlation will decrease.
- The QP parameter of the first frame – we chose to compress and decompress the first frame of the video (the "I" frame). We defined that when the QP is -1, no compression and decompression was used on this frame. Lower QP means better quality for the first frame. Therefore, we expect better prediction (in comparison to the original video) from lower QPs. Based on our theory, lower QPs will result in less correlation.
- The macroblocks' size – with smaller macroblocks the prediction can be more refined, therefore we expect better lower correlation for smaller macroblock sizes.

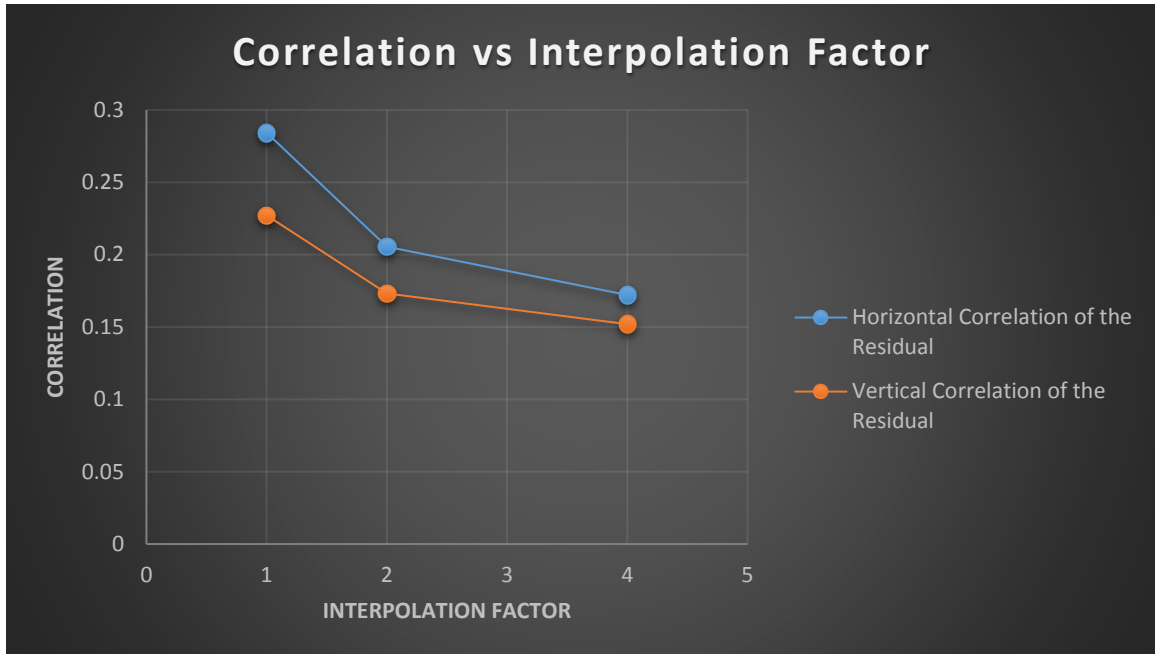
We ran our tests on the video "Old Town Cross" [5]. The GOP structure that we used was all "P" frames except for the first one, which was an "I" frame.

We conducted four major tests for each of the parameters. In the first three test, we played with only one parameter, while constraining the other two. The test were setup as follows:

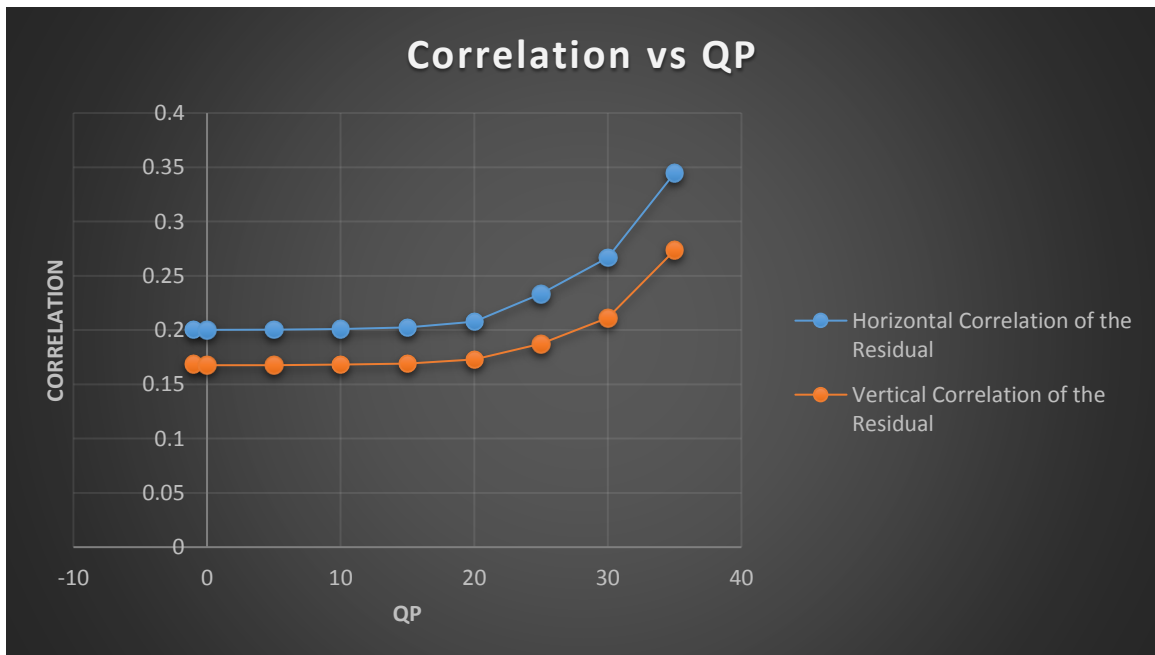
1. Interpolation factor test – we ran the encoder with a fixed search region of 5, fixed macroblock size of 8, fixed QP of -1, and with interpolation factors of 1, 2, and 4.
2. QP test – we ran the encoder with a fixed search region of 5, fixed macroblock size of 8, fixed interpolation factor of 2, and with QPs of -1, 0, 5, 10, 15, 20, 25, 30, and 35.
3. Macroblock size test - we ran the encoder with a fixed search region of 5, fixed interpolation factor of 2, fixed QP of -1, and with macroblock sizes of 4, 8, 16, 32, and 64.
4. Best prediction test – we ran the encoder with a fixed search region of 5, fixed macroblock size of 2, fixed interpolation factor of 4, and fixed QP of -1. In this test, we expect the best prediction, and therefore the lowest correlation.

