Technische Universität München
Chair of Media Technology
Prof. Dr.-Ing. Eckehard Steinbach

# Image and Video Compression Laboratory

Edited by Ingo Bauermann, Yang Peng, Dominik Van Opdenbosch, Kai Cui and Martin Oelsch

## Introduction

In the past decade there have been a variety of interesting developments in multimedia representation and communication. It has become very clear that all aspects of media have and will become more and more digital. As a key development with input from many different disciplines, video compression lies at the core of multimedia signal processing.

There have been significant advances in digital multimedia compression algorithms, which make it possible to deliver high-quality video at relatively low bit rates in today's networks. In this course you will develop, implement, test, and optimize a fully functional hybrid video coder/decoder as it is shown in Figure 1. Even if you do not understand the diagram completely at this point, you will as you proceed through the units of this lab.

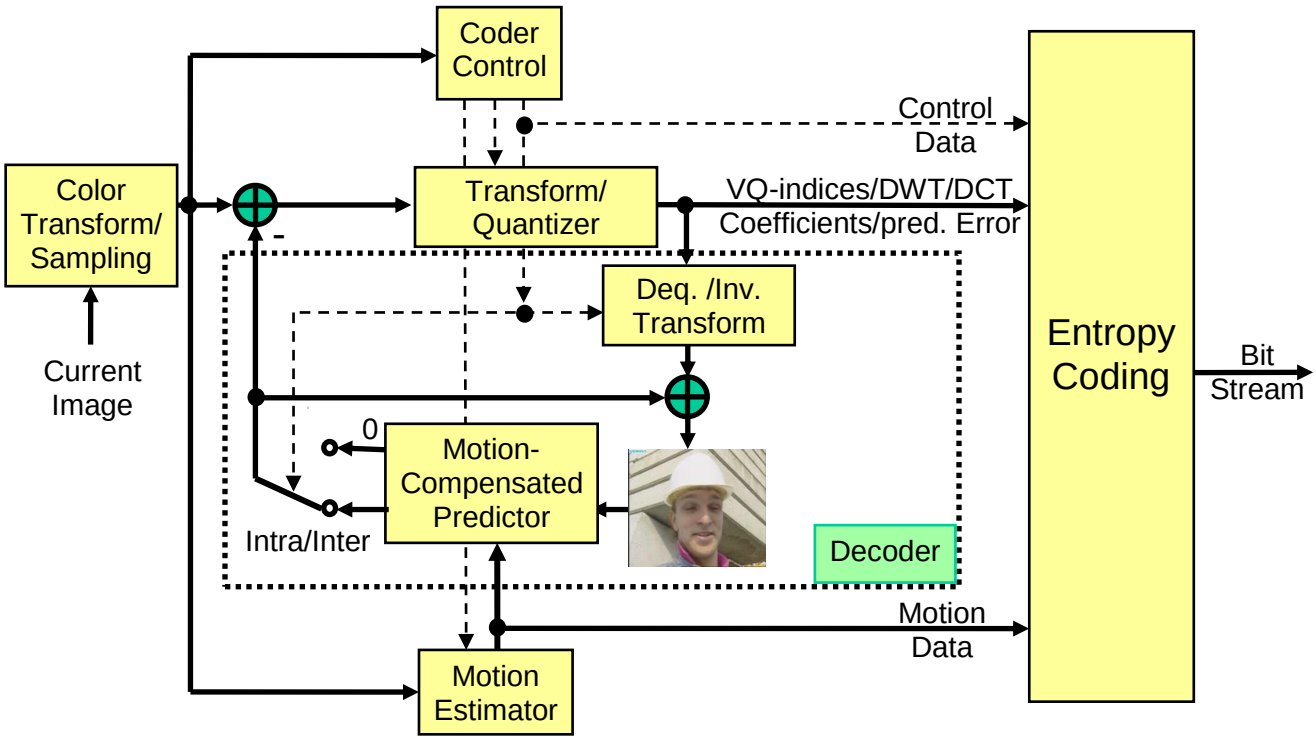



Figure 1: Hybrid video coder (with embedded decoder)

*image source: Prof. Thomas Wiegand TU Berlin/HHI*

The laboratory provides the participants with a detailed overview of the theoretical background and the implementation of a video coding system. From the eighth week of the course on, groups of students can choose from diverse components to develop one unique video coding/decoding system. At the end of the course all codecs will be presented by the participants and compared with respect to compression ratio, image quality, execution speed, and memory consumption.

## Lab Schedule:

1. Fundamentals
    1. Distortion Measure
    2. Filtering and Resampling
    3. Color Transform
    4. **Milestone: Lossy Still Image Codec** **(1st & 2nd week)**

2. Entropy Coding
    1. Statistical Modeling and Entropy
    2. Huffman Coding
    3. **Milestone: Predictive Still Image Codec** **(3rd & 4th week)**

3. Quantization
    1. Uniform Scalar Quantization
    2. Lloyd-Max Scalar Quantization
    3. Vector Quantization
    4. **Milestone: VQ Still Image Codec** **(5th to 6th week)**

4. Transform Coding
    1. Discrete Cosine Transform (DCT)
    2. **Milestone: Transform Still Image Codec** **(7th & 8th week)**

5. Motion Compensation
    1. Temporal Prediction and Blockmatching
    2. **Milestone: Hybrid Video Codec** **(9th & 10th week)**

6. Optimization (Teamwork)
    1. Codec structure
    2. Performance Comparison **(11th to 13th week)**

7. **Final Presentation**

**Note:**

- Participation in the lab only makes sense if you have attended, or at the same time attend the lecture “Image and Video Compression” held by Professor Steinbach.

- Every student has to solve **all** the problems noted with a “**P**” at home **before** the lab session starts. That means, **before** he/she starts working on the corresponding experiments the tutor will check his/her answers!

- The problems noted with “**E**” have to be solved during the lab sessions using Matlab and the code available in your course directory.

- During the final weeks of the lab you will work in teams. Team-working means that **everybody** is responsible for a part of the whole video compression system.

- After the kick-off meeting you will receive an email invitation to join the **Matlab Grader course**. In the course you will find more hints regarding the implementation of the different functions. The Grader will guide you through and help you to find mistakes early!

- Have fun!

**Internet:**

http://www.lmt.ei.tum.de/lehre/image-and-video-compression-laboratory.html

You can find all the download materials in Moodle.

# 1. Fundamentals

The primary goal of image compression is to minimize the number of bits used to represent the original image samples. Due to the fact that human visual perception can often tolerate significant information loss without interfering with the perception of the scene content, there are two main compression schemes.

Lossless compression: Compression without loss of any information. Often used for digital film, medicine and professional graphics to avoid accumulation of errors during several stages of editing. The main principle employed for lossless compression is entropy coding.

Lossy compression: The number of bits used to represent an image can be reduced by allowing a certain level of distortion, i.e. deviation of the reconstructed image from the original. The more distortion is allowed the smaller the compressed representation can be. This approach is widely used for imaging via the Internet, in presentations and so on. The main principle employed for lossy compression is quantization.

In this lab a lossy still image and video encoder and decoder will be developed and implemented.

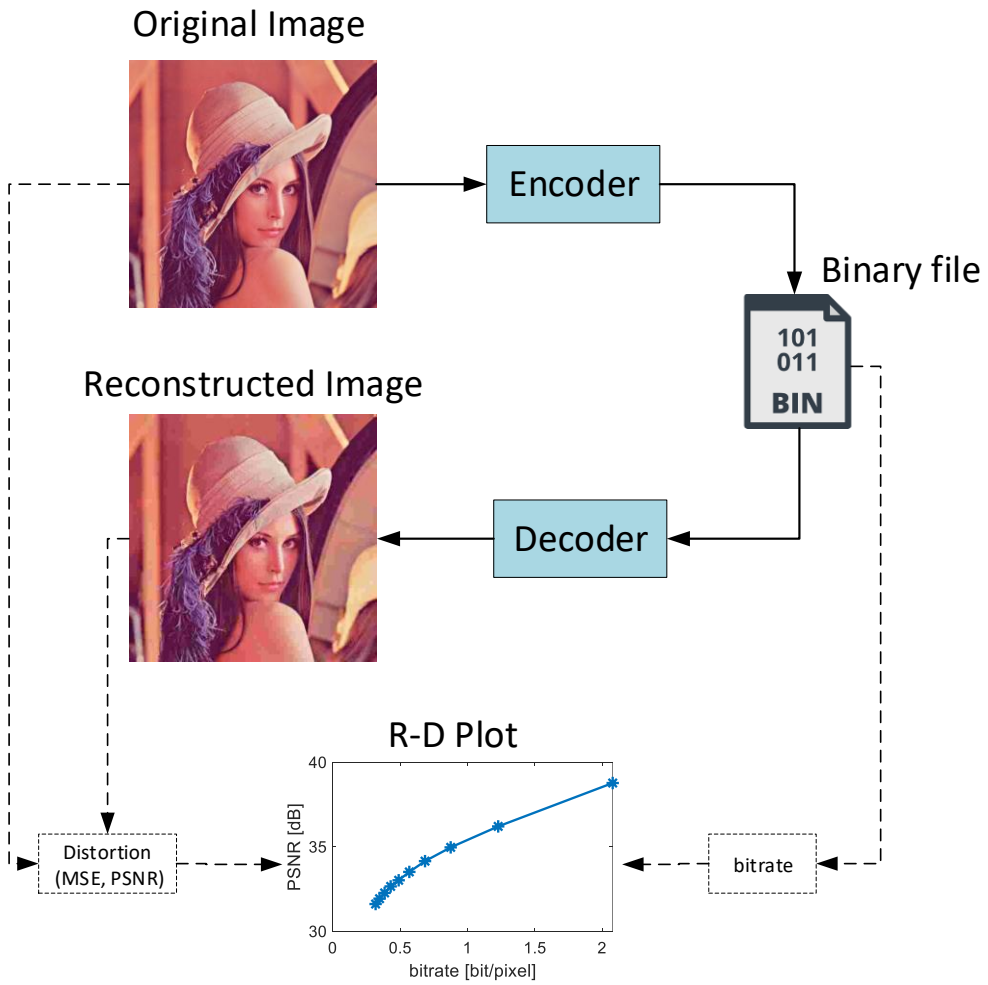



Figure 2: Pipeline of the IVC-Codec prototype

## 1.1 Distortion Measure

To compare and evaluate lossy compression schemes, distortion must be assessed in an appropriate manner. The most common measure of distortion is MSE and PSNR. Reconstructed images with medium to good quality typically have PSNR values of about 30 dB or more.

**P (1-1)** **Determine the PSNR for a reconstructed 512x512 8-bit grayscale image assuming a bit error in the least significant bit of exactly one pixel:**

**PSNR =**

**P (1-2)** **Determine the PSNR for a reconstructed 256x256 RGB color image (8-bit per color plane and pixel) assuming that a white pixel has flipped to a red one: Red-Component: $ff_{hex}$; Green- and Blue Component: $00_{hex}$.**

**PSNR =**

As we want to measure the performance of our coding scheme, a framework has to be set up to conveniently calculate interesting properties of the original image, the produced bit stream, and the decoded image.

