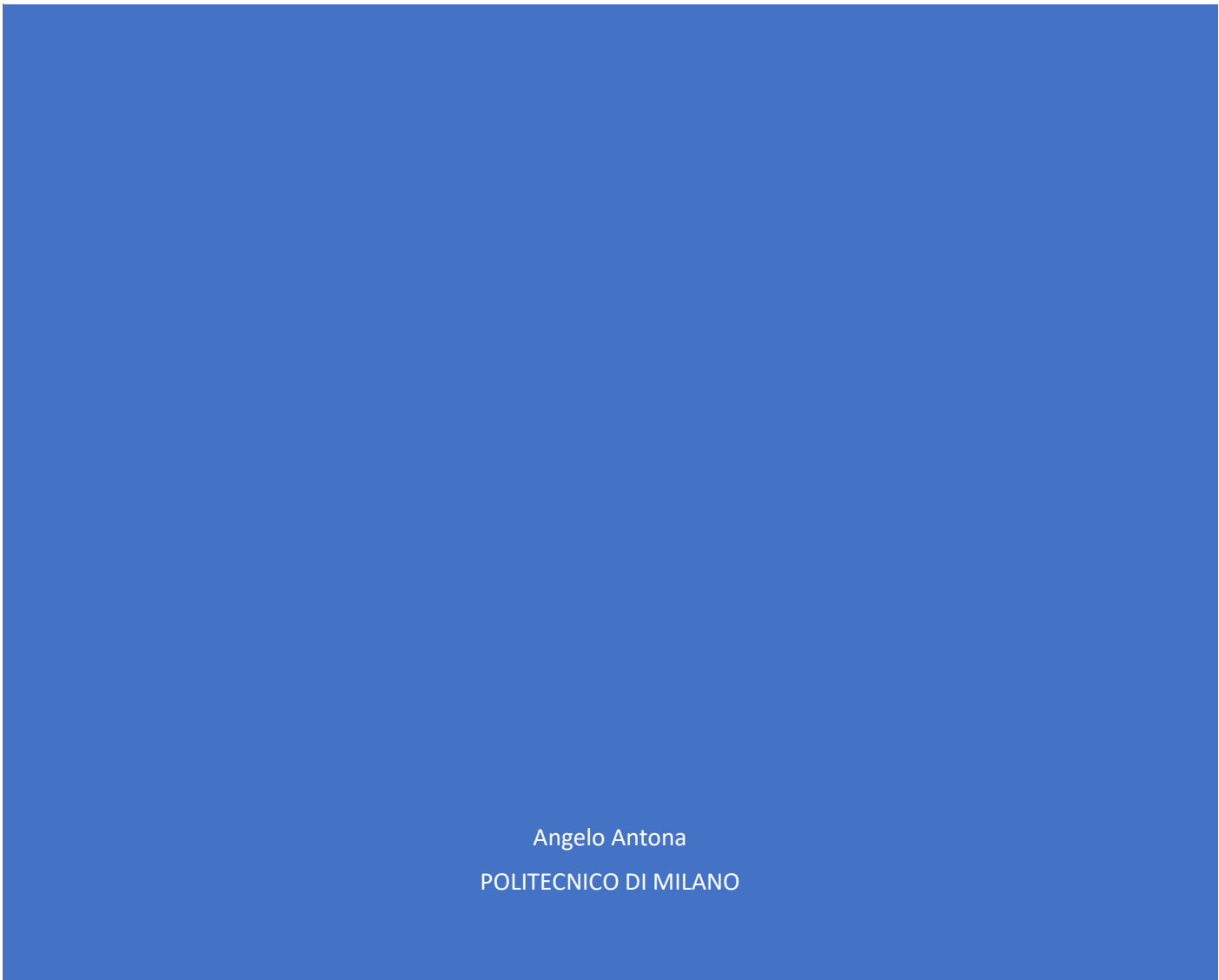




RIASSUNTO MSP2



Angelo Antona
POLITECNICO DI MILANO

1 TABLE OF CONTENTS

2	Lossless coding	5
2.1	Different types of codes.....	5
2.2	Important theorems about codes.....	5
2.3	Lossless Coding in MATLAB.....	5
2.3.1	Estimate the PMF of a Source.....	5
2.3.2	Calculate the Entropy of a Source.....	6
2.3.3	Estimate the Joint PMF and get the Entropy of two Sources X and Y	6
2.4	Questions from past exams	6
2.4.1	Question about Entropy.....	6
2.4.2	September, 10 2013.....	6
3	Lossy Coding: Quantizers	8
3.1	Quantizers	8
3.1.1	Scalar quantizer.....	8
3.1.2	Uniform Scalar Quantizer.....	8
3.1.3	Uniform Scalar Quantizer in MATLAB	8
3.1.4	Optimum Uniform Scalar Quantizer	9
3.1.5	Optimum Uniform Scalar Quantizer in MATLAB.....	9
3.1.6	Lloyd—Max Quantizer	9
3.1.7	Lloyd-Max Quantizer in MATLAB	9
3.1.8	Entropy-Constrained Scalar Quantizers (ECSQ)	10
3.1.9	Vector Quantizer	10
3.1.10	Vector Quantizer in MATLAB	11
3.2	Compander-Expander	11
3.3	Rate-Distortion Theorem	12
3.3.1	Key Concepts of the Rate-Distortion Theorem	12
3.3.2	Implications of the Theorem.....	12
3.3.3	Useful formulas	12
3.4	Questions from past exams	12
3.4.1	July, 12 2016.....	12
3.4.2	September, 26 2012.....	14
4	Lossy Coding: Predictive Coding	15
4.1	Differential Pulse Code Modulation (DPCM)	15
4.1.1	Open Loop DPCM.....	15
4.1.2	Closed Loop DPCM	16
4.1.3	Open Loop vs Closed Loop DPCM	16
4.1.4	Open Loop vs Closed Loop DPCM (Matlab)	17
4.2	Questions from past exams	17
4.2.1	September, 12 2016.....	17