4.3. Results

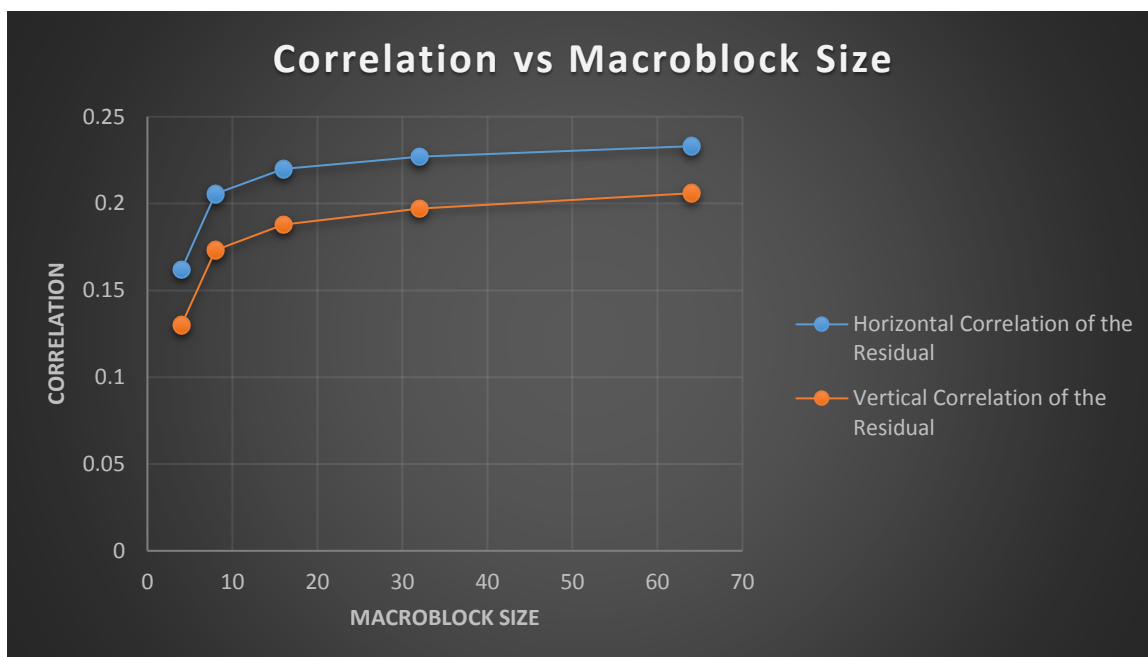
4.3.1. Test 1 result



4.3.2. Test 2 result



4.3.3. Test 3 result



4.3.4. Test 4 result

The correlation of the horizontal adjacent pixels was 0.0185, and the correlation of the vertical adjacent pixels was 0.0113. That is about 10 times lower than the lowest correlation that we found in the previous tests.

4.4. Conclusions

The results match our prediction. When we improve the prediction, the correlation decreases. When we tested with our best prediction, we saw little to no correlation.

5. Part 2 – HEVC

Our first goal in this part was to develop a research tool for providing data from intermediate stages of the HEVC. The tool will allow the user to analyze data that includes motion-compensation residuals, motion vector data, block and frame types, and bit-budget of components.

After the development of the tool, we will use it to conduct our study.

5.1. JCT-VC's HEVC application

We chose to work with the HEVC engine developed by the Joint Collaborative Team on Video Coding (JCT-VC) – HM 16.7. Their open source code project can be downloaded from one of their repositories [6]. The software includes both encoder and decoder functionality.

The project can be opened via Visual Studio (the .sln files can be found in the build directory). In the solution, nine projects can be found. The main projects are the TAppEncoder and TAppDecoder. Both can be easily compiled from there.

The TAppEncoder.exe expects several argument parameters (use --help for more details). The two important ones are "-i" for the input video, and "-c" for the configuration file name.

The TAppEncoder.exe uses a given configuration file that specifies all the presets of the encoding process. Several configuration file templets can be found under the cfg directory.

5.1.1. Configuration file

The configuration file defines the values to the encoder's parameters. If a parameter is not defined in the configuration file, it will be given its default value. Note that several configuration files may be used (with repeated "-c" in the command line). The encoder's parameters will be defined by the last value encountered on the command line.

All the parameters are specified in the software-manual [7]. Here are some that we found to be important:

- General I/O parameters: *BitstreamFile*, *ReconFile*, *FrameRate*, *FrameSkip* (number of frames to be skipped in input), *SourceWidth*, *SourceHeight*, *FramesToBeEncoded* (number of frames from the video to be encoded).
- GOP parameters: *GOPSize* and *Frame1*, *Frame2*, ... These parameters define the structure of the GOP. For each frame in the GOP you can define the type (I, P, or B), QP offset (an offset to add to the global QP parameter), reference pictures, and more.
- *DisableIntraInInter* – this flag disables intra PUs in inter slices. In our research, we focused only on inter predictions, so this flag was very useful.
- *IntraPeriod* – the period of I-frames. If set to -1, only the first frame will be an I frame.
- *QP* – the base value of the quantization parameter (0-51).
- *TransquantBypassEnableFlag* and *CUTransquantBypassFlagForce* – when both are true, the encoder will bypass the transform, quantization and filtering stages at the CU level. This flag will help us to test a compression without the transform.
- *RateControl* and *TargetBitrate* – when *RateControl* is true, the encoder will try to compress the video with the given *TargetBitrate*. The Encoder will dynamically change the QP values for different frames in order to reach its target bitrate. Sometimes the encoder will miss the target.

5.2. Our research tool

The research tool that we developed is consisted of two parts: the C++ code that creates a Frames.mat file from the TAppDecoder, and the Matlab code that analyzes the data from Frames.mat.

5.3. Changes to the HEVC application

In order to create such tool we had to modify the source code of the HEVC, and use it to create a .mat file that will be analyzed in Matlab.

Initially we had to examine the HEVC's source code to find the intermediate parameters that are important to us. We wanted to save for each frame its residual, prediction, transform coefficients, and the block partitioning structure.

We started our work by examining the TAppEncoder's source code, but we ran into some issues. The encoder's code is massive, going over it statically was very hard, and the lack of documentation did not contribute to our examination. In addition, the encoder's slow run time made it very hard to analyze the code dynamically.

We chose to take a different approach, and moved to examine the TAppDecoder instead. The decoder is much faster, thus easier for debugging. Unlike the encoder, the decoder goes straight to the task of reconstructing the frames using the bitstream's data. The decoder does not do all of the massive decision making process that the encoder goes through. In addition, the bitstream format that is exported by the encoder rarely changes. Therefore, the same decoder could be suitable to work with various HEVC encoders.

A drawback of analyzing data at the decoder, is that the input frames are not available, and therefore the considered residuals are the decompressed residuals, differing from the precompressed residuals. Nevertheless, we can assume that at high bit-rates, the decompressed residuals represent well enough the statistics of the precompressed residuals.

Most of the extracted information is read from two functions: `TDecCu::xReconInter`, and `TComTrQuant::invRecurTransformNxN`. From the first, we extract the data from each CU in each frame (including their motion-compensation data), and from the second, we extract the TUs (including the transform coefficients before and after dequantization).

In order to export all of the data to Matlab, we created a new project, StatisticsCollector, inside the Visual-Studio's solution. We wrote a class named DataExtractor that uses the Matlab's engine from within the TAppDecoder. An instance of this class is created in the main function (in decmain.cpp) and it is used throughout the decoder's code. At the end of the decoding process, a Frames.mat file is created in the current folder. This file can be opened via Matlab, and it contains all the data that was extracted from the decoder.

The documentation for the class's functions can be found in DataExtractor.h.

5.3.1. Frames.mat – the TAppDecoder's output

This file contains one variable, a cell array. The first cell contains the decoding time in seconds. From there, each cell contains a struct that identifies with one specific frame. We will explain about each of its fields.

The screenshot shows the MATLAB workspace with a variable named 'Frames' of type '1x30 cell'. The first cell contains a scalar value '1.9758e+03'. The second cell contains a struct. The following table shows the fields of this struct:

Field	Size	Class	Value
poc	1x1	uint32	1
QP	1x1	uint8	25
gopType	1x1	char	'B'
frameHeight	1x1	uint32	720
frameWidth	1x1	uint32	1280
totalBytes	1x1	double	1426
residual	720x1280	double (sparse)	720x1280 sparse double
prediction	720x1280	uint8	720x1280 uint8
reconstruction	720x1280	uint8	720x1280 uint8
coeffs_before_dq	720x1280	double (sparse)	720x1280 sparse double
coeffs_after_dq	720x1280	double (sparse)	720x1280 sparse double
culInfo	1x1629	struct	1x1629 struct

The 'culInfo' field is expanded to show a table of fields for each frame:

Fields	uiTPelX	uiBPelX	uiLPelY	uiRPelY	width	height	partitionShape	predictionMode	predictionUnits	transformUnits
1	0	15	0	15	16	16	'2Nx2N'	'INTER'	1x1 struct	[]
2	0	15	16	31	16	16	'2Nx2N'	'INTER'	1x1 struct	[]
3	16	23	0	7	8	8	'2Nx2N'	'INTER'	1x1 struct	[]
4	16	23	8	15	8	8	'2Nx2N'	'INTER'	1x1 struct	1x1 struct
5	24	31	0	7	8	8	'2Nx2N'	'INTER'	1x1 struct	[]
6	24	31	8	15	8	8	'2Nx2N'	'INTER'	1x1 struct	[]
7	16	31	16	31	16	16	'2Nx2N'	'INTER'	1x1 struct	[]
8	0	31	32	63	32	32	'2Nx2N'	'INTER'	1x1 struct	[]
9	32	63	0	31	32	32	'2Nx2N'	'INTER'	1x1 struct	[]

The 'predictionUnits' field for frame 4 is expanded to show:

Field	Size	Class	Value
xP	1x1	uint32	16
yP	1x1	uint32	8
width	1x1	uint32	8
height	1x1	uint32	8
motionVectors	1x2	struct	1x2 struct

The 'motionVectors' field is expanded to show:

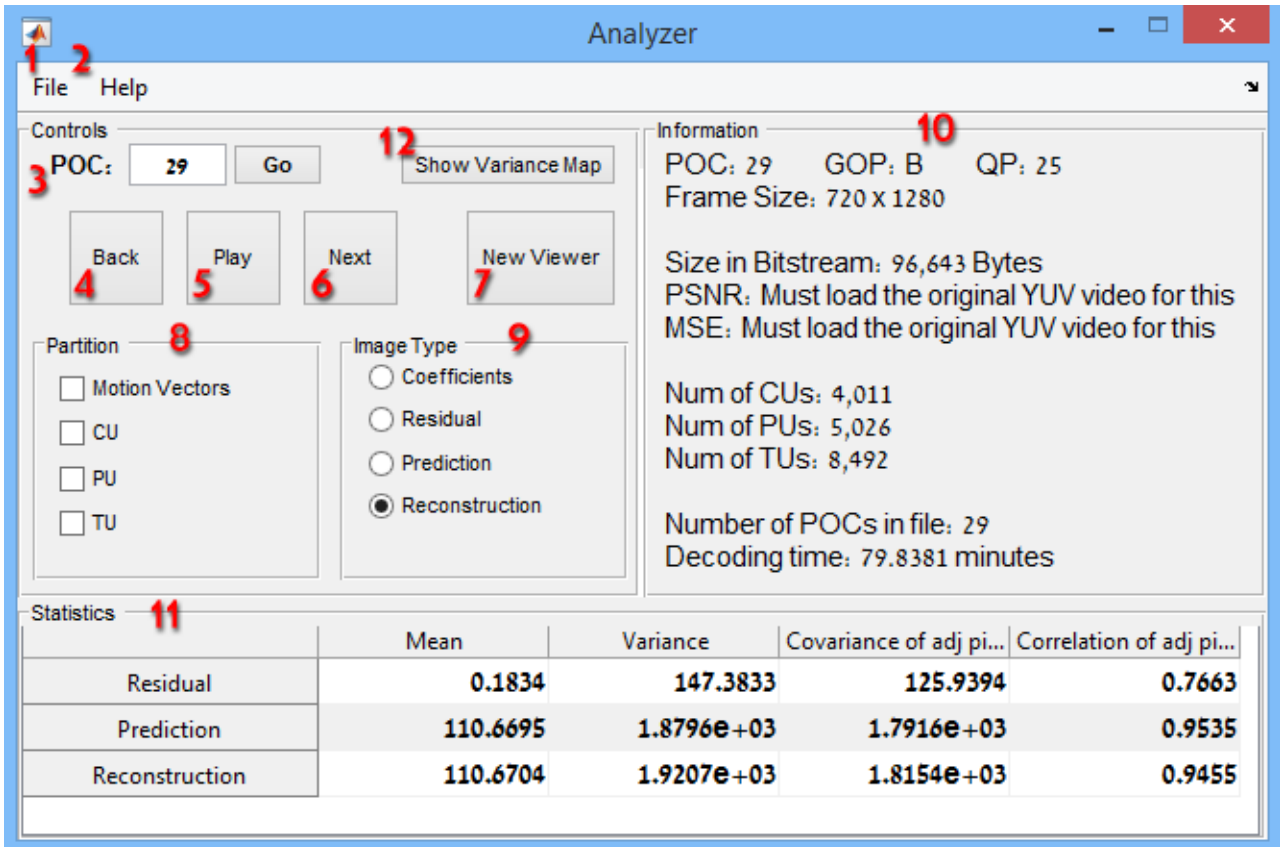
Fields	xP	yP	horizontal	vertical	referenceFramePOC
1	16	8	0	0	0
2	16	8	0	0	2

- poc – the POC (Picture Order Count) is used to identify the location of the image in the sequence. Every frame has one. Notice that the first valid POC will always be 1 because we don't save "I" frames.
- QP – the QP (Quantization Parameter) of the frame.
- gopType – a char that symbolize the GOP (Group of Pictures) picture type: I, P, or B.
- frameHeight, frameWidth – the size of the frame.
- totalBytes – the size in bytes of the encoded frame in the bitstream.
- residual – a sparse matrix of the decompressed residual image.
- prediction – the decompressed prediction image.

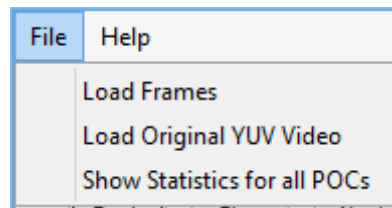
- reconstruction – the reconstruction of the frame by the decoder. This image is technically the sum of the residual and the prediction.
- coeffs_before_dq – a sparse matrix of the frame with the quantized transform coefficients. Side note: the nonzero coefficients can only be found inside a Transform Unit.
- coeffs_after_dq – a sparse matrix of the frame with the dequantized transform coefficients.
- cuInfo – a struct array that stores information about all of the CUs in the frame. Each cuInfo element contains the following information:
 - Location and size of the CU in the frame (uiTPeIX, uiBPeIX, uiLPeY, uiRPeY, width, height).
 - partitionShape – The partition shape of the Prediction Units in the CU (2Nx2N, NxN, Nx2N, etc.).
 - predictionMode – The prediction mode of the CU (INTER or INTRA). Note that in this project we only focused on inter predictions. Therefore, the only possible value for now is INTER.
 - predictionUnits – A struct array of the PUs. The array stores information about the location of each PU and there motion vectors.
 - transformUnits – A struct array of the TUs. The array stores information about the location of each TU.

5.4. The Analyzer

The Analyzer is the tool we developed in order to analyze the Frames.mat data. We will now go over all of its features.