**E (1-1)** **Run the 'Warming-up' script of the first chapter in the Matlab Grader to answer the following questions.**

**a. Analyze the Script code and press the 'Run Script' button. Explain briefly which kind of compression is used:**

**Explanation:**

**b. Determine the actual compression ratio (analyze the output in the command window):**

| |
|---|
| **Compression ratio:** |

**c. Determine the bit-rate in bit/pixel before and after compression:**

| |
|---|
| **Bit-rate (original):** |
| **Bit-rate (compressed):** |

**d. Create a function** `calcPSNR()` **to calculate the PSNR between an original image and its reconstruction and** `calcMSE()` **for MSE. Measure the quality of the reconstructed image 'smandril.tif' using the reconstructed image, which you find in the corresponding folder.**

| | |
|---|---|
| **PSNR:** | **MSE:** |

**e. Plot the rate versus distortion point (bit/pixel vs. PSNR) for the current "compression scheme".**

**f. Determine the PSNR values for the test images 'lena.tif' and 'monarch.tif'. Use the reconstructed images in the startup folder and add the result to your rate-distortion plot.**

**PSNR ('lena.tif') =**

**PSNR('monarch.tif') =**

## 1.2 Sampling, Filtering, Resampling

The sampling process used to create a digital image necessarily discards information due to limited spatial resolution of a used CCD-Sensor and finite precision of intensity values for common acquisition devices like digital cameras or camcorders. Most of this lost information is not essential for human visual perception to get a realistic impression of an image. But there are still the same problems as in other signal processing areas (like audio): aliasing. When you try to change the size of a digital image, you have to apply a low-pass filter to your signal before decimation. For a resampled image with a quarter the size of the original image, using a brick wall filter with half of the highest frequency (Nyquist-Criterion) along both axes as the cutoff frequency would be optimal. This theoretically optimal filter, however, can not be implemented. In practice, easier filters are applied to the signal before resampling.

**P (1-3)**

**a. Given is the sampled sine wave shown in Figure 3. Draw the discrete representation of the signal into Figure 4 after sub sampling by a factor of two. No anti-aliasing filter is used in this probem. Please note that the signal is periodic with period $N = 8$.**

**b. Now the signal is interpolated to its original resolution. Draw the digital signals reconstructed from the subsampled version (investigated in problem a.) at full resolution for a nearest neighbor, linear, and a perfect sinc interpolation into Figure 4.**
Note: No detailed calculation is required, only approximation.

**c. Calculate the MSE for the three interpolation techniques from problem b.**

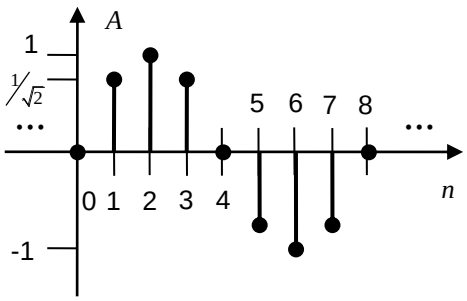



Figure 3: digital sine wave at full resolution

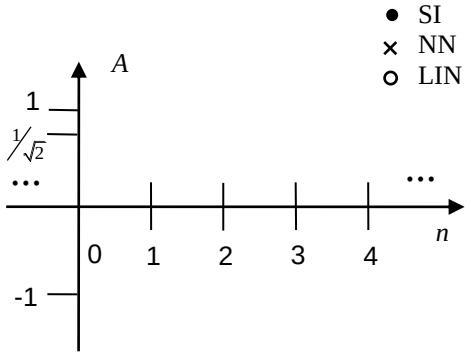
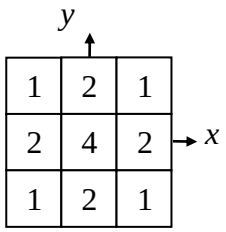



Figure 4: sub sampled representation

**d. Rank the three mentioned interpolation methods by their complexity and by the average deviation they introduce to a signal.**

**E (1-2)**

**a. Implement the subroutine `prefilterlowpass2d()` using the `conv2()` function of Matlab. Use the kernel below ($W_1$) and apply appropriate scaling (normalization of the filter):**




**b. Plot the frequency response of the filter and save the plot to the data/results directory.**

Hint: Use the `fft2()` and `fftshift()` commands described in the Matlab documentation.

**c. Filter the test image 'satpic1.bmp' and display the difference between the filtered and the unfiltered image.**

**d. Subsample the filtered test image by a factor of two in the vertical and horizontal direction using the `downsample()` command and then upsample the small image back to the original size using the `upsample()` command.**

**e. Determine the PSNR for the reconstructed (prefiltered→subsampled→upsampled→post-filtered) test image 'satpic1.bmp' using the filter from problem E(1-2)a ( $W_1$ ). Determine the PSNR for the reconstructed (*not* prefiltered→subsampled→upsampled→post-filtered) test image 'satpic1.bmp'. Compare and try to explain the results. What did you expect?**

**'satpic1.bmp' (prefiltered) PSNR =**

**'satpic1.bmp' (not prefiltered) PSNR =**

**E (1-3)** **Repeat problem E(1-2) for the 2-D FIR filter kernel derived from the matlab commands:**

```
w = fir1(40,0.5);
W2 = w'*w;
```

**Compare your results with E(1-2) and comment on the difference (visual results, PSNR, and complexity). Add the results to your RD-plot.**

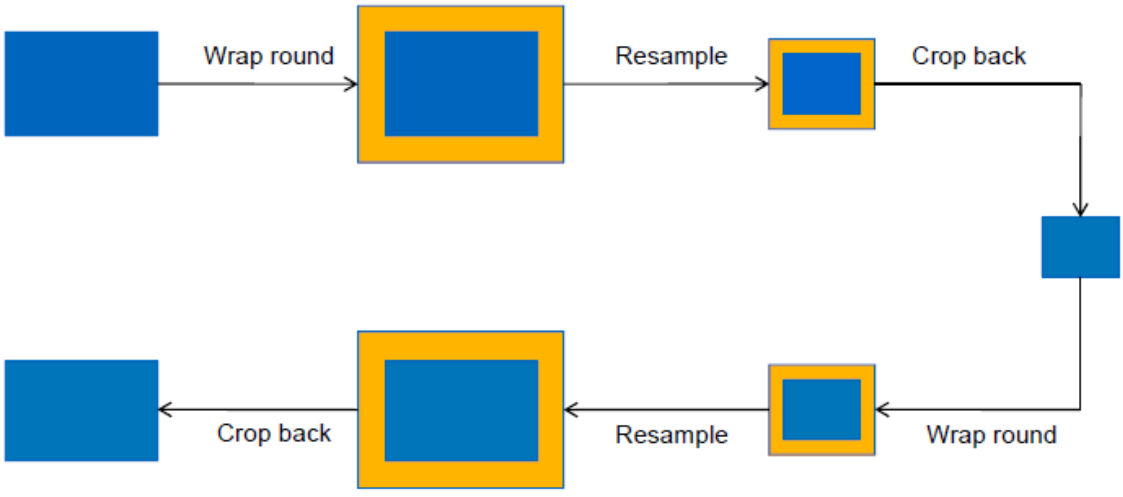
**'satpic1.bmp' (prefiltered) PSNR =**

**'satpic1.bmp' (not prefiltered) PSNR =**

**Comments:**

**E (1-4)** **a. Implement a subroutine to perform a sub sampling of a factor of two in horizontal and vertical direction. Make sure that there are no border effects after reconstruction.**

Hint: 1. The `resample()` command already performs filtering. The standard filter length is 21 taps. That causes border and ringing effects. Change the filter length to three on each side of the sample (use MATLAB-Help!).
2. Additionally, you can wrap around, filter, and crop back the image to original size to avoid overshooting of the filter output at edges. Mirroring the image at the edges (4 pixels each side) before filtering is one of the best approaches to avoid border effects.



**'sail.tif' reconstructed:**

**'lena.tif' reconstructed:**

## 1.3 Color Transform

An interesting example of irrelevance occurs in color imagery, where the human viewer is substantially less sensitive to rapid changes in the hue and saturation properties of the image than to intensity changes. This property is used to represent a sample with fewer bits without changing the perception of the image. This is done by converting an RGB-image to a color-space called 'YCbCr' (luminance, blue chrominance, and red chrominance) to separate luminance and color information. Then the chrominance (color) signals can be down sampled (bit-rate reduction) by some appropriate factor (2, 4, ...) without perceptible loss of quality.

**P (1-4)** **Calculate the compression ratio if the two chrominance images calculated with the irreversible color transform are subsampled by a factor of two horizontally and vertically. The luminance image remains unchanged.**

**Compression ratio =**

**E (1-5)** **Implement the subroutines** `ictRGB2YCbCr()` **and** `ictYCbCr2RGB()` **which perform the ICT. The equations for the ICT are listed up in the following:**

$$Y = 0.299R + 0.587G + 0.114B \qquad R = Y + 1.402Cr$$
$$Cb = -0.169R - 0.331G + 0.5B \qquad G = Y - 0.344Cb - 0.714Cr$$
$$Cr = 0.5R - 0.419G - 0.081B \qquad B = Y + 1.772Cb$$

**E (1-6)** **a. Transform the input image ('sail.tif') using the ICT and subsample both chrominance planes horizontally and vertically by a factor of 2. Round all resulting values to integer values. Use the padding approach from E (1-4)!**

Hint: Again, use the command `resample()` and do not try to process and store differently sized components using one single matrix. Process the data component by component.

**b. Reconstruct the image and describe the difference to the original.**

**Description:**

**c. Calculate the bit rate and determine the PSNR for the implemented compression scheme assuming that 8 bit per component are used to represent the image after the color transform. Use the test image 'sail.tif'.**

| **Bit-rate:** | **PSNR:** |
|---|---|

## 1.4 Rate-Distortion Performance

To compare your results and to find out the most powerful combination of techniques, it is convenient to have an overview of the performance of the implemented modules.

**E (1-7)**

**Modify your program to enable visualization of D-R plots for several combinations of the techniques you have developed so far. Try several input images. Which combination performs best in the rate-distortion sense?**

Hint: Your plot should look similar to Figure 5. For future experiments keep in mind that a reasonable image quality is about 25 dB and higher. Always try to span the range from 25 to 40dB if you are analyzing operational rate distortion performance. However, always have a look at the result!