4.2.2	January, 15 2021	18
5	Lossy Coding: Transform Coding	20
5.1	Transform Coding general concept.....	20
5.1.1	Geometric interpretation of Transform Coding.....	20
5.1.2	Bit allocation	21
5.1.3	MATLAB Implementation of a general Transform Coding Algorithm	22
5.2	Different Types of Transformation Matrices	22
5.2.1	Karhunen-Loève Transform (KLT)	22
5.2.2	MATLAB implementation of KLT	22
5.2.3	Discrete Cosine Transform (DCT)	23
5.2.4	MATLAB implementation of DCT	24
5.3	Questions from past exams	24
5.3.1	January, 15 2021	24
5.3.2	February, 18 2021	24
6	Speech Coding.....	26
6.1	Speech Signals Characteristics	26
6.2	Linear Predicting Coding for Speech	26
6.2.1	Data Model.....	26
6.2.2	Source-filter Interpretation.....	27
6.2.3	Application of this Model into LPC.....	27
6.3	Waveform Coders	28
6.3.1	PCM in Waveform Coders.....	29
6.3.2	DPCM in Waveform Coders	29
6.4	Vocoders	30
6.4.1	Channel Vocoder.....	31
6.4.2	Linear Predictive Vocoder	31
6.4.3	Open Loop Excitation Models	33
6.5	Analysis Linear Predictive Coders MATLAB implementation.....	34
6.5.1	Windowing Configuration.....	34
6.5.2	Voiced vs Unvoiced detection.....	34
6.5.3	Speech Synthesis.....	35
6.6	Analysis-by-synthesis Linear Predictive Coders	35
6.7	Common Excitation Models used in Speech Processing.....	36
6.7.1	MultiPulse Excited Linear Prediction (MPLP).....	36
6.7.2	Regular Pulse Excitation (RPE)	36
6.7.3	Code Excited Linear Prediction (CELP)	37
7	Audio Coding.....	38
7.1	Perceptual Coding General Process	38
7.1.1	Encoder	38

7.1.2	Decoder.....	39
7.2	Ear Anatomy and Psychoacoustic Basics	39
7.2.1	The ear	39
7.2.2	SOUNDS ILLUSIONS	41
7.2.2.1	<i>Superposition of Pure Tones - 1st Order Effect</i>	41
7.2.3	Timbral Consonance and Dissonance	41
7.2.4	Pitch: The Subjective Impression of Frequency	42
7.2.5	Masking.....	44
7.3	Perceptual Models	44
7.4	MPEG-1 Psychoacoustic Model 1.....	45
7.5	Filter banks.....	47
7.5.1	Filter Banks Requirements	47
7.6	Filter Bank Families	48
7.6.1	Pseudo-Quadrature Mirror Filter (PQMF) filter banks	48
7.6.2	Modified Discrete Cosine Transform (MDCT)	48
7.6.3	Adaptive Filter Banks	49
7.6.4	Hybrid filter Banks.....	50
7.7	Quantization in Audio Coding	50
7.7.1	Bit Allocation	50
7.8	Entropy Coding in Audio Coding	51
8	Image Coding	52
8.1	Digital Images.....	52
8.2	Lossless Coding in Digital Images.....	52
8.3	Lossy Coding in Digital Images	52
8.4	Redundancy and Irrelevancy.....	53
8.4.1	Statistical Redundancy	53
8.4.2	Visual Irrelevancy	53
8.4.3	RGB → YCbCr conversion in MATLAB	54
8.5	Image Compression Systems	54
8.6	2D transforms in Image Coding.....	55
8.6.1	2D Transforms Properties	55
8.6.2	Two important 2D Transforms: DFT and DCT	56
8.6.3	DCT for Image Coding in MATLAB.....	57
8.6.4	Choice of Block Size.....	57
8.6.5	Quantization of the Transform Coefficients	57
8.7	Zonal Coding.....	58
8.8	Threshold Coding	58
8.9	JPEG.....	59
8.10	JPEG MATLAB implementation	60

8.10.1	JPEG compression (DCT)	60
8.10.2	JPEG compression (KLT)	61
9	Video Coding	62
9.1	Motion Estimation	62
9.1.1	Issues in Motion Estimation	62
9.1.2	Block Matching.....	63
9.1.3	Motion compensation.....	64
9.1.4	Displaced Frame Difference (DFD).....	64
9.1.5	Mean Square Error (MSE) and Peak Signal Noise Ratio (PSNR)	64
9.1.6	Matching Criteria	65
9.2	Search Criteria.....	65
9.2.1	Full Search	65
9.2.2	Fast Search	66
9.3	Sub-pixel Motion Estimation.....	66
9.4	Progressive Scanning and Interlaced Scanning (<i>time sampling</i>).....	67
9.4.1	Progressive Scan.....	67
9.4.2	Interlaced Scan.....	67
9.5	Colour Video Sequences	67
9.6	Hybrid DPCM/DCT video coding	67
9.6.1	Encoding Process	68
9.6.2	Decoding Process	69
9.6.3	Key Concepts in Hybrid DPCM/DCT Video Coding.....	69
9.7	Video Coding in MATLAB	70
9.7.1	Motion Estimation on a single image block	70
9.7.2	Motion Estimation and Motion Compensation on a whole image.....	71
10	Other MATLAB Utilities	72
10.1	Linear Regression	72
10.2	Define <i>x_axys</i> in Frequency Domain.....	72
11	Mind map.....	73

2 LOSSLESS CODING

Lossless coding is a method of data compression that allows the original data to be perfectly reconstructed from the compressed data. It eliminates redundancy in the data without losing any information, ensuring that the decompressed data is identical to the original. It is adoptable only when we are dealing with finite alphabet sources.

2.1 DIFFERENT TYPES OF CODES

One to One Code: A one-to-one code (or injective code) maps each symbol from the source alphabet to a unique code word in the target alphabet, with no two different source symbols being mapped to the same code word. This property ensures that each code word uniquely identifies its corresponding source symbol. However, a one-to-one code does not inherently guarantee efficient decodability or the absence of ambiguity in the concatenation of code words.

Uniquely Decodable Code: A uniquely decodable code is a code in which a sequence of code words can be decoded into the original sequence of source symbols in only one way, regardless of how the code words are concatenated. This means that, given a sequence of code words, there is a unique way to partition and map them back to the original sequence of symbols from the source alphabet. Uniquely decodable codes eliminate ambiguity in decoding sequences of symbols but do not necessarily provide a way to determine where one code word ends and the next begins without additional information.

Prefix (Instantaneous) Code: A prefix code (or instantaneous code) is a special type of uniquely decodable code in which no code word is a prefix of any other code word. This property ensures that the code can be decoded instantly, symbol by symbol, without needing to look ahead to subsequent symbols for context.

While all prefix codes are uniquely decodable, not all uniquely decodable codes are prefix codes. The prefix property is a stricter requirement that removes the ambiguity in decoding sequences of code words without needing additional information. Uniquely decodable codes ensure that there is only one way to decode a sequence, but decoding might require looking at a larger context or even the entire sequence.

2.2 IMPORTANT THEOREMS ABOUT CODES

Kraft inequality: The Kraft inequality provides a necessary and sufficient condition for the existence of a prefix code with a given set of codeword lengths: $\sum (2^{-l_i}) \leq 1$ where l_i are the length of the codewords associated with the symbols of the alphabet. This inequality ensures that the codewords are structured in such a way that no codeword is a prefix of another.

Shannon's Source Coding Theorem: Shannon's Source Coding Theorem establishes the fundamental limit on the compressibility of information for lossless encoding: $H(X) \leq L < H(X) + 1$.