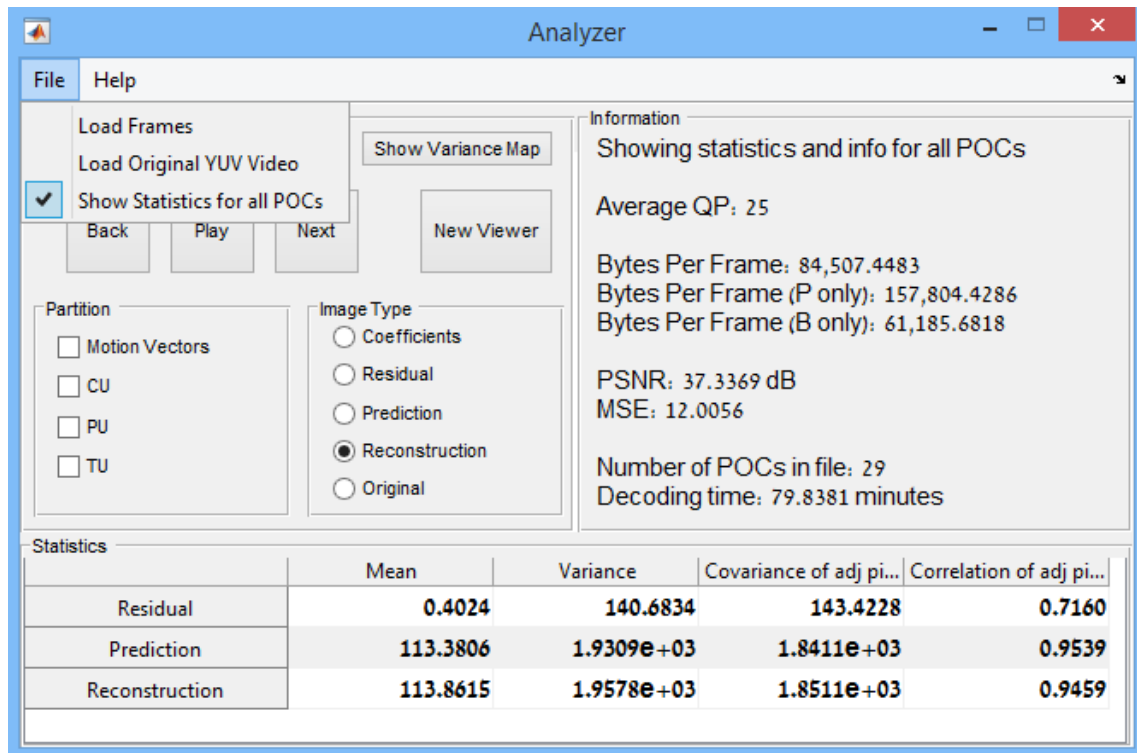
1. The File menu



- Load frames – browse for a different mat file, and load it. The file must be an output of the TAppDecoder.exe. Side note: there several ways to load a Frames.mat data, to the Analyzer. When initiated, the Analyzer will search the Matlab Workspace for a Frames variable to load. If not found, the Analyzer will search for a Frames.mat file in the Analyzer folder. If not found, the Analyzer will prompt a browse dialog to locate a Frames.mat file.
- Load original YUV video – browse for the original video that was encoded. The Analyzer will use the original video to calculate the MSE and PSNR between the original frame and the reconstructed frame.

	Mean	Variance	Covariance of adj pi...	Correlation of adj pi...
Residual	0.1834	147.3833	125.9394	0.7663
Prediction	110.6695	1.8796e+03	1.7916e+03	0.9535
Reconstruction	110.6704	1.9207e+03	1.8154e+03	0.9455

- Show statistics for all POCs – gathers statistics from all available POCs, and previews the results in the information and statistics panels.

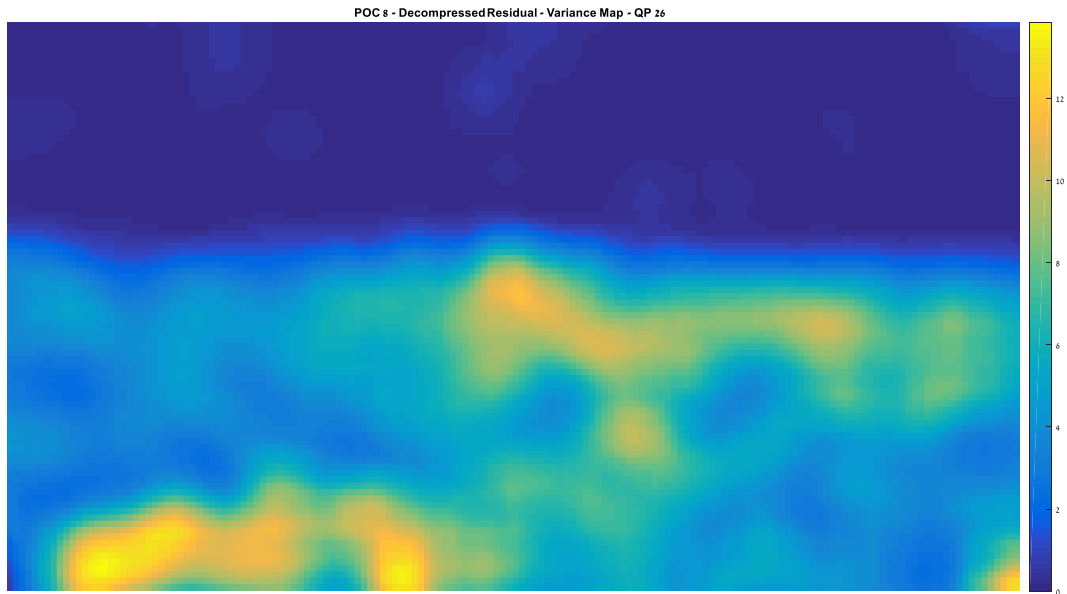


Clicking on this option again will show the current POCs information and statistics again. Moving to a new POC will also uncheck this option.

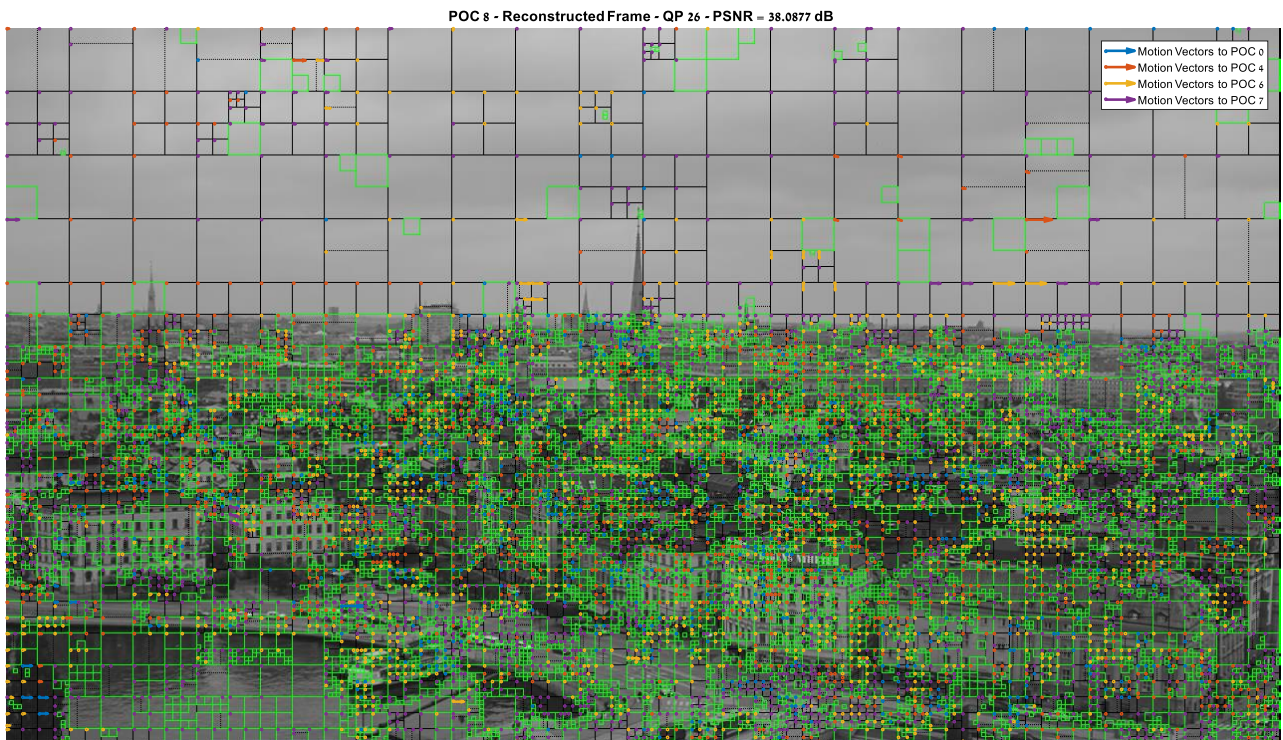
2. The Help window – pops up a message box with some information about the analyzer.
3. POC control – type a POC number and click on Go.
4. Back button – go to the previous POC.
5. Play button – when clicked, it will cycle through all of the POCs in the file, starting from the current POC.
6. Next button – go to the next POC.
7. New viewer button – this will open a new viewer window. The viewer will be explained about later.
8. The partition panel – when the viewer is opened, you can select which types of partitions to show on the frame.
9. The image type panel – when the viewer is opened, the selected type of image will be shown.
10. The information panel – in this panel you can view details about each POC. The details include the QP; GOP type; size in the encoded bitstream; PSNR and MSE (if the original video was loaded); number of CUs, PUs and TUs in the frame; The number of POCs in the Frames.mat file, and the decoding time of the whole video.
11. The statistics table – in this table you can view different statistical parameters of the current POC. Statistical information is available for the residual, predicted and reconstructed images (that were decoded). For each image, we calculate the mean and variance of the whole image; and the covariance and correlation of every 2 adjacent pixels (both horizontally and vertically).