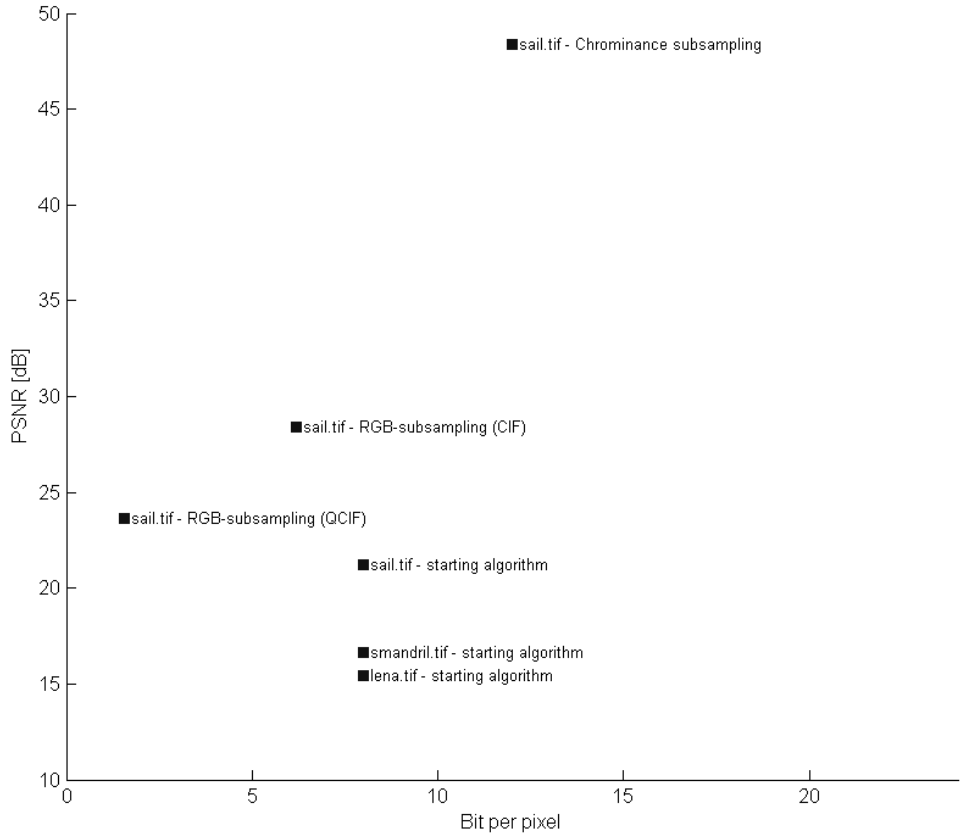



Figure 5: Rate-Distortion performance of selected experiments of the first chapter

# 2. Entropy Coding

In the first chapter you have evaluated and developed basic techniques and tools typically used before and after compressing images. The milestone for the second week will be a lossy predictive still image codec. You will implement a spatial prediction scheme to take advantage of the statistical properties of the prediction error. You will gather statistics from a set of test images and will use these statistics to efficiently compute appropriate Huffman code tables.

## 2.1 Statistical Modeling and Entropy

In the startup folder you will find a test image set. These images are:

data/images/lena.tif
data/images/smandril.tif
data/images/sail.tif

This set contains three images for the next exercises, but later in this course you may want to use your own set of test images. To save time on testing your implementations use the image

data/images/lena_small.tif

which is a subsampled version of 'lena.tif'. All these images are color images in the RGB format. For the following problems assume that every color-plane is treated as a grayscale image and is processed separately (8-bit per color component).

### 2.1.1 Memoryless coding of images

**P (2-1)** **Determine the number of counters (histogram bins) that have to be used to gather statistics from the test images assuming that every possible color intensity value in the three test images is stored using 8 bit. Prepare Matlab code to count the frequency of intensity values.**

**Number of counters:**

**Matlab code:**

**E (2-1)**

**a. Implement the subroutine `stats_marg()` that calculates the approximation of the PMF of an image. Use a three dimensional Matrix (the color image!) as input parameter to your function. Calculate the PMF's of every test image separately.**

Hint: 1. You can read several image formats using 'imread'. The returned matrix contains 8-bit integer values, e.g. `imread('data/images/lena.tif')`. Convert the image to a floating point representation using `double()` before using it.
2. Use the command `hist()` to calculate 1D histograms.
3. Do not count color components separately. Just treat every color intensity value as a sample.

**b. These PMF's will now be used to find the lower bound on the average code word length for each image. Determine the minimum average code word length for each test image. Name the function you have to implement `calc_entropy()`.**

| | |
|---|---|
| **'lena.tif' :** | **H =** |
| **'sail.tif':** | **H =** |
| **'smandril.tif' :** | **H =** |

**c. Now find the minimum average code word length, if all three test images are to be encoded with one common code table. For this create a function `min_code_length()`, which computes the minimum average code word length using the common code table and the PMF of each of the images. I.e. statistics are gathered from a merged version of the images.**

| | |
|---|---|
| **'lena.tif' :** | **H =** |
| **'sail.tif':** | **H =** |
| **'smandril.tif' :** | **H =** |

**d. How much worse or better in bit/pixel does the combined code table perform in comparison to the individual code table in part b.?**

| | |
|---|---|
| **'lena.tif' :** | **$\Delta$H =** |
| **'sail.tif':** | **$\Delta$H =** |
| **'smandril.tif' :** | **$\Delta$H =** |

## 2.1.2 Lossless joint encoding of pixel pairs

**P (2-2)**

**a. Determine the number of counters that have to be used to establish the joint pmf of vertically adjacent pixels (pixel pairs).**

**Counters for pixel pairs:**

**b. Determine the number of counters that have to be used to establish the joint pmf of blocks of 4 by 4 pixels. Determine the formula for calculating the number of counters for blocks of *nxn* pixels.**

**Counters for 4 by 4 pixel blocks:**

**Counters for *nxn* pixel blocks:**

**c. Consider Figures 6 to 8 below. Rank the 8bit grayscale images by their marginal and their joint entropy (pixel pairs of vertically adjacent pixels). State why and in which case joint encoding would result in a lower bit-rate.**

Hint: In Figure 7 we have all gray level values equally distributed from the left (0) to the right (255).

**Explanation:**

Figure 6: 'lena.tif'

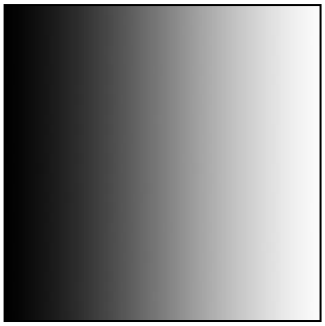


Figure 7: 'ramp.bmp'

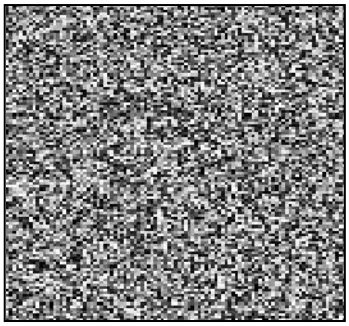


Figure 8: 'randomdot.tif'

**E (2-2)**

**a. Consider joint encoding of more than one pixel at a time by forming pixel pairs as shown in Figure 9. Implement the sub routine `stats_joint()` which takes a color image as input and gives a joint PMF as the output. For image 'lena.tif', plot the joint histogram of every two horizontally adjacent pixels.**

Hint: The Matlab command `mesh()` produces a 3D mesh from two-dimensional matrices.

**b. Find the minimum average code word length if a variable length code is used to represent pairs of horizontally adjacent pixels for 'lena.tif'.**

**'lena.tif':** $H_{joint}$ **=**

**c. Compare your results with those from problem E (2-1).**

**Comparison:**

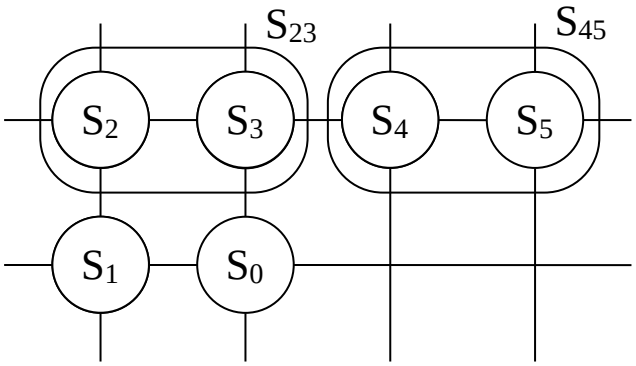



Figure 9: Forming pixel pairs for joint entropy coding.

## 2.1.3 Lossless conditional coding assuming a Markov-1 source

**P (2-3)**

**Again consider Figures 6 to 8. This time rank the images by their conditional entropy. Conditional probabilities are calculated given the pixel to the left. For instance, in Figure 9 pixel $S_0$ will be conditioned on pixel $S_1$. Compare the result to P (2-2) c.**

**Comparison:**

**E (2-3)**

**a. Consider conditional encoding of sample $S_4$ in Figure 9 given its neighbor to the left $S_3$. Plot the conditional histogram of two horizontally adjacent pixels for the image 'lena.tif' (notice that this time the pixel pairs are overlapping). For convenience the most left pixel in each row can be omitted. Name this sub routine** `stats_cond()`**.**

**b. Determine the minimum average code word length for 'lena.tif' using conditional encoding and the statistics from problem a.**

**'lena.tif':** $H_{cond}$ **=**

**c. Compare your results with those from problem E(2-1) and E(2-2).**

**Comparison:**

## 2.1.4 Lossless predictive coding of images

An obvious property of natural images is that a region around one pixel generally speaking doesn't change too much. Adjacent pixels are similar and therefore assuming a Markov-Source of higher order could be an appropriate model. The computational complexity for conditional Huffman tables would be very large. We also would need large memory for storing the conditional code. A more practical technique than conditional coding is to derive a prediction scheme which produces a prediction error with greatly reduced statistical dependencies between adjacent samples.

**P (2-4)**

**For the given autocorrelation matrix calculate the minimum variance linear predictor of order 1.**

$$R = \begin{bmatrix} 0.3040 & 0.3033 \\ 0.3033 & 0.3040 \end{bmatrix}$$

**E (2-4)**

**a. Only pixel $S_1$ is used as a predictor ($a_1$ = 1) for pixel $S_0$ (see Figure 9). Find the minimum average code word length for encoding the prediction error for the image 'lena.tif'. Do <u>not</u> convert to YCbCr and use the same predictor coefficient for all RGB channels.**

**Entropy of residual error: $H_{err}$ =**

**b. Find the minimum average code word length for encoding the prediction error for image 'lena.tif' using the minimum-entropy predictor from the lab introduction slides. Convert the image to YCbCr domain prior to prediction. Note that now we have different predictor coefficients for each channel.**

**Entropy of residual error: $H_{err}$ =**

**c. Discuss the performance of the code books in E(2-4) and E(2-3) with respect to compression ratio and computational complexity.**

**Comparison:**

## 2.2 Huffman Coding

To actually encode the images you have to set up coding tables which are uniquely decodable. These prefix codes for the test images can be derived from the statistics of the images themselves with the disadvantage of transmission overhead. To avoid this overhead the encoder and decoder in this course both will use the same code tables pre-computed in 'config.m'. To test your algorithms and to save time use the test image 'lena_small.tif'.

**P (2-5)**

**Develop an algorithm in words to compute the tree structured Huffman code table for a given alphabet and a corresponding set of probabilities. Assume the alphabet and the associated probabilities as given. Give an algorithm for decoding a source encoded using your Huffman code.**

**Computing Huffman code:**

**Decoding algorithm:**

**E (2-5)**

**Calculate the codewords and codeword lengths of the Huffman Code of the residual error produced by using the minimum-entropy predictor from problem E(2-4)b. Make sure that every *possible* source symbol is covered by the code even if it does not occur in the test set. Plot the codelengths and show that the code is a prefix code. Use the function `buildHuffman()` found in your analysis folder. This function calculates codewords and codesizes (in bits) of a variable length, non adaptive Huffman code given a probability mass function. Use 'lena_small.tif' as input for the predictor.**

| |
|---|
| **Number of codewords:** |
| **Max. codewordlength:** |
| **Min. codewordlength:** |

## 2.3 Milestone: Predictive Still Image Codec

At the end of the second chapter you should be able to finish a fully functional image encoder and decoder. Using prediction and Huffman Coding of the residual error you should achieve quite high compression ratios for coding of still images.