The theorem implies that it is impossible to losslessly compress any data source to a size below its entropy without losing information, but it is possible to get arbitrarily close.

2.3 LOSSLESS CODING IN MATLAB

2.3.1 Estimate the PMF of a Source

```
% Define the decision regions
alphabet=unique(y);
edges = min(alphabet)-0.5:1:max(alphabet)+0.5;
% Generate an histogram of the source realization
histogram = histcounts(x,edges);
% Normalize the histogram to get the PMF
pmf=histogram/sum(histogram);
% Plot the PMF
bar(pmf);
```

2.3.2 Calculate the Entropy of a Source

```
% Entropy
H = -sum(PMF(PMF>0) .* log2(PMF(PMF>0))) ;
```

2.3.3 Estimate the Joint PMF and get the Entropy of two Sources X and Y

```
alphabet_x=unique(x);
alphabet_y=unique(y);
edges_x = min(alphabet_x)-0.5:1:max(alphabet_x)+0.5;
edges_y = max(alphabet_y)-0.5:1:max(alphabet_y)+0.5;

% P(x,y)
hist_xy=histcounts2(x,y,edges_x,edges_y);
PMF_xy = hist_xy/sum(hist_xy(:));
% H(x,y)
H_joint=-sum(sum(PMF_xy(PMF_xy>0) .* log2(PMF_xy(PMF_xy>0)))) ;
% Plot
imagesc(PMF_xy);
```

2.4 QUESTIONS FROM PAST EXAMS

2.4.1 Question about Entropy

Consider two discrete memoryless sources X and Y characterized by a finite-element alphabet. You know the value of the entropy $H(X)$, the entropy $H(Y)$, and the mutual information $I(X, Y)$.

- a) Using entropy properties, derive the value of the joint entropy $H(X, Y)$.

To derive the value of the joint entropy we can use the relationship between joint entropy $H(X, Y)$, conditional entropy $H(X|Y)$ and mutual information $I(X, Y)$.

Remembering the draw on the right, we can express the joint entropy as: $H(X, Y) = H(X) + H(Y) - I(X, Y)$.

- b) Using entropy properties, derive the value of the conditional entropy $H(X|Y)$.

Using the previously seen draw, we can write that: $H(X|Y) = H(X) - I(X, Y)$

- c) What is the meaning of zero mutual information? In which case does it happen?

Mutual information measures the statistical dependence between two variables X and Y.

Zero mutual information $I(X, Y) = 0$ indicates that knowing the value of one variable provides no information about the other. Therefore, it happens when the two variables X and Y are completely independent and there is no correlation between them.

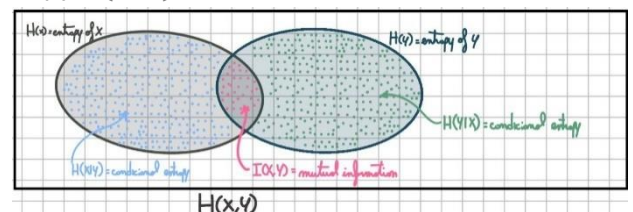


Figure 1

2.4.2 September, 10 2013

Consider a sequence of symbols abbbbbaaacccacbcccbcacbbbbbcb emitted by a source X.

- a) Learn the pmf of the source from the observed data and compute the entropy of the source (assuming a memoryless source).

In the sequence of symbols $\{abbbbbaaacccacbcccbcacbbbbbcb\}$ we have an alphabet of 3 symbols a (emitted 6 times), b (emitted 13 times) and c (emitted 11 times), for a total of 30 emitted symbols. With this information we can estimate the pmf of the memoryless source: $p(a) = \frac{6}{30}$; $p(b) = \frac{13}{30}$; $p(c) = \frac{11}{30}$.

Consequently, the entropy of the source is:

$$H_1(X) = - \sum (p(a) * \log_2(p(a))) \rightarrow -\frac{6}{30} * \log_2\left(\frac{6}{30}\right) - \frac{13}{30} * \log_2\left(\frac{13}{30}\right) - \frac{11}{30} * \log_2\left(\frac{11}{30}\right) = 1,52 \text{ bits}$$

- b) To exploit source memory, group symbols in blocks of size $N = 2$ and compute $H_2(X)$.

If we group symbols in blocks of $N = 2$ we get $\{ab; bb; ba; aa; cc; ca; cb; cc; cc; bc; ac; bb; bb; bb; bc\}$. In this case we have the symbols ab (1 time), bb (4 times), ba (1 time), aa (1 time), cc (3 times), ca (1 time), cb (1 time), bc (2 times), ac (1 time), for a total of 15 symbols emitted from the source. The pmf in this case is:

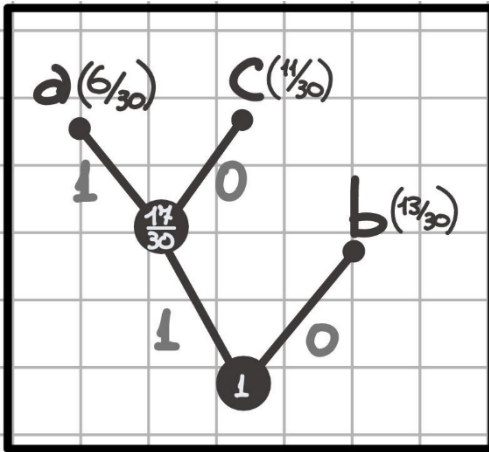
$$p(ab) = \frac{1}{15}; p(bb) = \frac{4}{15}; p(ba) = \frac{1}{15}; p(aa) = \frac{1}{15}; p(cc) = \frac{3}{15}; p(ca) = \frac{1}{15}; p(cb) = \frac{1}{15}; p(bc) = \frac{2}{15}; p(ac) = \frac{1}{15}.$$

We can easily calculate the entropy:

$$H_2(X) = - \sum (p(a) * \log_2(p(a))) \rightarrow 6 * \left(-\frac{1}{15} * \log_2\left(\frac{1}{15}\right) \right) - \frac{2}{15} * \log_2\left(\frac{2}{15}\right) - \frac{3}{15} * \log_2\left(\frac{3}{15}\right) - \frac{4}{15} * \log_2\left(\frac{4}{15}\right) = 2,92 \text{ bits}$$

- c) Compute the number of bits needed to encode the sequence $bbbcaa$ using Huffman coding in two cases:
- without exploiting source memory.
 - exploiting source memory.