In the case of the residuals, the statistics are calculated only inside the Transform Units. This is because residuals can only be found in Tus.

12. Show variance map – this is a toggle button that enables/disables the variance map view. When enabled, the Viewer will show a heat map of the variance of the pixels in the frame. The heat map is calculated by averaging the variance of the pixels in groups of 64x64 squares.



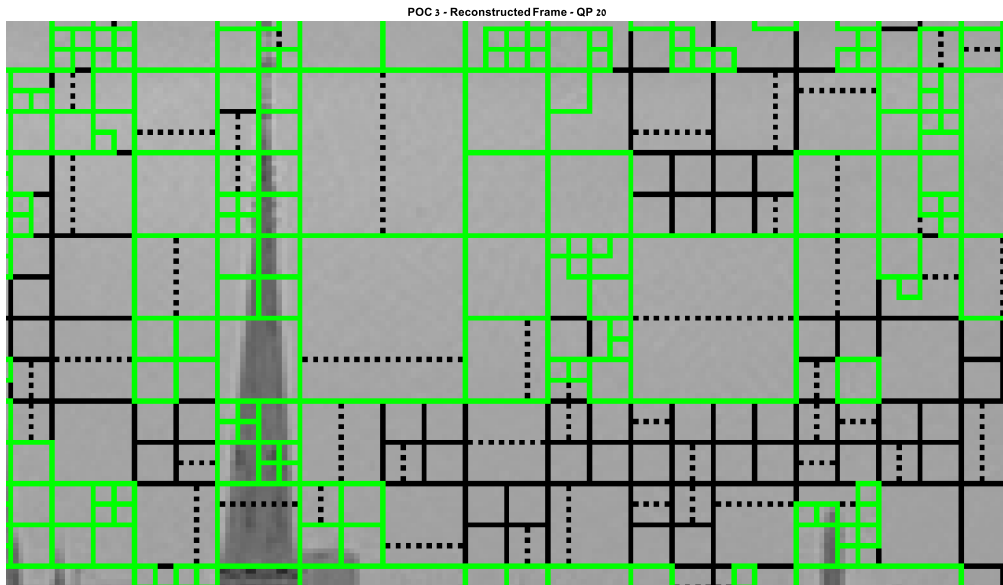
5.4.1. The Viewer



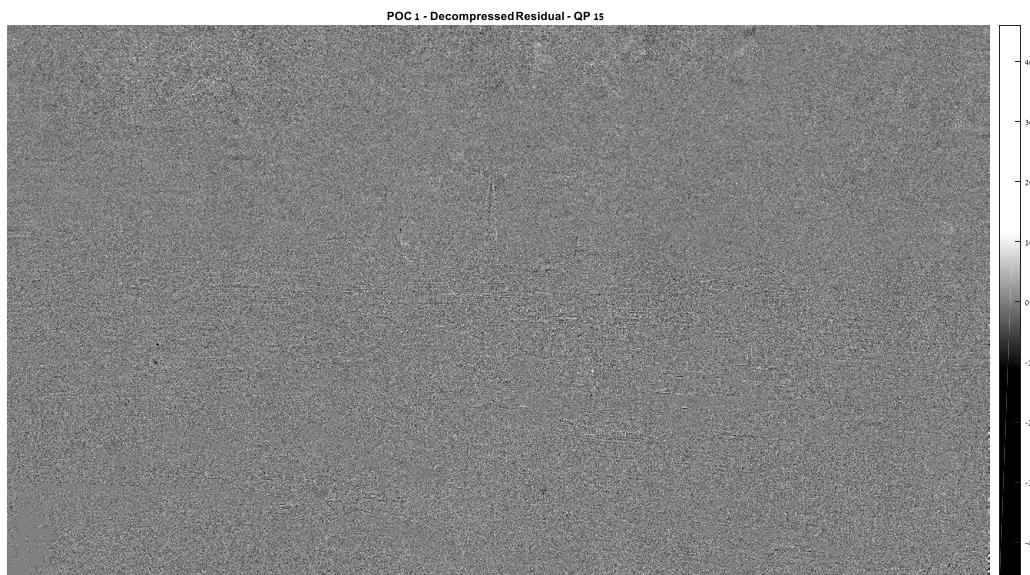
The viewer window is a Matlab figure that shows the current frame with the current partition and image type settings.

Motion vectors are colored and the legend on the top right corner, indicates the POC that the motion vector is directed to. All PUs have motion vectors (again, we only analyze inter predictions). If the motion vector is of size zero, it will be shown as a dot. Note that the motion vectors can be scaled a little.

The CU blocks are bonded by black boxes, and their division into PUs are shown with dotted lines. The TUs that contain nonzero coefficients are bonded by a green box.



For the residual image or the transform coefficients, a color-bar will appear on the side of the image. The value of zero will always be colored in gray.