**E (2-6)**

**a. Implement the functionality to actually encode and decode an image using prediction and Huffman coding of the prediction error. Apply an ICT to the image and subsample the chrominance images by a factor of two horizontally and vertically. Use the Huffman code from problem E(2-5) (obtained from 'lena_small.tif') for encoding.**

Hints: 1. The functions `enc_huffman_new()` and `dec_huffman_new()` in the encoder and decoder directories are implementations of a Huffman encoder and decoder, respectively. The calling parameters and return variables can be determined from the corresponding file headers. These functions need the output of the `buildHuffman()` function.
2. The file 'testhuffman.m' is an example implementation of the Huffman coding scheme.

**b. Determine the bit rate, compression ratio and PSNR for 'lena.tif' using your implementation from problem E(2-6).**

| | |
|---|---|
| **Bit rate:** | **bit/pixel** |
| **Compression ratio:** | |
| **PSNR:** | **dB** |

**E (2-7)**

**Extend the tool from E(1-7) to have an overview over the progress of your codec. Add the rate-distortion points for your predictive still image codec.**

# 3. Quantization

In Chapter 2 besides developing lossless compression schemes you have already started to work on a lossy compression scheme. The part of the codec where information got lost was the sub sampling process, applied to the input data after the color transformation. We have exploited different sensitivity to luminance and color information of the human visual perception. In this chapter we will quantize the intensity values to further reduce the amount of data used to represent an image. The milestone for the end of the 3rd Chapter is a still image codec based on vector quantization (VQ).

## 3.1 Uniform Scalar Quantizer

**P (3-1)**

The images we are working with are already a quantized representation of the real world. A resolution of eight bit per color is sufficient for humans to perceive a 'continuous' range of colors. Reducing the number of bits per pixel using quantization introduces additional distortion but reduces the required bit-rate.

**a. Draw the D-R pairs for the three quantizers in Figure 12 and the signal shown in Figure 10 into Figure 11.**

**b. Calculate the variance of the signal and determine the distortion for the case that nothing is stored at all. Add this value to the D-R curve.**

**c. Determine the minimum average code word length for lossless compression of this sequence (marginal entropy) and plot the corresponding D-R pair into Figure 11.**

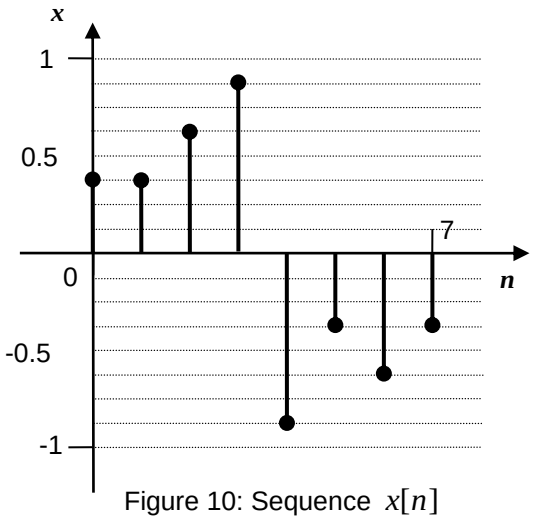



Figure 10: Sequence $x[n]$

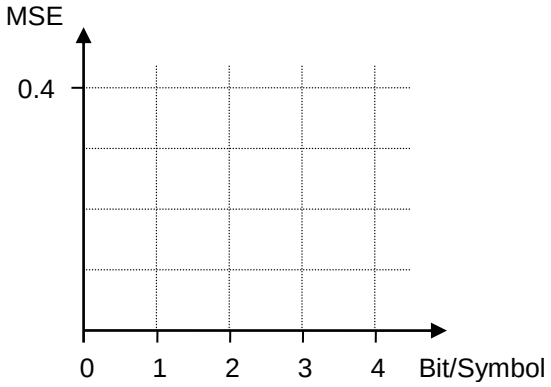



Figure 11: D-R function

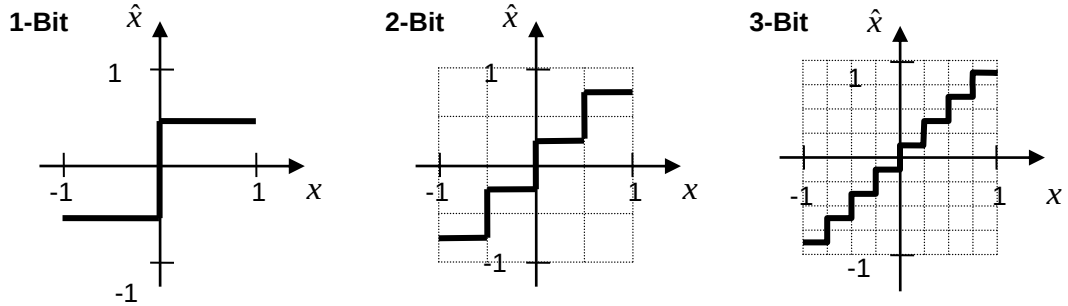



Figure 12: Mid-rise Uniform Quantizers

**MSE(R):**

**E (3-1)**

**a. Implement a function `UniQuant()` that performs a mid-rise uniform quantization from the value range $0 \leq Intensities < 1$ to $0 \leq Indices < 2^M$ where M is the number of bits used.**

**b. Implement a function `InvUniQuant()` that performs the inverse operation.**

**c. Fill in the following table:**

| Image | M/color plane | PSNR(dB) |
|---|---|---|
| lena_small.tif | 1 | 16.1675 |
| lena_small.tif | 2 | 22.9543 |
| lena_small.tif | 3 | 28.4172 |
| lena_small.tif | 5 | 40.6655 |
| lena_small.tif | 7 | 51.1319 |
| lena.tif | 3 | 28.6464 |
| lena.tif | 5 | 40.6938 |

**d. Draw the Rate-Distortion curve (PSNR over Rate) for your uniform quantizer and for the test image 'lena_small.tif' and 'lena.tif' into Figure 13 and determine the slope of the curve:**

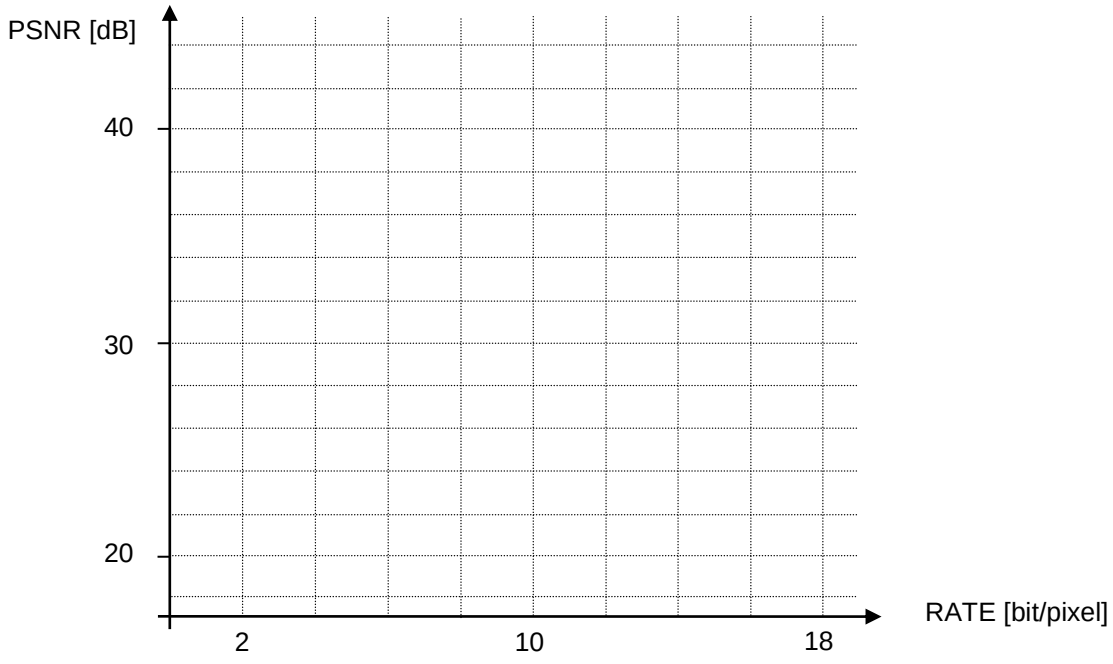



Figure 13: R-D Curve

## 3.2 Lloyd-Max Scalar Quantizer

The Lloyd-Max quantizer is a scalar quantizer which can be seen as a special case of a vector quantizer (VQ) designed with the LBG algorithm (refer to the lecture for details!). For a given distortion criterion, encoding rule, and training set, given an initial codebook, the LBG algorithm iteratively optimizes the decoder while fixing the encoder (changing representatives while keeping the quantization cell boundaries fixed) and then the encoder while fixing the decoder (changing the quantization cell boundaries while keeping the representative fixed). This iterative procedure guarantees non-increasing distortion and therefore convergence. The resulting codebook is then indexed with a **fixed-length** code. In practice we will use mean squared error (MSE) as the distortion criterion, and a nearest neighbor mapping as the encoding rule. Some notation has to be defined:

$C_N = \{\hat{\boldsymbol{x}}_0, \hat{\boldsymbol{x}}_1, .., \hat{\boldsymbol{x}}_q, .., \hat{\boldsymbol{x}}_{M-1}\}$ is the codebook $C$ at iteration step $N$ where $\hat{\boldsymbol{x}}_q$ denotes a representative vector and $q$ is the codeword index.

$Q(\boldsymbol{x}) = q$ is the mapping from input vector $\boldsymbol{x}$ to a codeword index $q$.

$\rho(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \| \boldsymbol{x} - \hat{\boldsymbol{x}} \|^2$ defines the distortion between vectors $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ : MSE

$J(\boldsymbol{x}, \hat{\boldsymbol{x}}, l) = \rho(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \lambda l$ Lagrangian cost function denoting the cost of mapping vector $\boldsymbol{x}$ to a reproduction codeword $\hat{\boldsymbol{x}}$ , associated with a code word of length $l$ , weighted with the Lagrangian multiplier $\lambda$ .

$T = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_{\|T\|-1}\}$ is the training set of size $\| T \|$vectors. This test set normally consists of pixels or pixel pairs

$B_q^N = \{\boldsymbol{x} : J(\boldsymbol{x}, \hat{\boldsymbol{x}}_{\boldsymbol{q}}) \le J(\boldsymbol{x}, \hat{\boldsymbol{x}}_{\boldsymbol{k}}); \forall k \ne q\}$ is a partition obtained by using least Lagrangian costs at iteration $N$.