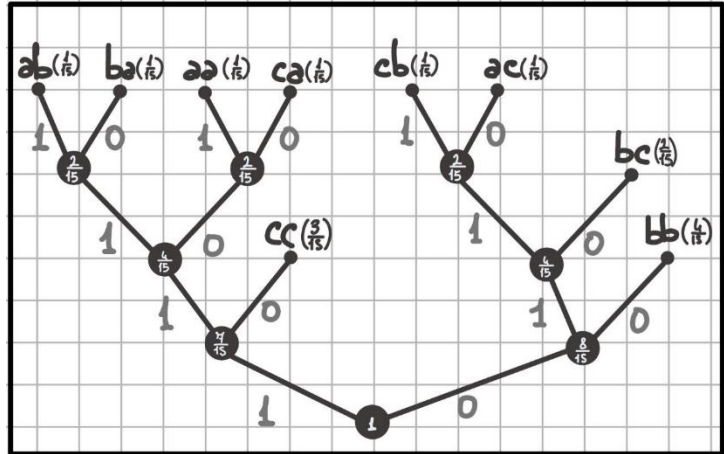
- a) Huffman coding without exploiting source memory:



$$a = 11; b = 0; c = 10$$

$$E(L) = \frac{6}{30} * 2 + \frac{11}{30} * 2 + \frac{13}{30} * 1 = 1,57 \frac{\text{bits}}{\text{symbol}}$$

- b) Huffman coding exploiting source memory:



$$ab = 1111; ba = 1110; aa = 1101; ca = 1100; cb = 0111; ac = 0110;$$

$$cc = 10; bc = 010; bb = 00.$$

$$E(L) = 6 * \left(\frac{1}{15} * 4 \right) + \frac{2}{15} * 3 + \frac{3}{15} * 2 + \frac{4}{15} * 2 = 2,93 \frac{\text{bits}}{2 \text{ symbols}} = 1,46 \frac{\text{bits}}{\text{symbol}}$$

Figure 2

3 LOSSY CODING: QUANTIZERS

Lossy coding is a data compression technique that reduces the size of data by permanently eliminating certain information, particularly redundant or less important data. Unlike lossless coding, lossy coding does not allow for the original data to be perfectly reconstructed from the compressed version.

3.1 QUANTIZERS

A quantizer is a process or device that converts a continuous range of values into a finite set of discrete values.

3.1.1 Scalar quantizer

A scalar quantizer is a type of quantizer that operates on individual samples of a signal independently, converting each input value (a scalar) into a discrete level.

A scalar quantizer is defined by:

- A set of decision regions I_k with $k = 1, 2, \dots, n$.
- A set of reproduction values y_k with $k = 1, 2, \dots, n$.
- The quantization rule: $q(x) = y_k$.

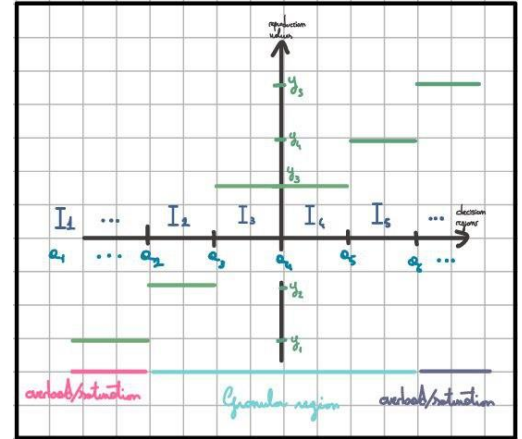


Figure 3

There are two common types of scalar quantizers:

- **Mid-rise quantizer:** The mid-rise quantizer is designed such that the quantization levels (or steps) are centred around zero and do not include zero itself. This type of quantizer starts with a "rise" from zero, and the quantization levels are placed at intervals above and below zero. Mid-rise quantizers are often used in scenarios where the input signal does not center around zero or where having zero as a non-quantization level is advantageous.
- **Mid-tread quantizer:** The mid-tread quantizer, in contrast, includes zero as one of its quantization levels. It is characterized by having "treads" or steps that start at zero, making it suitable for signals that are centered around zero or have zero as a common value.

We use a mid-tread quantizer when dealing with signals that have zero as a common value or when it's crucial to minimize the quantization error for values close to zero. This is often the case in audio and image processing, where a zero value may represent silence or a neutral color/brightness level.

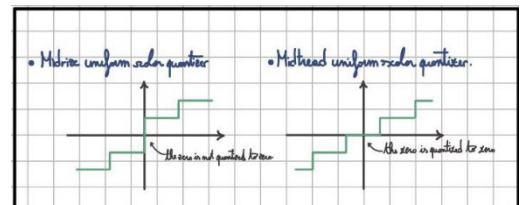


Figure 4

3.1.2 Uniform Scalar Quantizer

A uniform scalar quantizer is a type of scalar quantizer where the input range is divided into equally sized intervals ($a_{k+1} - a_k = \Delta$), and each interval is mapped to a discrete quantization level. This means that the step size between consecutive quantization levels is constant ($y_k = \frac{a_{k+1} + a_k}{2}$). They are particularly well-suited for signals that are evenly distributed across the input range. For signals with this characteristic, uniform scalar quantizers can achieve good performance with a relatively simple design.

3.1.2.1 Theoretical lower bound for Uniformly distributed signals

Uniform Scalar Quantizers perform particularly well with uniformly distributed signals. For this type of inputs, it approaches the theoretical lower bound, which is $MSE = variance * 2^{-2R}$.

3.1.3 Uniform Scalar Quantizer in MATLAB

```
% Mid-rise quantizer
delta = (max(x) - min(x)) / M;
x_tilde = delta * floor(x/delta) + delta/2;
x_tilde(x_tilde > delta/2 * (M-1)) = delta/2 * (M-1);
x_tilde(x_tilde < -delta/2 * (M-1)) = -delta/2 * (M-1);
```

```
% Mid-tread quantizer.
delta = (max(x) - min(x)) / M;
x_tilde = delta * floor((x + delta/2) / delta);
x_tilde(x_tilde > delta/2 * (M-1)) = delta/2 * (M-1);
x_tilde(x_tilde < -delta/2 * (M-1)) = -delta/2 * (M-1);
```

3.1.4 Optimum Uniform Scalar Quantizer

The optimum uniform quantizer is a type of uniform scalar quantizer that is designed to minimize the MSE for a given source distribution and a fixed number of quantization levels. It achieves the best possible performance, in terms of minimizing the quantization error, within the constraints of uniform quantization.

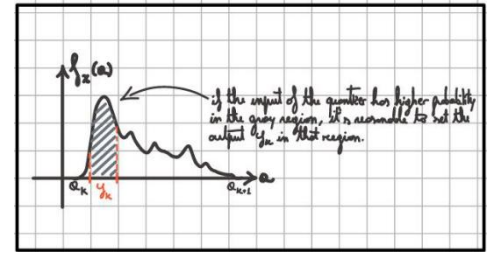


Figure 5

An Optimum Uniform Quantizer is designed in the following way:

1. We chose the number (and consequently the width) of the decision regions I_k .
2. We chose as output values (codebook) the expected value for each decision region: $y_k = E(x|x \in I_k)$.