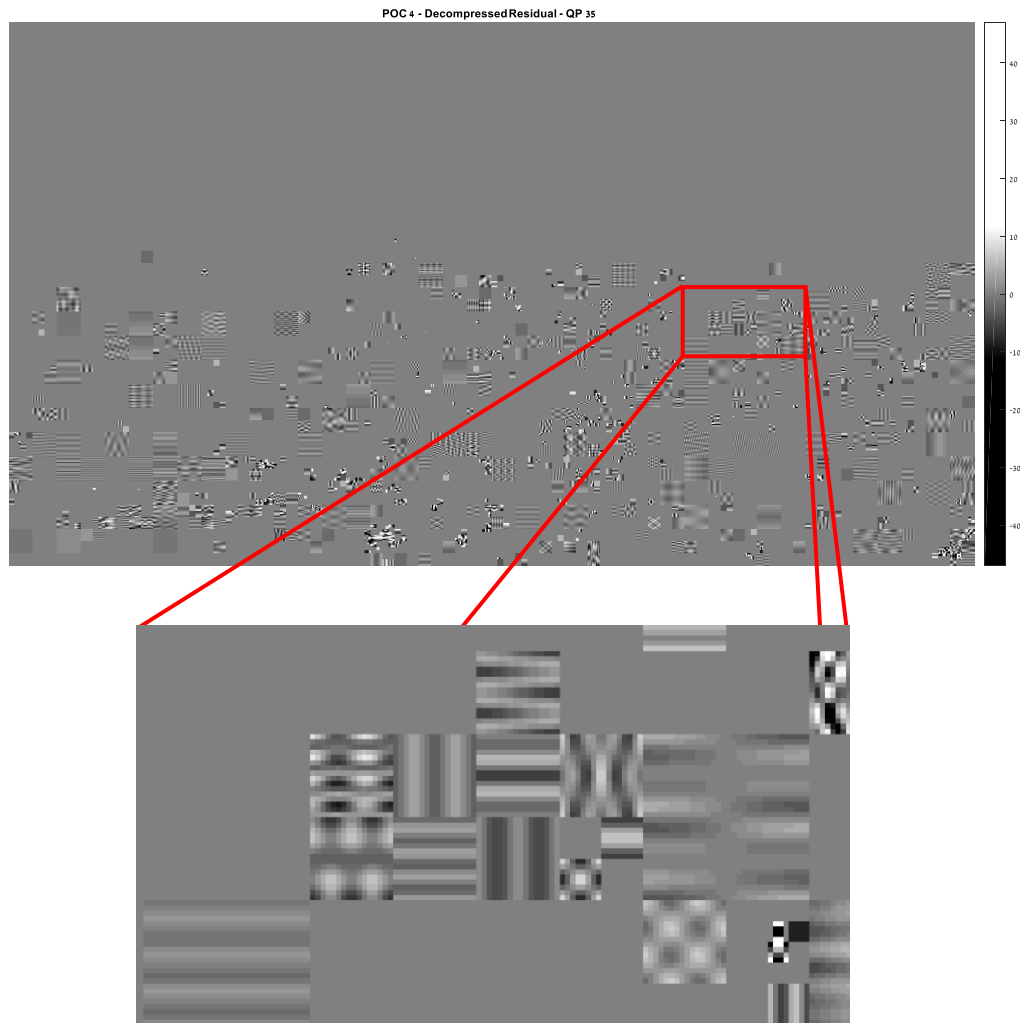


5.4.2. Implementation of the Analyzer

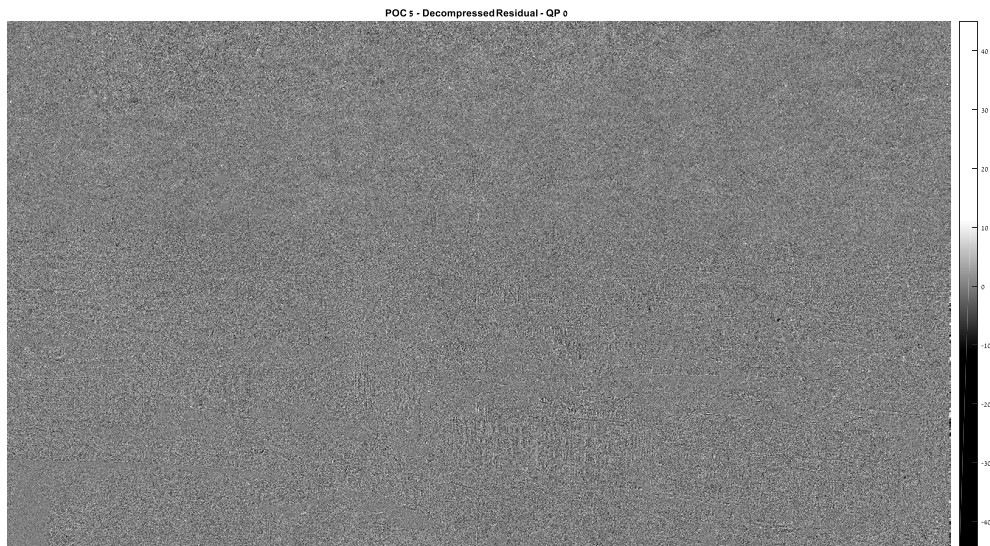
The Analyzer uses FrameObj objects, which are initialized from the given Frames array. Every time a new POC is reached, the Analyzer will build this object for the current POC. This may take time. However, after the creation of this object, it will be saved in the memory, and therefore it will be quicker to access this POC in the future. The Analyzer will keep up to 20 FrameObjs in the memory (this value can be changed in the FramesArray class).

5.5. The Decompressed Residual Images

Our engine runs inside the TAppDecoder. Therefore, the analyzed frames are the decompressed versions of the original frames. This means that the results from the first part of the project can't be directly compared to the results from this part. In the first part the residual images that we analyzed are the direct result of the motion prediction stage. In this part, the analyzed residuals are the result of the inverse-transform operation on the decompressed transform coefficients from the bitstream. Because of this, we can sometimes notice basis images of the DCT transform in the residual image:



The image above shows a residual image taken from the decompressed "Old Town Cross" video with QP = 35. As we decrease the QP and improve the prediction, the residual images look more and more like their original versions.



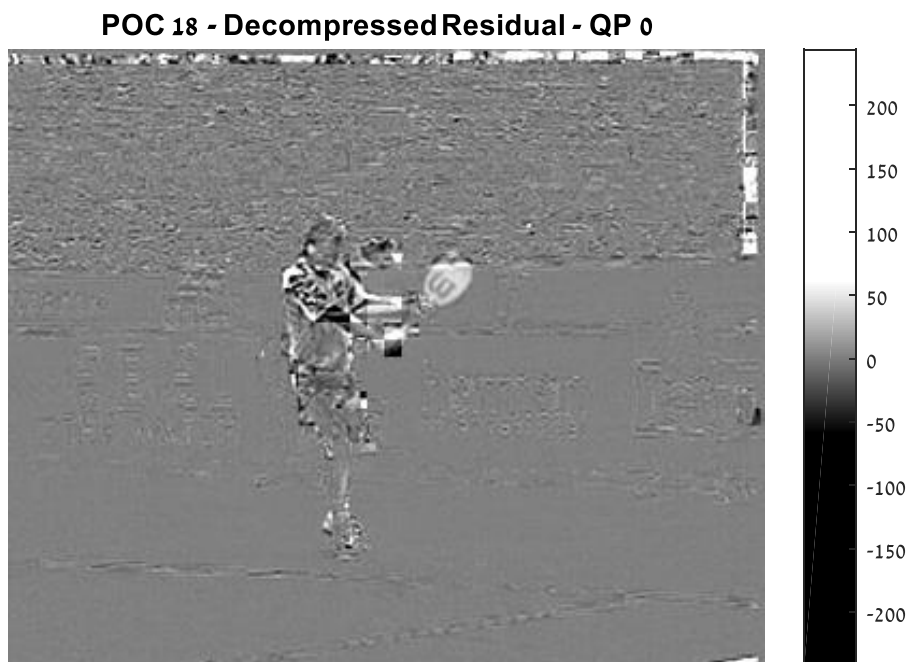
The image above shows a residual image taken from the decompressed “Old Town Cross” video with QP = 0. In this case the residual looks more like white noise.

In our experiments, we will assume that as the QP decreases, the decompressed residual images share similar statistics to the residual images that were produced in the encoder (like in part 1). Therefore, the results for higher QPs should be taken with a grain of salt.