$T'_N = Q(T, C_N, B_q^N)$ denotes the quantized training set $T'_N$ . Using partition $B_q$ and the codebook $C$ , the trainingset $T$ is quantized to $T'_N$ at iteration $N$.

$d_N(T, T'_N) = E\{\rho(\boldsymbol{x}, \hat{\boldsymbol{x}})\}$ is the average distortion between $T$ and $T'$

For an efficient implementation and easy extension to the entropy-constrained vector quantizer (ECVQ) we will implement a modified LBG algorithm. In general, the LBG algorithm uses a **variable-length** code (VLC) to index the representative vectors and minimizes the global cost function $J = d + \lambda H(\hat{\boldsymbol{x}})$.

Given the trainings set $T$ and some fixed trade-off between rate and distortion (a value for $\lambda$), the modified LBG algorithm for ECVQ works as follows:

1. **Initialization**
   Setup an initial codebook $C_1$ and determine $J_1 = d(T, T') + \lambda H(\hat{\boldsymbol{x}})$. $H(\hat{\boldsymbol{x}})$ is calculated using the histogram of $T'$.
2. **Optimize the encoder**
   Given the codebook $C_N = \{\hat{\boldsymbol{x}}_q\}$, find the optimal partition $B_q^N$ into quantization cells. In the entropy constrained case update the channel codeword lengths using $l_q = -\log_2(p_q)$.
3. **Update representative vectors**
   Find the centroid in each quantization cell $\hat{\boldsymbol{x}}_q = E\{\boldsymbol{x} \mid \boldsymbol{x} \in B_q^N\}$, to form the optimal reproduction alphabet $C_{N+1}$.
4. **Update Lagrangian cost function**
   Determine $J_{N+1} = d(T, T') + \lambda H(\hat{\boldsymbol{x}})$.
   Stop if $\frac{|J_N - J_{N+1}|}{J_{N+1}} < \varepsilon$. Otherwise increment $N$, go to step two.

**P (3-2)**

**Figures 14, 15, 16 and 17 show histograms of a scalar quantizer output for an 8-bit input image. The $\hat{x}_i$ in figures show the value of the quantizer representative values.**

**a. Which of the histograms in Figures 14-17 is a possible outcome of a 2-bit uniform quantizer? Why?**

**2-bit uniform quantizer output:** 17

**Explanation:** 4 representative values, uniformly distributed

**b. Which of the histograms is a possible outcome of a 3-bit Lloyd-Max quantizer? Why?**

**3-bit Lloyd-Max quantizer output:** 16

**Explanation:** 8 representative values

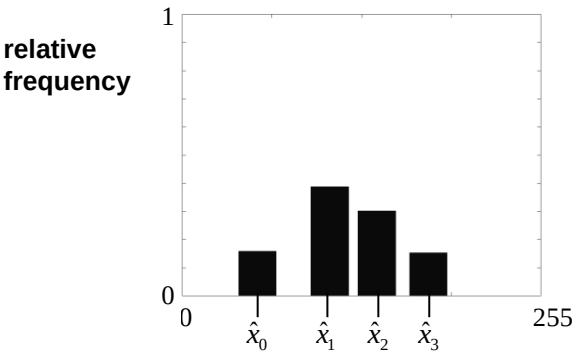



Figure 14: Quantizer outcome 1

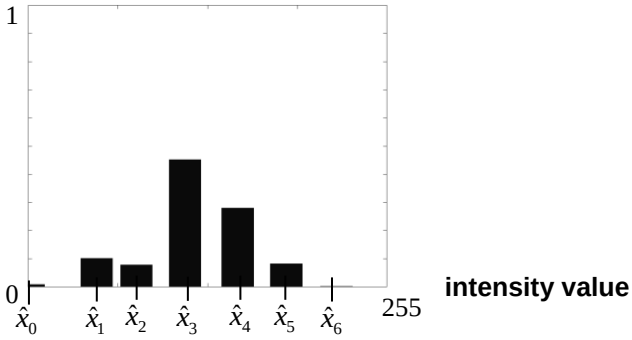



Figure 15: Quantizer outcome 2

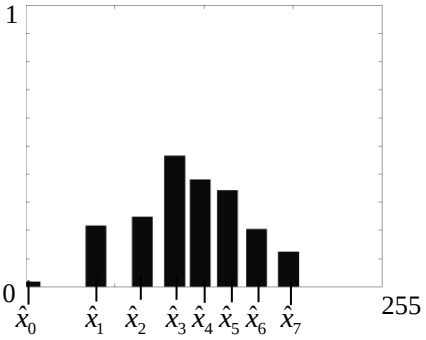



Figure 16: Quantizer outcome 3

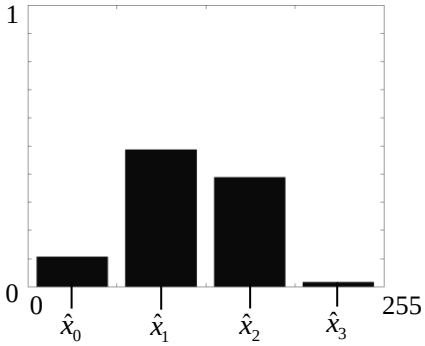



Figure 17: Quantizer outcome 4

**P (3-3)** **Figure 18 shows the distortion-rate function for a memoryless gaussian random variable. This random variable is to be quantized.**

**a. Determine the variance of the source.**

$\sigma^2 =$ 1.414 bit/sample = 0, MSE =variance

**b. Suppose we initialize the quantizer design with uniform quantization. This leads to the distortion-rate pair A. Starting from this point, in which direction will the distortion-rate pair shift during the iterations of a 2-bit Lloyd-Max quantizer design? Why?**

**Direction:** vertical downwards until rate-distortion curve

**Explanation:** Lloyd-Max quantizer, fixed code length

**c. Given the outcome of the 2-bit Lloyd-Max quantizer from problem b, in which direction would the distortion-rate pair shift after VLC encoding of the indices? Why?**

**Direction:** left

**Explanation:** variable length codeword can reduce the dependency of indexes
distortion no change during index entropy coding

**d. Which distortion-rate pairs (B to E) can be reached by a scalar quantizer (assume high-rate limits)?**

**Points that can be reached:** B, E

**Explanation:** 4.35dB to rate-distortion curve (theoretical bound), at high rate
D achievable through vector quantizer
C below rate0distortion curve

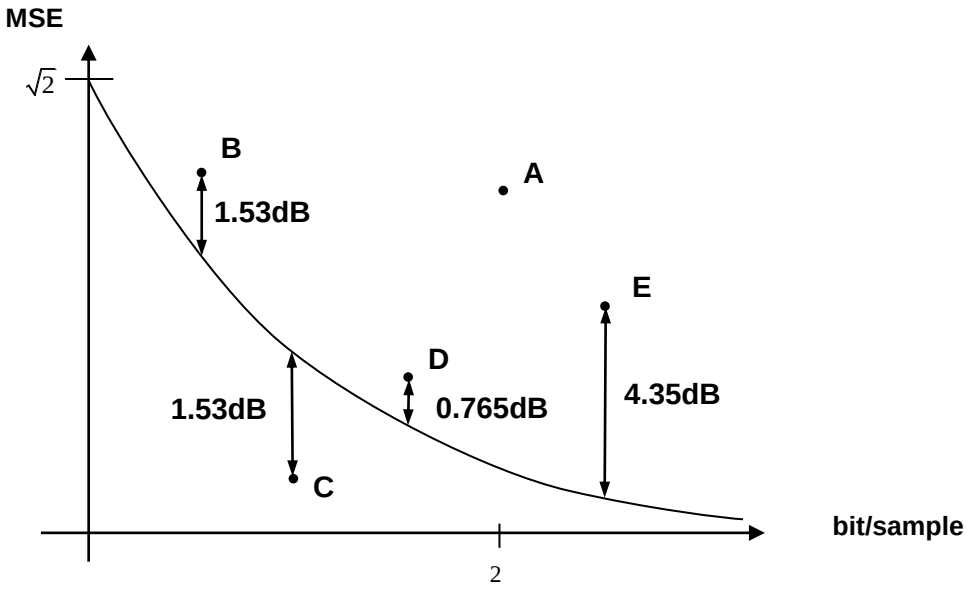



Figure 18: Rate-Distortion function of a memoryless gaussian random variable

**E (3-2)**

**The function `LloydMax3()` is to be implemented. It performs a Lloyd-Max optimization of quantization intervals for a three bit *fixed length code* and an appropriate $\varepsilon$ using training data.**

Hints: 1. Choose $\varepsilon = 0.001$ if you work in the intensity value range from 0..255.

2. Setting $\lambda = 0$ in the LBG-algorithm results in the unconstrained case where the representative vectors are encoded using a fixed-length code and by minimizing average distortion $d$ . Input vectors $\boldsymbol{x}$ consist of only one element (scalar quantizer). If you encounter empty cells during the optimization steps, split up the most populated cell to achieve minimum distortion. You may want to use a code book structure like the following (nx3 matrix – where n is the number of representatives):

| Representative Level | $\sum x$ | $\|B_q\|$ |
|---|---|---|
| 12.345 | … | … |
| … | … | … |

**E (3-3)**

**a. Compare the results from a three bit uniform quantizer to the performance of a Lloyd-Max-Quantizer. Calculate the PSNR for both cases and fill in the table below:**

| Image | $bit/$ intensity value | PSNR Lloyd-Max | PSNR uniform quantization |
|---|---|---|---|
| lena.tif | 3 | **30.0217** | **28.6464** |

## 3.3 Entropy Constrained Vector Quantizer (ECVQ)

In this part we investigate the joint quantization of more than one intensity value.

**P (3-4)**

**a. Consider Figures 19 and 20. Both show the same joint pmf of horizontally adjacent pixel pairs $i_1$ and $i_2$ of the image 'lena.tif'. Black represents a high probability and white corresponds to zero probability. Draw a rough approximation of the quantization cells that would be calculated by a 2-bit Lloyd-Max Scalar Quantizer into Figure 19 (i.e. 2bits per axis!).**

**b. Draw in a rough approximation of the quantization cells that would be calculated by an entropy constrained vector quantizer using λ=0 into Figure 20. Assume the ECVQ is initialized with a 4-bit uniform vector quantizer (ECVQ is performed on blocks of two horizontally adjacent pixels).**

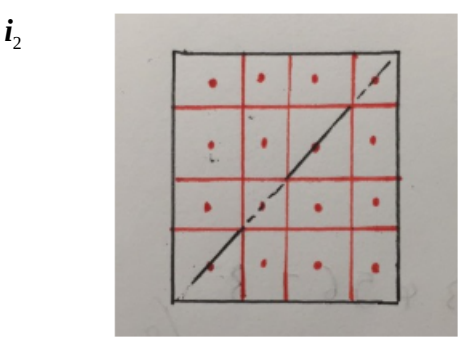



Figure 19: joint histogram of 'lena.tif'

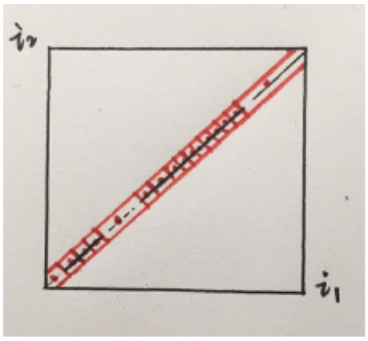



Figure 20: joint histogram of 'lena.tif'

each axis separately

**c. Explain the main advantages and disadvantages of vector quantization in comparison to scalar quantization.**

**Advantages:** better performance, efficiency (lower MSE)
In scalar quantizer there are wasted cells with zero probability.
While in vector quantization, the representative values lie on the diagonal with high probability.