3.1.5 Optimum Uniform Scalar Quantizer in MATLAB

```
% Optimum Uniform Scalar Quantizer
delta = (max(X)-min(X))/M;
% Define the thresholds of the decision regions.
thresh = [-inf, -delta*(M/2 - 1) : delta : delta*(M/2 - 1), inf];
% Assign to each decision region its correspondent expected value.
for j = 1:M
    Y_index = (X>thresh(j)) & (X<=thresh(j+1));
    Y_vec(Y_index) = mean(X(Y_index));
end
```

3.1.6 Lloyd—Max Quantizer

The Lloyd-Max quantizer is an iterative algorithm designed to find the optimum scalar quantization levels and decision boundaries for a given probability density function (PDF) of a signal source, with the objective of minimizing the MSE between the original and quantized signals.

The design procedure is the following:

1. Choose an initial set of values for the reproduction levels y_k .
2. Optimize the partition for the given levels, which gives the thresholds I_k .
3. Optimize the set of reproduction levels for the partition found above: $y_k = E(x|x \in I_k)$.
4. Repeat steps 2 and 3 until convergence is achieved.

Optimums Quantizer vs Lloyd Quantizer:

- **Adaptability:** The Lloyd quantizer is more adaptable to various signal distributions due to its optimization process, whereas the optimum uniform quantizer is generally better suited for signals with uniform distributions.
- **Implementation Complexity:** The Lloyd quantizer is more complex to implement and requires more computational resources, making the optimum uniform quantizer a preferable choice for simpler applications or where computational efficiency is crucial.
- **Optimization for Specific Distributions:** The main strength of the Lloyd quantizer lies in its ability to minimize quantization error for a given signal distribution, while the optimum uniform quantizer's strength is in its simplicity and effectiveness for uniformly distributed signals.

3.1.7 Lloyd-Max Quantizer in MATLAB

```
% Lloyd-Max quantizer
[thresh, codebook] = lloyds(X,M);
[~, Y, MSE] = quantiz(X, thresh, codebook);
```

3.1.8 Entropy-Constrained Scalar Quantizers (ECSQ)

Entropy-Constrained Scalar Quantizers (ECSQ) are a class of quantization techniques that incorporate entropy minimization into the design of scalar quantizers. Unlike traditional quantization methods that primarily focus on minimizing distortion, such as mean squared error, ECSQ aims to achieve a balance between two critical aspects: minimizing distortion and minimizing the bit rate required to encode the quantized values (see [rate-distortion curve](#)).

For a ECSQ we have that $R \cong H(Y) \leq \log_2(M)$, where Y are the quantized output symbols and M is the number of levels (and so $\log_2(M)$ is the bit rate).

3.1.9 Vector Quantizer

Vector Quantization (VQ) is a technique used in signal processing and data compression that quantizes vectors of data rather than scalar values, as in scalar quantization. In VQ, the input data, which can be multi-dimensional, is partitioned into blocks or vectors. Each vector is then mapped to the nearest vector in a predefined codebook.

A vector quantizer is defined by:

- A **set of decision regions** I_k with $k = 1, 2, \dots, n$.
- A **set of reproduction vectors** y_k with $k = 1, 2, \dots, n$. Each y_n vector is the centroid of the decision region I_n .
- The **quantization rule**: $q(x) = y_k$.

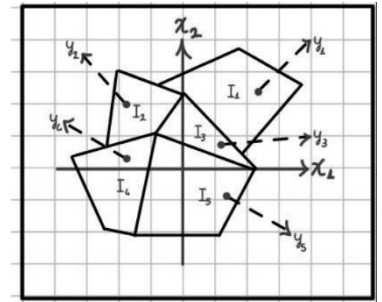


Figure 6

An optimal vector quantization satisfies the following conditions:

- Nearest neighbours: given the set of codevectors y_k , the optimal partition of the space R^N is the one that minimize the distortion.
- Centroid condition: given the partitions I_k , the codevectors must be the centroids of the decision regions.

3.1.9.1 Linde-Buzo-Gray algorithm

The most popular algorithm for vector quantization is the Linde-Buzo-Gray algorithm, which is similar to the Lloyd-Max quantization algorithm:

1. Chose an initial codebook of reconstruction vectors y_k .
2. Optimize the partition for the given codebook, which gives the regions I_k .
3. Optimize the codebook for the partition found in the previous point.
4. Repeat steps 2 and 3 until convergence is achieved.

3.1.10 Vector Quantizer in MATLAB

```

%% 2D Vector quantizer with Cb = y1=(1,2); y2=(1,4); y3=(-1,2); y4=(0,-2) .
codebook = [1,1,-1,0;
            2,4,2,-2];
% Quantize the following sequence.
x = [-4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
X_matrix = [x(1:2:end);x(2:2:end)];
Y = zeros(size(X_matrix));
for i = 1:length(X_matrix(1,:))
    % get the current column vector.
    currVector=X_matrix(:, i);
    %calculate the minimum euclidian distance and extract the corrispondent
    index
    [minDist, idxMinDist] = min(sqrt(sum((currVector-codebook).^2,1)));
    % Give in output the value correspondent to the point of the cb with
    the smaller euclidian distance.
    Y(:,i) = codebook(:,idxMinDist);
end

%% Plot: optimal assignment regions.
plot(codebook(1,:), codebook(2,:), 'x');
voronoi(codebook(1,:), codebook(2,:));
title('Optimal Assignment Regions.');
```

3.2 COMPANDER-EXPANDER

A compander (compressor-expander) is a device or algorithm used in signal processing that combines the functions of compression and expansion. It's designed to improve the dynamic range of audio signals or other types of data. The process involves three main stages:

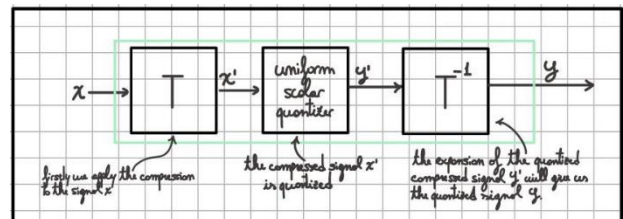


Figure 7

- **Compressor:** The compressor part of a compander reduces the dynamic range of the input signal. It does this by compressing the amplitude of the signal's louder parts while leaving the softer parts relatively unchanged. This is typically achieved by applying a nonlinear mapping to the signal's amplitude, where the mapping function has a greater slope for lower amplitude values and a lesser slope for higher amplitude values. The main goal of compression is to make the signal's loud and soft parts closer in volume, which can help in reducing the effects of noise and variations in signal strength during recording or transmission.
- **Quantization:** the compressed signal is quantized.
- **Expander:** The expander part of a compander works on the compressed signal to restore its original dynamic range, effectively expanding the amplitude of the signal's components. When the compressed signal is received or retrieved, the expander applies an inverse mapping to the amplitude, such that the original dynamic variations of the signal are reconstructed. The expansion process increases the dynamic range of the signal, making the loud parts louder and the soft parts softer, thus counteracting the compression effect.