5.6. More examples from the Analyzer

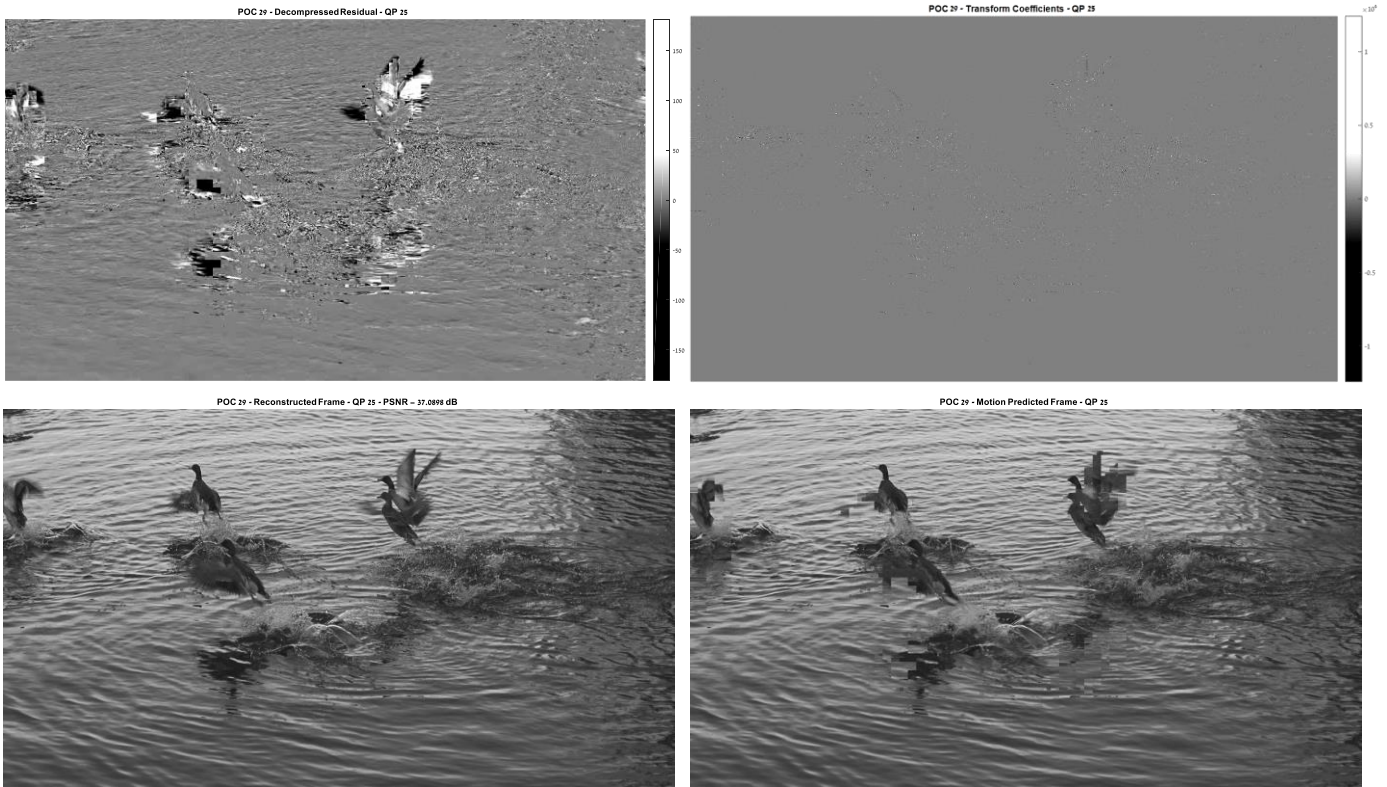
Another type of residuals that we have seen with the analyzer, are the more picture like residuals. These residuals appear in videos that exhibit more motion:

Here is one from the video "Stefan" [8]:



In this video, the player moves a lot while the background remains static. This can clearly be seen in the residual image above.

Another example is with the video "Ducks take off" [9]:



Here we can see the Analyzers output for the images of the transform coefficients, the decompressed residual, the motion prediction, and the reconstructed frame (with a PSNR of 37 dB).

5.7. Testing

In this part, we ran many tests with different parameters on the video of "Old Town Cross". Throughout the tests, we used the same GOP of "BBBP" while only the first frame was an "I" frame.

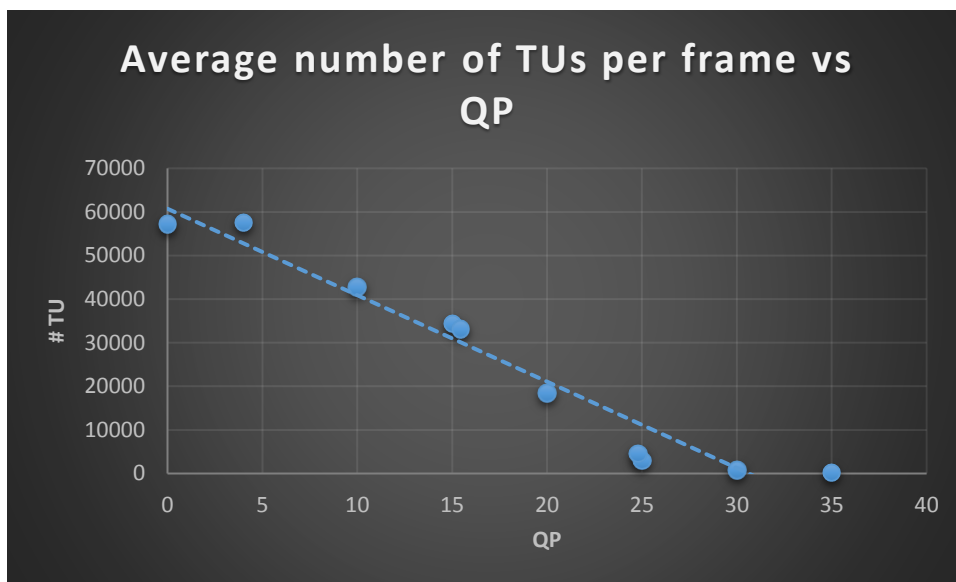
The purpose of the experiments was mainly to test our theory, but also to understand the different parameter settings of the HEVC. We conducted three major tests:

1. *QP* tests – in each test all the frames used the same *QP*. The *QPs* were 35, 30, 25, 20, 15, 10, and 0.
2. *RateControl* and *TargetBitrate* – we ran these tests with different target bitrates while the *RateControl* flag was on. When this flag is enabled, *QP* should be ignored, because the decoder will try to set different *QPs* for different frames in order to reach its target bitrate. The bitrates were 1E5, 1E6, 1E7, 1E8, 1E9.
3. *TransquantBypassEnableFlag* and *CUTransquantBypassFlagForce* tests – we ran some tests with different *QPs* while these flag were set to true. In theory, the *QP* should have no affect since these flags cause the encoder to skip the transformation and quantization stages. The *QPs* were 35, and 0.

5.8. Results

5.8.1. The credibility of our statistical results

After running the tests, we got statistical results about the correlation between every 2 adjacent pixels in the residual images. As previously said, the statistics were taken only from the Transform Units. We discovered that when using lower QP and higher bitrate, the number of the Transform Units increases.

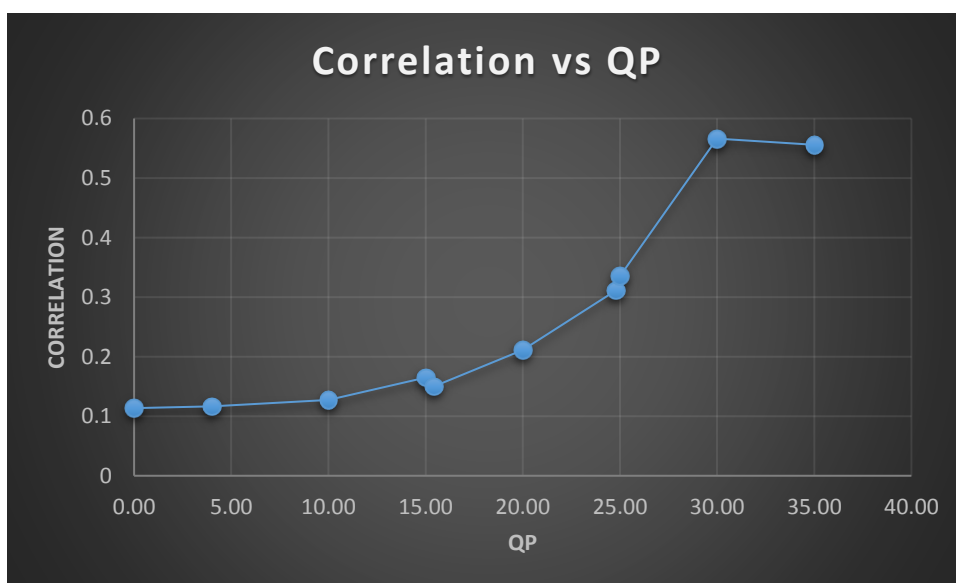


Because of this, the statistical results that we got in cases of higher QPs, are gathered from a much smaller group of data samples, and should not be considered as accurate as the statistical result that is gathered from the cases of lower QP.

5.8.2. Correlation between pixels in the residual image

We will now look at the results from the first and second tests. We are interested to see the effects of the QP (which relates to the quality of the frames) on the correlation between adjacent pixels in the residual image.

Note that the QP here is the average QP of all frames from the video.



As we can see the correlation increases as the QP increases, as expected.

6. Project conclusions

In both parts, we can see similar results. When coding at high bitrates and increasing the quality of the prediction, the residual images show very low correlation.