**Disadvantages:** higher computational complexity

## 3.4 Milestone:

At the end of this Chapter you should be able to finish a fully functional still image codec based on vector quantization.

**E (3-4)**

**a. Perform unconstrained VQ on the image 'lena_small.tif'. Choose the input vectors as blocks of intensity values of size 2 by 2 and choose an 8-bit uniform vector quantizer to initialize your codebook. *Reconstruct the image from the Huffman coded bitstream.***

Hints: Don't forget to split the most populated cell if you encounter empty cells (For your convenience just add a small offset to the code book entry of the most populated cell to get a new representative). Choose $\varepsilon = 0.1$ if you work in the intensity value range from 0..255. Add the D-R pair for every iteration to a plot to get an impression of the optimization process.

**b. Determine the bitrate and PSNR of your algorithm and image 'lena.tif'.**

| | | |
|---|---|---|
| **'lena.tif':** | **Bit-rate:** | 5.6744 bits/pixel |
| | **PSNR =** | 34.6496 dB |

**E (3-5)**

**Plot the rate-distortion point for 'lena.tif' with your vector quantizer into the plot from E(1-7).**

# 4. Transform Coding

In Chapter 3 you have developed a lossy compression scheme that is based on vector quantization. This technique introduces an error to the reconstructed image by reducing the number of bits that are used to represent the intensity values of a pixel. This distortion is acceptable to the human visual perception system under certain conditions and can help to reduce the necessary bit-rate significantly. Besides chrominance sub-sampling, entropy coding and quantization there are transform coding techniques which do not reduce the bit-rate directly but help to model the source image and let bit-rate reducing techniques exploit statistical properties to work even more efficiently. Transforms like the DCT or DWT are the core of widespread lossy compression algorithms today. A still image codec based on transform coding is the milestone for this Chapter.

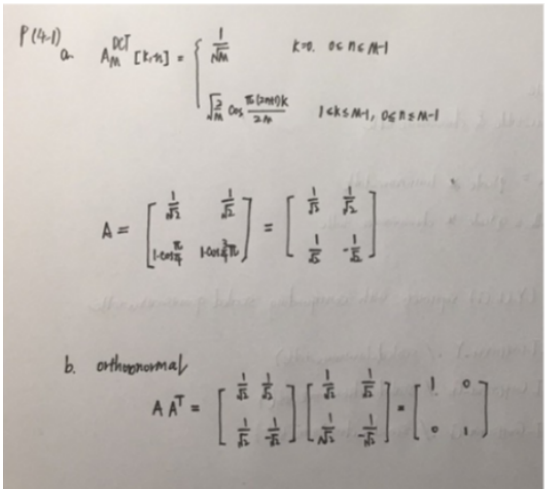
**Note: The preparation has to be done by every student and for all questions in this chapter. The implementation has to be done only for the DCT case. The second part concerning the DWT is voluntary for those who want to base their final codec on it.**

## 4.1 Discrete Cosine Transform (DCT)

The basic building block for the well known JPEG image compression standard is the DCT. This transform is very close to the KLT (Karhunen-Loeve-Transform) which produces pair-wise uncorrelated transform coefficients. Decorrelation of transform coefficients is important to image and video compression as it allows one to treat every coefficient independently without loss of compression efficiency.

**P (4-1)**

**a. Determine the transform matrix $A$ of a 1D Type II-DCT of length 2.**



**Show that $A$ is orthonormal.**

**P (4-2)** **Prepare an algorithm (Matlab notation) that performs a zig-zag scan on DCT coefficients calculated for 8 by 8 blocks of intensity values.**

**P (4-3)**

**Prepare an algorithm (Matlab notation) that performs a scalar quantization on the transform coefficients using predefined quantization tables (see Figures 21 and 22) on a color image in YCbCr format. The quantization is applied in the following way (producing quantization indices!):**

$$i_y = round\left(\frac{y_{L:n_1,n_2}}{L_{n_1,n_2}}\right); \qquad i_C = round\left(\frac{y_{C:n_1,n_2}}{C_{n_1,n_2}}\right)$$

**where $n_1,n_2$ are the coordinates of the coefficient within a 8 by 8 block and $L_{n1,n2}$ and $C_{n1,n2}$ are the corresponding elements from the quantization tables for luminance and chrominance components and are weighting factors according to the sensitivity of the human visual perception system.**

$$L = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 55 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 21: Quantization table (luminance)

$$C = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 13 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

Figure 22: Quantization table (chrominance)

**E (4-1)**

**a. Implement a function** `DCT8x8()` **that performs a 2D DCT on an 8 by 8 block of intensity values (use the Matlab command 'dct'). Color planes are processed separately (Assume an YCbCr representation for the input image). Implenent a function** `IDCT8x8()` **for the inverse transform.**

Hint: The `dct()`/`idct()` command only processes columns. You have to perform the dct/idct on columns and rows separately.

**b. Implement a function** `Quant8x8()` **to quantize the transform coefficients of an 8 by 8 block using the quantization tables from problem P(4-3). Implement a function** `DeQuant8x8()` **for the inverse operation.**

**c. Implement a function** `ZigZag8x8()` **for the zig-zag scan on the quantized coefficients of an 8 by 8 block. Implement a function** `DeZigZag8x8()` **for the inverse operation.**

**d. Implement a function** `ZeroRunEnc_EoB()` **for zero-run encoding on the zig-zag scanned DCT-coefficients of an 8 by 8 block. Implement a function** `ZeroRunDec_EoB()` **for the zero-run decoding.**

***e.*** **Convert the image 'lena_small.tif' to the YCbCr domain using ICT and process it in the following steps: DCT, quantization, zig-zag scan, zero-run encoding and Huffman table computation.**

***f.*** **Convert 'lena.tif' to YCbCr domain using ICT and then implement a function** `IntraEncode()` **to compress the YCbCr image in the following steps: DCT, quantization, zig-zag scan, zero-run encoding and Huffman coding. Use the Huffman table calculated from 'lena_small.tif'.**

***g.*** **Implement a function `IntraDecode()` to decode the YCbCr image from the huffman encoded bitstream. Convert the reconstructed image back to the RGB domain and calculate the PSNR and bit-rate.**

| | | |
|---|---|---|
| **'lena.tif':** | **bit-rate:** | **1.3583 bbp** |
| | **PSNR =** | **32.9116 dB** |

**E (4-2)**

**Plot the rate-distortion curve (You can adjust the rate-distortion trade-off by multiplying the quantization matrices with a scalar factor!) for your transform still image coder and different test images into the plot from E(1-7). Compare your result to Figure 23.**

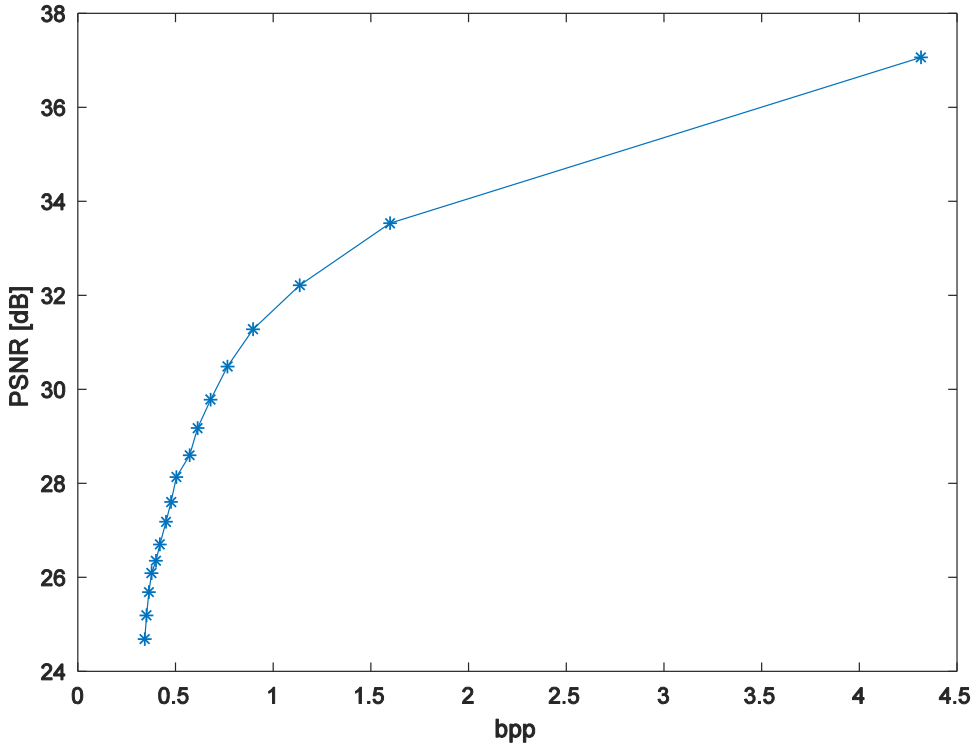



Figure 23: Operational rate-distortion performance of the still image codec based on the JPEG standard encoding algorithm. Test image: 'lena.tif' with Huffman code trained on 'lena_small.tif'.

## 4.2 Subband Coding and the Discrete Wavelet Transform (DWT)

A principle limitation of block transforms (like the DCT) is that the source image must be processed in independent blocks. As highly correlated neighboring pixels can lie within different blocks these blocks exhibit residual correlation. Only for a block size equal to the image size, a full decorrelation of the coefficients can be achieved with the cost of high computational effort. Subband and wavelet transformations on the other hand process the whole image at once. The connection between subband coding and wavelet based coding schemes is that close that those terms are often interchangeable. More particularly, the Discrete Wavelet Transform (DWT) is generally understood as a tree-structured subband transform. The DWT is the main building block of JPEG2000 which is the successor of JPEG with a better performance in the rate-distortion sense.