The transformation function T that we use in the exercises is the **cumulative function**: $x' = T\{x\} = \int_{-\infty}^x p_X(a) * da$.

The benefit of the compander/expander operation is that it helps in mitigating the perceptual impact of quantization noise. By compressing the input signal before quantization, the quantization levels can be allocated more efficiently, reducing the quantization noise in critical regions of the signal.

3.3 RATE-DISTORTION THEOREM

The Rate-Distortion Theorem is a fundamental result in information theory, established by Claude Shannon, which essentially defines the limits of compression for a given level of distortion, or conversely, the minimum distortion achievable for a given compression rate.

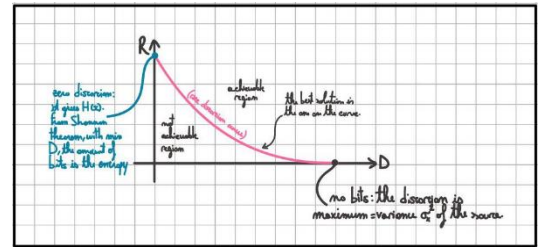


Figure 8

3.3.1 Key Concepts of the Rate-Distortion Theorem

Rate (R): This refers to the average number of bits per symbol used to represent the compressed signal. It's equal to $\log_2(M)$ where M is the number of levels of our quantizer.

Distortion (D): Distortion measures the difference between the original signal and the reconstructed signal after compression and decompression. It quantifies the loss of quality due to compression, often assessed by a distortion measure such as MSE.

Rate-Distortion Function ($R(D)$): This function represents the minimum rate R at which the source can be compressed without exceeding a specified maximum distortion level D .

3.3.2 Implications of the Theorem

Compression Limits: It establishes the fundamental limit on how much a signal can be compressed while ensuring that the distortion between the original and the compressed signal does not exceed a certain threshold. This limit is quantified by the rate-distortion function $R(D)$.

Optimal Compression: The theorem implies that for any given level of distortion, there is a lower bound on the compression rate below which it is not possible to compress the signal without exceeding the allowable distortion. Conversely, for a given rate, there is a minimum distortion that can be achieved.

Trade-off Between Rate and Distortion: The theorem highlights a trade-off between the compression rate and the quality of the reconstructed signal. Reducing the compression rate (i.e., using more bits) can decrease the distortion, while increasing the rate (i.e., compressing more aggressively) leads to higher distortion.

3.3.3 Useful formulas

For a random variable, the rate distortion curve is lower bounded by the function $R_L = \frac{1}{2} * \log_2(2 * \pi * e * D)$.

Shannon-Lower-Bound: For a Gaussian, the distortion rate function can be simplified as follows: $D_L = \text{variance} * 2^{-2R}$. The distortion-rate curve of a Lloyd quantizer applied on a Gaussian-like source is very similar to the Shannon Lower Bound.

3.4 QUESTIONS FROM PAST EXAMS

3.4.1 July, 12 2016

Consider a continuous source X that emits symbols following a PDF $p_X(x)$.

- a) Explain how to design the Lloyd-Max quantizer. Consider $M = 3$ quantization bins and use the thresholds $\{-\infty, a_2, a_3, +\infty\}$ as initialization.

A Lloyd max quantizer is a type of linear quantizer that estimates the statistical distribution of the source to improve the quantization quality (minimize the MSE). The design process is the following:

1. We start with a guess of the possible thresholds $\{-\infty, a_2, a_3, +\infty\}$.
2. We calculate the y_k output values of the correspondent thresholds as the expected value of the input in that specific decision region.
3. Given the new y_k , we re-calculate the thresholds as the middle point between the y_k values ($I_k = \frac{y_k + y_{k+1}}{2}$).
4. We repeat points 2 and 3 until the difference between values of I_k of two consecutive iterations is smaller than a predefined threshold, meaning that the algorithm reached convergence.

- b) Consider this series of symbols emitted by X: $x = \{-0.9, 0.1, -0.8, -0.7, 0.7, -0.6, -0.2, 0.1, 0.7, -0.2, 1.0\}$. Design a Lloyd-Max quantizer tailored to these symbols using $M = 3$ bins and the thresholds $\{-\infty, -0.5, 0.5, +\infty\}$ as initialization. Stop after one complete iteration.

- c) How many iterations must be performed to reach the algorithm's convergence?

1. Iteration 1:

- We have three initial decisions $\{-\infty, -0.5, 0.5, +\infty\}$.
- We calculate the output value for each decision regions:
 - $\{-\infty, -0.5\}$ in which we have the values $\{-0.9, -0.8, -0.7, -0.6\}$. The codebook value for this region is $y_1 = \frac{1}{4} * (-0.9 - 0.8 - 0.7 - 0.6) = -0.75$.
 - $\{-0.5, 0.5\}$ in which we have the values $\{0.1, -0.2, 0.1, -0.2\}$. The output value correspondent to this region is $y_2 = \frac{1}{2} * (0.1 - 0.2) = -0.05$.
 - $\{0.5, +\infty\}$ in which falls the values $\{0.7, 0.7, 1.0\}$, therefore the output is $y_3 = \frac{2}{3} * 0.7 + \frac{1}{3} * 1 = 0.8$.

- c. Now that we have the new y_k values, we calculate the new $I_k = \frac{y_k + y_{k+1}}{2}$:

$$I_1 = \frac{-\infty - 0.75}{2} = -\infty; I_2 = \frac{-0.75 - 0.05}{2} = -0.4; I_3 = \frac{-0.05 + 0.8}{2} = 0.375; I_4 = \frac{0.8 + \infty}{2} = +\infty.$$

2. Iteration 2:

- We repeat point 2 to re-optimize the output values:
 - $\{-\infty, -0.4\}$ in which we have the values $\{-0.9, -0.8, -0.7, -0.6\}$. The codebook value for this region is $y_1 = \frac{1}{4} * (-0.9 - 0.8 - 0.7 - 0.6) = -0.75$.
 - $\{-0.4, 0.375\}$ in which we have the values $\{-0.2, 0.1, -0.2, 1.0\}$. The output value correspondent to this region is $y_2 = \frac{1}{2} * (0.1 - 0.2) = -0.05$.
 - $\{0.375, +\infty\}$ in which falls the values $\{0.7, 0.7, 1.0\}$, therefore the output is $y_3 = \frac{2}{3} * 0.7 + \frac{1}{3} * 1 = 0.8$.
- The y_k are exactly the same of the previous iteration, so the algorithm reached convergence.