Accordingly, our empirical results show that transformations should be of a little use at high bit-rate coding, as decorrelation is not needed. This implies that motion-compensation predictions provide residual signals having low correlation and therefore can be directly compressed using scalar quantization.

7. Project summary

In this project, we questioned the need of transform coding on the residual images. We implemented a MATLAB version of a hybrid video coder, where we could easily control the prediction's quality. We extracted data from a standard HEVC decoder, and implemented a MATLAB application that analyzes the data.

We tested our hypothesis on both implementations, and got similar results. The residual's correlation decreases as the prediction's quality increases. Because of that, we concluded that when coding at high bitrates, the transform coding on residuals might not be necessary.

8. Appendix - user instructions

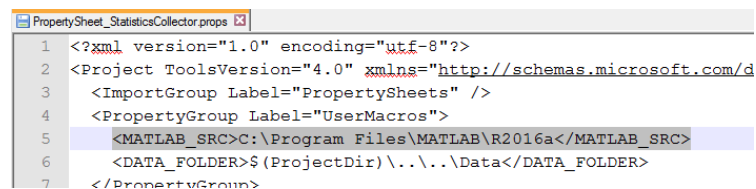
8.1. HEVC's TAppDecoder and TAppEncoder

8.1.1. Compile the applications

All the HEVC's project files are located under the directory HM-16.7.

The TAppDecoder uses the Matlab's engine to collect the statistics. Therefore, it must be linked to the project. Follow these steps to do so:

1. Open "HM-16.7\build\StatisticsCollector\Build_Config_Files\PropertySheet_StatisticsCollector.props" with a text editor, and change the MATLAB_SRC path in line 5:



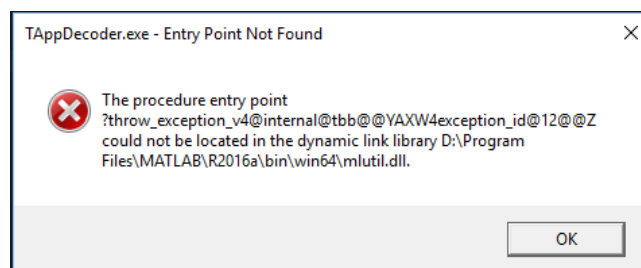
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/d
3 <ImportGroup Label="PropertySheets" />
4 <PropertyGroup Label="UserMacros">
5 <MATLAB_SRC>C:\Program Files\MATLAB\R2016a\</MATLAB_SRC>
6 <DATA_FOLDER>$(ProjectDir)\..\..\Data</DATA_FOLDER>
7 </PropertyGroup>
```

2. Open the project's solution via Visual Studio (2015 and up). The solution is located at "HM-16.7\build\HM_SC_vc2015.sln".
3. Set the TAppDecoder or TAppEncoder as StartUp project, and build the project.
4. To run the application via Visual Studio, you need to change the PATH variable:
 - 4.1. Go to the properties of the TAppDecoder.
 - 4.2. Go to configuration properties -> debugging -> environment.
 - 4.3. Set it to " PATH=%PATH%;\$(MATLAB_SRC)\bin\win64".
 - 4.4. Do the same for the StatisticsCollector project.

8.1.2. Possible problems with the applications

The Matlab Engine can sometimes cause unexpected problems. Here are some that we encountered:

- If the following error appears, copy the tbb.dll (located in \$(MATLAB_SRC)\bin\win64) to the same folder as the TAppDecoder.exe.

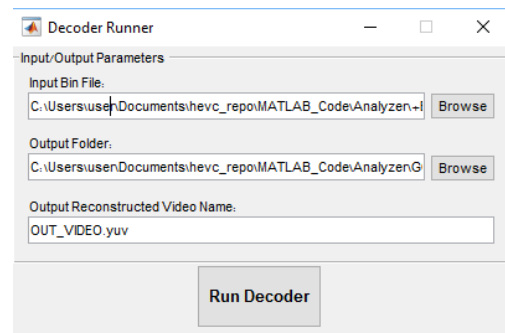
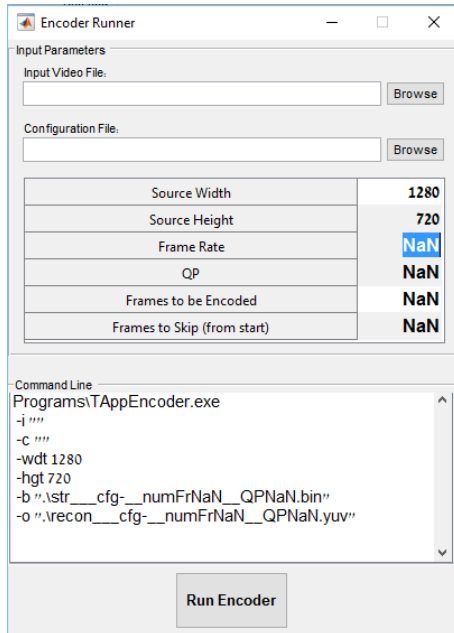


- Other problems should be solved by the settings in PropertySheet_StatisticsCollector.props, but just in case, follow the instructions in this post: https://www.mathworks.com/matlabcentral/answers/100603-how-can-i-compile-a-matlab-engine-application-using-microsoft-visual-studio-9-0-or-10-0#answer_109951.

8.1.3. Run

After compiling the TAppDecoder and TAppEncoder, move the exe files (located in "HM-16.7\bin\vc2013\x64\Release") to the folder "Analyzer\Programs".

We supplied two GUIs that can run these applications:



The Encoder Runner (on the left) runs the TAppEncoder.exe with the given input parameters. The command line on the bottom is automatically generated as you change the parameters. You can edit it, and click on "Run Encoder" to start the encoder. The output will be shown in Matlab's command window.

The Decoder Runner is simpler, browse for the bin file (the encoder's output) and select an output folder for the Frames.mat file and for the reconstructed video.

8.2. The Analyzer

In Matlab, call the function Analyzer from the folder "Analyzer".

There several ways to load a Frames.mat data, to the Analyzer. When initiated, the Analyzer will search the Matlab Workspace for a Frames variable to load. If not found, the Analyzer will search for a Frames.mat file in the Analyzer folder. If not found, the Analyzer will prompt a browse dialog to locate a Frames.mat file.

9. References

- [1] G. J. Sullivan, et al., "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, 2012.
- [2] Qingxiong Huangyuan, et al., "Performance Evaluation of H.265/MPEG-HEVC," 2014.
- [3] "HEVC – What are CTU, CU, CTB, CB, PB, and TB?," 28 October 2012. [Online]. Available: <https://codesequoia.wordpress.com/2012/10/28/hevc-ctu-cu-ctb-cb-pb-and-tb/>.
- [4] Il-Koo Kim, et al., "Block Partitioning Structure in the HEVC Standard," Vols. VOL. 22, NO. 12, no. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, pp. 1967-1706, 2012.
- [5] Xiph.org Video Test Media [derf's collection], "old_town_cross_420_720p50.y4m," [Online]. Available: https://media.xiph.org/video/derf/y4m/old_town_cross_420_720p50.y4m.
- [6] JCT-VC, "High Efficiency Video Coding (HEVC)," [Online]. Available: <https://hevc.hhi.fraunhofer.de/>.
- [7] Frank Bossen, David Flynn, Karl Sharman, Karsten Sühning, "HM Software Manual," 2017.
- [8] Video Trace Library, "stefan_cif.7z," [Online]. Available: http://trace.eas.asu.edu/yuv/stefan/stefan_cif.7z.
- [9] Xiph.org Video Test Media [derf's collection], "ducks_take_off_420_720p50.y4m," [Online]. Available: https://media.xiph.org/video/derf/y4m/ducks_take_off_420_720p50.y4m.
- [10] "Make your first HEVC stream," 4 November 2012. [Online]. Available: <https://codesequoia.wordpress.com/2012/11/04/make-your-first-hevc-stream/>.