**P (4-5)**

**a. Draw the block structure of a 3-stage 1D tree-structured subband coder into Figure 24. How many filtering and decimation blocks are needed?**

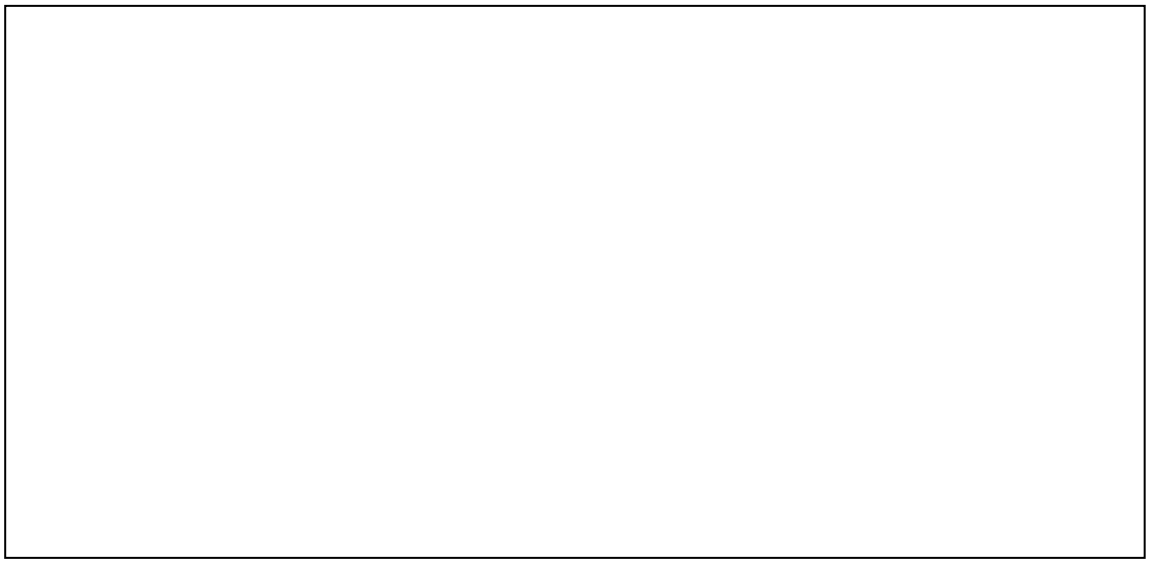


Figure 24: block structure of a 3-stage subband coder

**b. Name the output signals of problem a. and draw the approximated passband structure of the output signals into Figure 25. The used filters are quadrature mirror filters.**

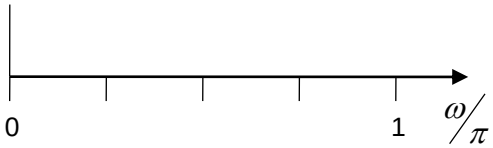



Figure 25: Passband structure of a 3-stage filterbank using QMF

**E (4-3)** **Implement a recursive algorithm that computes the n-stage DWT (`dwt_97()`) for grayscale images using the filters given and calculated in problem P(4-6). Use the image 'lena_gray.tif' as test image. To process more component images concatenate the components to get a larger but only one component image.**

**E (4-4)** **Implement a recursive algorithm that computes the n-stage inverse DWT (`idwt_97()`) for the output of the function from problem E(4-3).**

## 4.2.2 Embedded Zero Tree Wavelet Algorithm

As natural images generally have low pass characteristics most of the DWT coefficients in higher subbands are very small. This property is exploited by quantization and performing a zero tree encoding.

**E (4-5)** **Implement the EZW Algorithm for the 3-stage DWT implementation from problem E(4-3).**

a. **Peform the subband decomposition on the image.**
b. **Choose $T=6$ as the deadzone width (integer representation – zero mean) and quantize the input samples to even values (step size 2). I.e.: Only keep even valued coefficients with an absolute value greater than or equal to six.**
c. **Process each subband separately and use the scanline scanning order through each subband (and determine the children in the corresponding subbands).**
d. **Instead of arithmetic coding just spend two bits on the 4 significance symbols (POS, NEG, ZTR, IZ) and one bit for the refinement symbols (0, 1).**
e. **Implement the decoder. Use the image 'lena_gray.tif' as test image and extend your algorithm to enable color image processing.**

**'lena_gray': compression ratio:**

**PSNR =**

**E (4-6)** **Let the decoder stop reading from the bitstream after about 70kB of the encoded stream is processed and note the achieved PSNR for the test image 'lena_gray.tif'.**

**'lena_gray': compression ratio:**

**PSNR =**

**E (4-7)** **Plot the Rate-Distortion point for your transform still image coder and different test images into the plot from E(1-7).**

**E (4-8)** **Play with the deadzone threshold $T$ and the quantization and compare the results to the DCT-mode. Calculate the entropy of the significance sequence and the refinement sequence.**

# 5. Motion Compensation

Up to now we have considered lossy and lossless still image compression. You mainly exploited the spatial redundancy (low pass properties of natural images). From now on we want to encode sequences of images and therefore have a temporal dimension to take into account. Compensating for object or camera motion to exploit temporal redundancy is what you will do in this chapter. The milestone for this Chapter is a video codec based on motion compensation.

## 5.1 Temporal prediction

In the previous Chapters a still image codec based on spatial prediction has been implemented. For image sequences, pixels found in previous or future frames are used to predict intensity values in the current frame. Rather than describing the motion of every pixel in a frame, pixel blocks are used to achieve a reasonable trade-off between signaling overhead and motion field accuracy. One motion vector per block is signaled to the decoder and the process is called motion compensated prediction (MCP).

The test sequences used in this chapter are 20 frames taken from the

‘foreman’ sequence,
‘akiyo’ sequence,
‘coastguard’ sequence,
‘container’ sequence,
‘news’ sequence, and
‘silent’ sequence.

All of them are well known test sequence used in video compression. The zip-archives can be downloaded from the class web page.

**P (5-1)**

**Figure 24 shows the structure of a hybrid video coder. You already have implemented one technique for almost every block in the codec.**

**a. Note the blocks for which you have not implemented an algorithm so far.**

**motion compensated predictor**
**motion estimator**

**b. Note the blocks for which you already have implemented an algorithm and note the techniques you implemented.**

**(1) Color Transform: convert RGB to YCbCr**
**(2) Transform: DCT**
**(3) Quantizer: quantize DCT coefficients of luminance and chrominance channel with corresponding quantization table and introduce qScale to control the accuracy of quantization(rate)**
**(4) entropy coding: marginal Huffman coding**

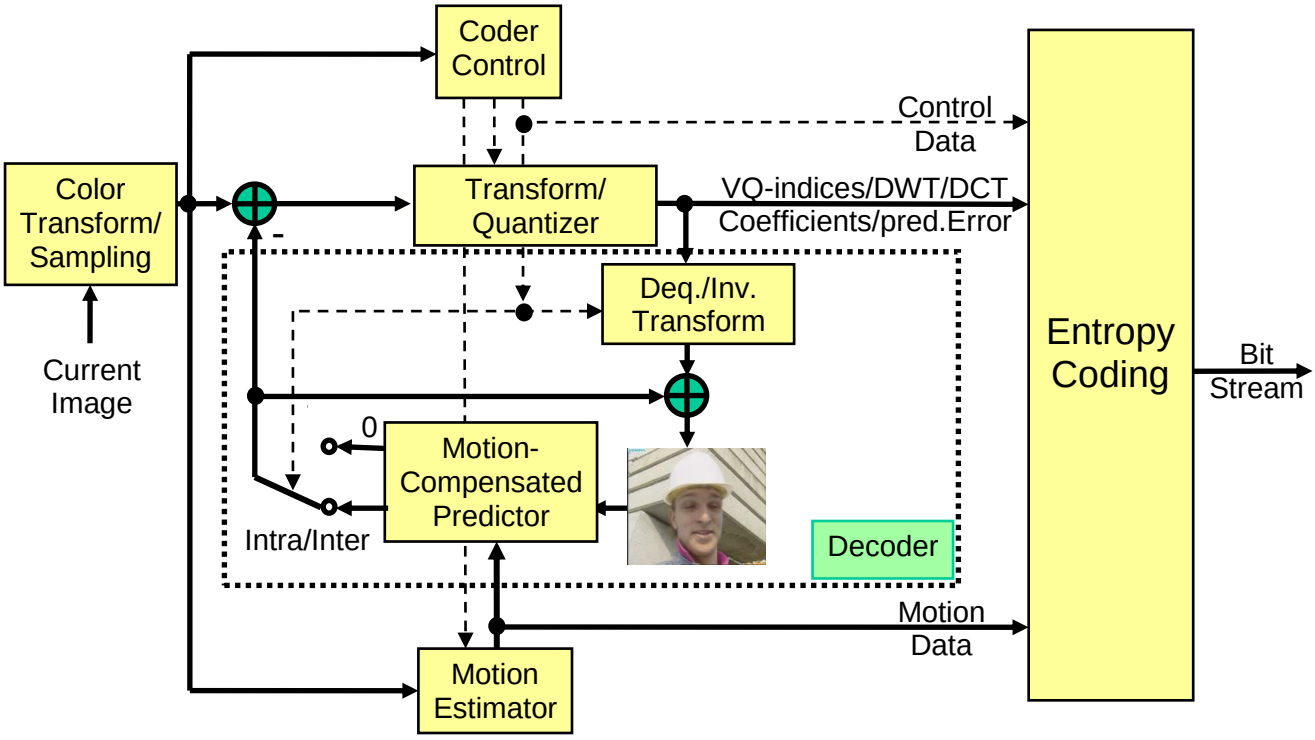



Figure 24: Hybrid video coder (with embedded decoder)

*image source: Prof. Thomas Wiegand TU Berlin/HHI*

**P (5-2)**

**a. Determine the number of operations (i.e. summations, multiplications) one has to perform for full search of n by n intensity blocks within a search area of $\pm m$ pixels. Assume that the sum of squared differences (SSD) distortion measure is used to find the best match.**

number of comparison: (2m+1)^2

each comparison of two nxn blocks: (3*n^2-1)

total number of operations: (2m+1)^2 * (3*n^2-1)

**b. Repeat problem a. using the sum of absolute differences (SAD) as the distortion measurement to find the best match.**

number of comparison (2m+1)^2

each comparison of two nxn blocks: (2*n^2-1) operations

total number of operations: (2m+1)^2 * (2*n^2-1)

**P (5-3)**

**Determine the number of candidate motion vectors that can occur and determine the size of the resulting matrix containing the motion vectors for images in CIF-format.**

**Number of candidate motion vectors per search area:** (2m+1)^2

**Number of motion vectors per CIF-frame:** (352/n) * (288/n) block size(nxn)

**E (5-1)**

**For the following tasks first have a look at the overall pipeline (Figure 25) of the motion estimation and motion compensation based video codec which you will develop.**

**a. The compression scheme used for the first frame to be encoded has to be still image compression. Use the "intra mode" you implemented in Chapter 4. Note the PSNR and bit rate for the first reconstructed frame of the test sequence: 'fore0020.bmp'. Use the Huffman code obtained from the intra-encoded 'lena_small.tif' E (4-1e).**

**PSNR:** 33.7211 dB

**Bit-rate:** 1.3590 bits/pixel

**b. Perform the color transform for all frames coded in "inter mode".**

**c. Perform full search block matching for 8x8 pixel blocks in the current frame within a $\pm 4$ pixel search area. Use the SSD as the distortion measure. Only the luminance component should be used for full search (i.e. actually search for 8x8x1 block correspondences). Find the minimum of the SSD within each search area and and save the resulting motion vector in the matrix discussed in P(5-3)b.**

**d. Build a predicted image by performing motion compensation. Calculate the prediction error for the luminance and the chrominance components.**

**e. Use Huffmann coding to encode the motion vector field. Use the motion vector field from first to second frame to train your Huffmann code.**

**f. Encode the prediction error using the function `IntraEncode()` implemented in Chapter 4. Modifications may be needed. Use the prediction error of the second frame to train your Huffmann code.**