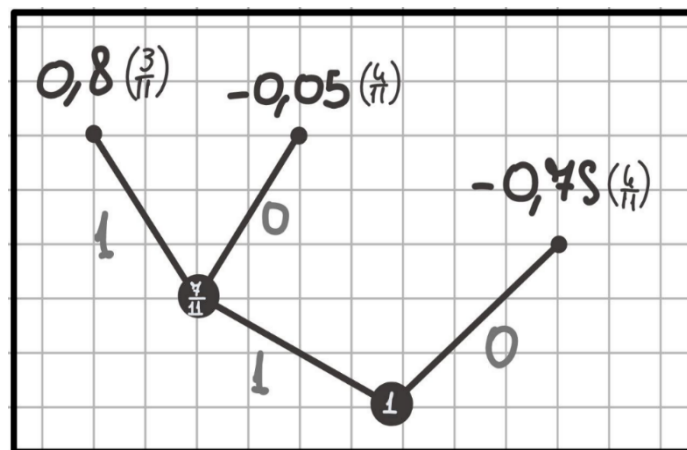
- d) Quantize x using the Lloyd-Max quantizer designed at point (b), thus obtaining $\tilde{x}$.

To quantize the sequence of symbols $x = \{-0.9, 0.1, -0.8, -0.7, 0.7, -0.6, -0.2, 0.1, 0.7, -0.2, 1.0\}$ we must substitute each symbol with the correspondent y_k of the decision region in which it falls. If we do so, we obtain: $\tilde{x} = \{-0.75, -0.05, -0.75, -0.75, 0.8, -0.75, -0.05, -0.05, 0.8, -0.05, 0.8\}$.

- e) Compute the entropy of the source representing $\tilde{x}$ estimating the PMF directly from $\tilde{x}$.

The PMF of X is characterized by these probabilities: $p(-0.75) = \frac{4}{11}$; $p(-0.05) = \frac{4}{11}$; $p(0.8) = \frac{3}{11}$. The value of the entropy of $\tilde{x}$ is: $H(\tilde{x}) = -\frac{4}{11} * \log_2\left(\frac{4}{11}\right) - \frac{4}{11} * \log_2\left(\frac{5}{11}\right) - \frac{3}{11} * \log_2\left(\frac{2}{11}\right) = 1.57 \text{ bits}$.

- f) Derive a Huffman code to transmit $\tilde{x}$. Represent using this code a string of four symbols of your choice.



$$0.8 = 11; -0.05 = 10; -0.75 = 0.$$

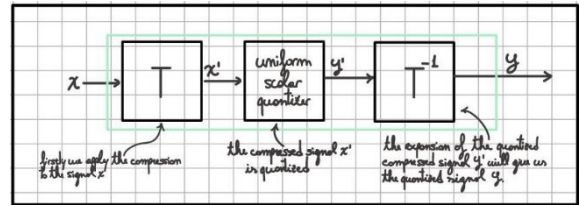
$$\tilde{x} = \{0.10.0.0.11.0.10.10.11.10.11\}$$

3.4.2 September, 26 2012

Consider a continuous source X , whose PDF is given by $p_X(x) = e^{-x}$, $x \geq 0$.

- a) Design a quantizer based on compander-expander. Use 1 *bits* to represent each symbol. Show the equivalent non-uniform quantizer in the original signal domain.

A compander expander is structured as showed in the picture on the right. This means that we need to do three steps:



1. We apply the transformation (compression) on the source original output x :

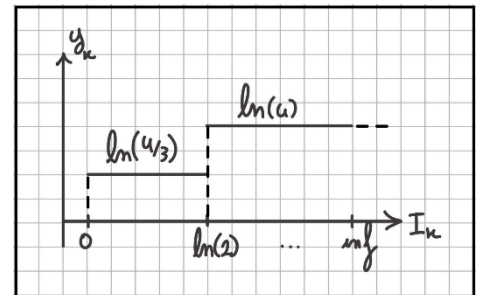
$$x' = T\{x\} = \int_{-\infty}^x p_X(a) * da = \int_0^x e^{-a} = 1 - e^{-x}.$$

2. We quantize this continuous input into a discrete set of symbols (1 *bit*: $\text{codebook} = \{0,1\}$):

The input of the quantizer x' goes from 0 (for $x = 0$) to 1 (for $x = +\infty$), so we can use a uniform scalar quantizer putting the decision threshold at 0.5 and sending as outputs $y'_1 = \frac{0+0.5}{2} = 0.25$ when the input is below that threshold and $y'_2 = \frac{0.5+1}{2} = 0.75$ otherwise.

3. Apply the inverse transformation ($x' = 1 - e^{-x}$) $^{-1} \rightarrow x = T^{-1}\{x'\} = -\ln(1 - x') = \ln\left(\frac{1}{1-x'}\right)$ to y' obtaining:

- Output values:
 - $T^{-1}\{0.75\} = \ln(4)$;
 - $T^{-1}\{0.25\} = \ln\left(\frac{4}{3}\right)$.
- Output thresholds:
 - $T^{-1}\{0\} = 0$;
 - $T^{-1}\{0.5\} = \ln(2)$; $T^{-1}\{1\} = \text{inf}$.



- b) Compute the entropy of the output symbols of the quantizer.

Firstly, we need to calculate the PMF of the final output y :

- $p\left(\ln\left(\frac{4}{3}\right)\right) = \int_0^{\ln(2)} e^{-x} dx = 1 - e^{-\ln(2)} = 1 - e^{\ln(\frac{1}{2})} = 1 - \frac{1}{2} = \frac{1}{2}$;
- $p(\ln(4)) = \int_{\ln(2)}^{\text{inf}} e^{-x} dx = 0 + e^{-\ln(2)} = e^{\ln(\frac{1}{2})} = \frac{1}{2}$.

Then we can calculate the entropy as: $H(y) = -2 * \left[\frac{1}{2} * \log_2\left(\frac{1}{2}\right)\right] = 1 \frac{\text{bit}}{\text{symbol}}$.

4 LOSSY CODING: PREDICTIVE CODING

4.1 DIFFERENTIAL PULSE CODE MODULATION (DPCM)

Differential Pulse Code Modulation (DPCM) is a signal encoding scheme used to minimize the bit rate for the transmission or storage of digital audio, images, or video. It works by encoding the difference between successive samples rather than encoding each sample independently. This method exploits the fact that many successive samples are similar to each other, allowing for more efficient data compression by reducing redundancy.

4.1.1 Open Loop DPCM

Encoder: In an open loop DPCM system, the encoder computes the difference between the current sample and its predicted value. The predicted value is typically generated from previous samples using a linear prediction filter. The difference, also known as the prediction error, is then quantized and encoded for transmission or storage. This process does not involve any feedback from the decoder to adjust the prediction model.