**g. Plot the operational D-R curve for your codec using the foreman test sequence. Use the "intra mode" only for the first frame (Still image Mode). To evaluate your Video Codec determine the average PSNR over all frames and also mean the rate over the whole sequence. Determine at least 5 different D-R pairs (PSNR should be approximately within the range of 25 to 40dB). Your curves should look similar to the ones in Figure 26.**

**h. Encode at least three sequences out of those you can download from the webpage. Please make sure that the D-R curve for the 'foreman' sequence is included in the final presentation.**

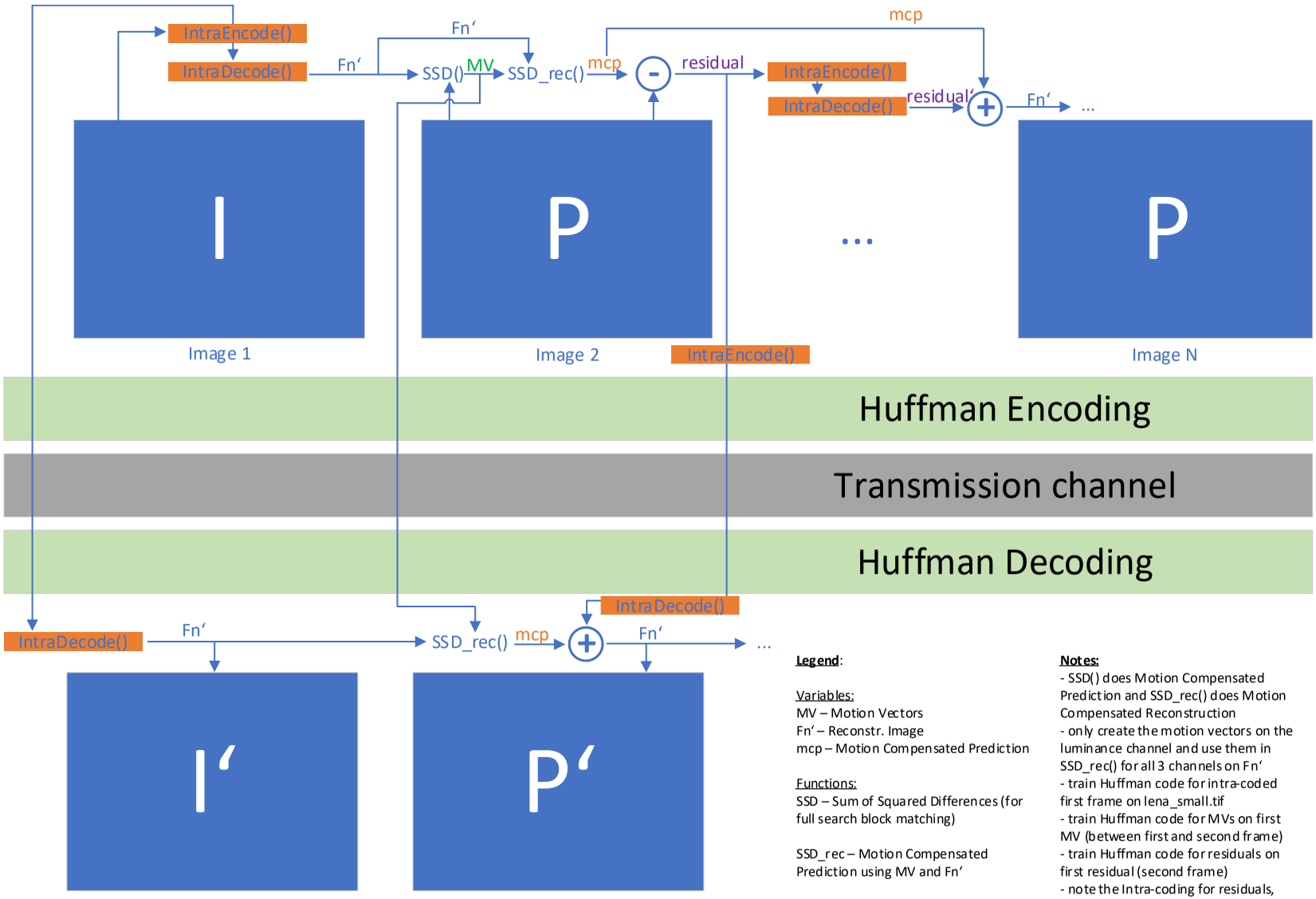





Figure 25: Pipeline for the motion estimation and motion compensation based Video Codec

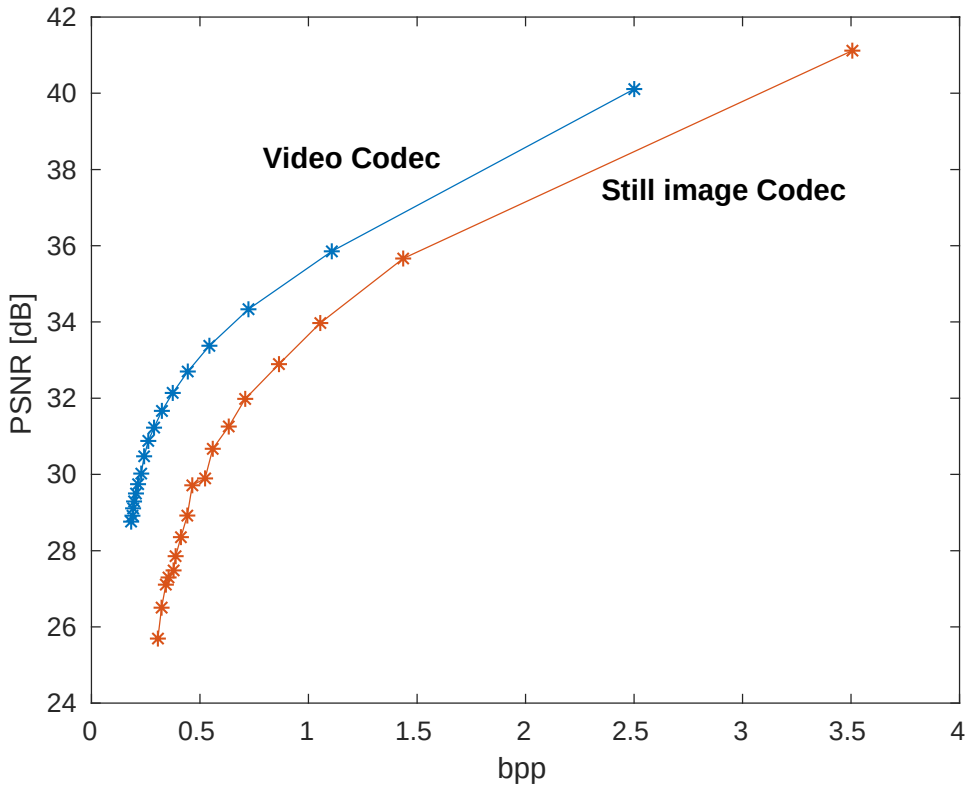



Figure 26: Expected performance of the image and video compression scheme implemented during the IVC-LAB (mean PSNR (RGB) over mean rate). Test sequence: Foreman.

**qscale range**

**adapt lower bound upper bound**

# 6. Optimization

During the lessons of the laboratory you have implemented several modules used in lossy and lossless image and video compression. These modules can be arranged in many ways to get coding schemes with different properties and different trade-offs between performance and complexity. A variety of optimizations can be applied to the modeling and encoding steps. In this Chapter you will compose your own codec and optimize it with respect to

- bitrate
- quality
- execution speed

Your codec should be tested on the provided sequences but still be able to process arbitrary sequences given as images in RGB format. In the following Sections some suggestions for the next two weeks of the lab are introduced.

You are free to choose the components of the codec that you will present at the end of this course. One example would be DCT-based encoding of the residual error using motion compensated prediction and a DWT-based coding scheme for the intra mode. This is just an example – try to develop your own codec. Also work out the differences to standard codecs to prepare for your presentation. Do not concentrate on compression ratio too much. E.g. a codec that allows a rate distortion adjustment and uses a fast decoding algorithm is also a very good outcome of this laboratory.

## 6.1 Modeling

In most cases when calculating code tables or representative vectors you used only one representative image: ‘lena_small.tif’. The statistics obtained here do not fit for the foreman sequence or any arbitrary video. You might want to collect a number of images that represent natural images and videos better and gather statistics using this test image set. Consider adaptive updating of statistics while encoding and decoding or storing statistics with the bit stream.

Note: Make sure your codec can handle different frame sizes. Do not adapt your algorithm to the foreman sequence as your codec will be tested on an arbitrary sequence!

## 6.2 Simplifications

Simplifications have been made in modeling and coding steps during this course. To improve the coding performance, review your modules. As an example the DCT algorithm you implemented does not encode DC and AC coefficients separately and the motion compensation algorithm does not work on sub sampled chrominance components. You may want to implement these and other improvements.

## 6.3 Rate-Distortion

Keep log of your improvements. You might want to modify your algorithm to remember former results and visualize them in a rate-distortion plot. This feature would give you a tool to adjust quantization tables or other parameters to optimize bit allocation and quality trade-offs. Do not just optimize PSNR and MSE. Always take a look at the result!

## 6.4 Execution speed

Execution speed can be increased significantly by avoiding time consuming calculations in loops. Parameters should be calculated in advance if possible. Initialize variables before using them. Think about implementing the fast DCT algorithm or logarithmic search for example. Execution speed and memory consumption are related. Try to allocate memory economically to avoid time consuming automatic reallocations. Use the ‘clear <variable>’ command to free variables that are not used anymore.

# 7. Final Presentation

During the lessons of the laboratory you have implemented and optimized your hybrid video codec. In the final presentation you will explain the general structure of your codec and the detailed implementation of your optimizations.

## General Guidelines:

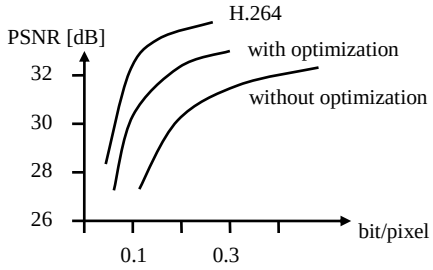
- Duration per group: Approximately 15 minutes, followed by a discussion and questions of about 5 minutes.

- You can use either transparencies or PowerPoint. You might want to use the white board to explain topics in more detail. Power Point files have to be submitted at least one day before your presentation if you do not use your own laptop.

- Briefly explain the structure of your codec in a few minutes. Do not explain the hybrid coding structure too much in detail. Concentrate on your own extensions and make references to the general structure.

- You may want to analyze your optimization with respect to compression ratio, image and video quality, and execution speed.

- **You must show rate-distortion curves for your codec! Show the operational rate-distortion function for the whole test sequence (mean PSNR from about 25dB to 40dB and the corresponding mean rate – before and after optimization). E.g.:**




- Explain the statistics and tables you used and state improvements and losses in comparison to other models.
- Explain the simplifications you made in order to adjust the trade-off between compression ratio, video quality and computational complexity.
- You might want to compare compression and decompression time and give some reasons.
- Please do not show too much Matlab code (except for the case that it is really good looking).
- If you worked in groups every student should concentrate on one part of the codec and optimizations. Please do not repeat too much of the general things.