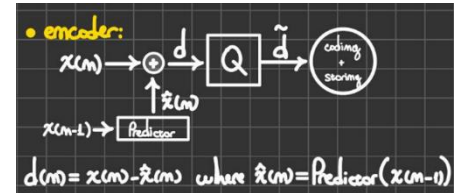


Figure 9

Here's a step-by-step explanation of the encoder in an open loop DPCM system:

1. **Prediction ($\hat{x}_n$):** The first step in the DPCM encoder is to predict the current sample value based on previous sample values. This prediction is usually made using a linear predictor that forms a prediction equation. *The prediction equation typically uses a few of the most recent past samples of x_n to estimate the current sample value.* The accuracy of this prediction greatly influences the efficiency of the DPCM system.
2. **Difference Calculation (d):** Once the prediction for the current sample is made, the encoder calculates the difference between the actual current sample value and its predicted value. This difference is known as the *prediction error or differential signal*. Since adjacent samples in a signal are often similar, this differential signal will generally have a smaller amplitude than the original signal values, leading to potential savings in bit rate after quantization.
3. **Quantization ($\tilde{d}$):** The next step is to quantize the prediction error. The quantizer's resolution (the number of bits used) directly affects the quality of the reconstructed signal: fewer bits may lead to significant quantization noise, while more bits reduce this noise but require a higher bit rate.
4. **Encoding (es. Huffman):** After quantization, the quantized prediction errors are encoded into a digital bitstream. This encoding may use fixed-length coding, but more efficient encoders use variable-length coding (such as Huffman coding) to assign shorter codes to more frequent values and longer codes to less frequent values, further reducing the average bit rate.
5. **Transmission or Storage:** The encoded bitstream is then ready for transmission over a communication channel or for storage. Since the bitstream now represents differential values rather than absolute sample values, it typically requires fewer bits for the same level of signal fidelity, assuming the signal has a high correlation between successive samples.

Decoder: The decoder works by receiving the quantized difference values and reconstructing the predicted values using the same prediction model as the encoder. It adds the quantized difference to the predicted value to reconstruct the original signal. Since there is no feedback loop, the decoder's prediction model must match the encoder's model exactly. Here's a step-by-step explanation of how the decoder in an open loop DPCM system works:

1. **Decoding the Quantized Differences (es. From Huffman, $\tilde{d}$):** The first step in the decoding process involves receiving the encoded bitstream, which represents the quantized differences (or prediction errors) between actual sample values and their predicted values. The decoder decodes this bitstream to recover the quantized differences. This step essentially reverses the encoding process, converting the digital codes back into the quantized values of the prediction error.

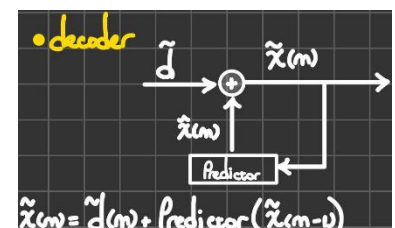


Figure 10

2. **Reconstruction of the Predicted Sample ($\hat{x}_n$):** The decoder uses the *same prediction model as the encoder* to generate predicted values for each sample. For the very first sample (or for a predefined starting point in the stream), a fixed initial value or a specially encoded value is used because there are no previous samples to base the prediction on. For subsequent samples, *the decoder uses the reconstructed values of previous samples to predict the next sample value*.
3. **Adding the Quantized Difference ($\tilde{x}_n$):** After predicting the next sample value, the decoder adds the quantized difference ($\tilde{d}$) to the predicted value $\hat{x}_n$. This step compensates for the prediction error by adjusting the predicted value to match the original sample value more closely. The result of this addition is the reconstructed sample value $\tilde{x}_n$.

4.1.2 Closed Loop DPCM

Encoder: In a closed loop DPCM system, the encoder includes a feedback loop from the quantizer output back to the input of the prediction model. This allows the encoder to adjust its prediction based on the quantized values, which more closely represents the decoder's process of reconstructing the signal. This feedback mechanism can improve the accuracy of the predictions and reduce the quantization error, leading to higher quality reconstructed signals. Here's how the encoder in a closed loop DPCM system works, step by step:

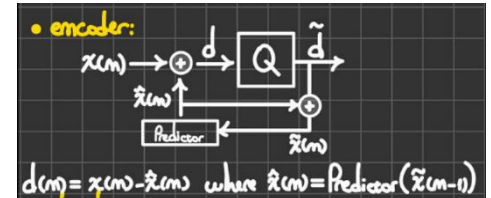


Figure 11

1. **Prediction ($\hat{x}_n$):** Similar to open loop DPCM, the process begins with predicting the current sample's value. The prediction is typically based on previous output samples. However, in closed loop DPCM, this prediction can be adjusted based on feedback from later steps. This is what distinguishes closed loop from open loop DPCM: the quantized error $\tilde{d}$ is summed up to the predicted sample $\hat{x}_n$ and the result $\tilde{x}_n$ is fed back to adjust the prediction model or algorithm. This feedback mechanism allows the encoder to *more closely mimic the decoder's process*, potentially leading to more accurate predictions.
2. **Difference Calculation (d):** The encoder calculates the difference between the actual sample and its predicted value. This difference or error represents the new information that needs to be encoded and transmitted.
3. **Quantization ($\tilde{d}$):** The prediction error is then quantized. Quantization reduces the precision of the prediction error to a limited set of values, which is essential for reducing the bit rate. The quantization step size can significantly affect the system's performance, with finer quantization offering better accuracy but at the cost of a higher bit rate.
4. **Encoding and Transmission (Huffman):** The quantized differences are then encoded into a digital format for transmission or storage. As with open loop DPCM, the encoding process may involve variable-length coding to optimize the bit rate further.

Decoder: The decoder in a closed loop system operates similarly to the open loop system. However, because the encoder's prediction model is adjusted based on the quantized output, the decoder can more accurately reconstruct the original signal by following the same prediction and addition process.

For a more detailed explanation, read the Open Loop DPCM decoder paragraph.

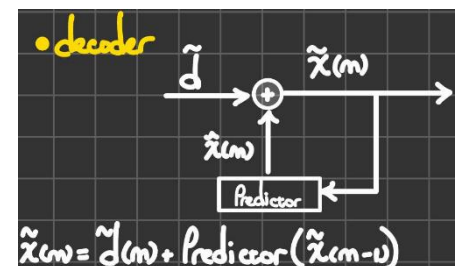


Figure 12

4.1.3 Open Loop vs Closed Loop DPCM

Open loop and closed loop Differential Pulse Code Modulation (DPCM)

systems each offer distinct advantages and face specific disadvantages, tailored to different application needs.

Open loop DPCM is simpler and more straightforward, benefiting applications where low computational complexity and predictability are essential. Its design, free of feedback mechanisms, ensures lower latency, making it ideal for real-time applications. However, this simplicity comes at the cost of adaptability and efficiency, particularly for complex signals, and it is more susceptible to error propagation without a means to correct or adjust based on the signal's changing characteristics.

Conversely, closed loop DPCM introduces a feedback mechanism to adapt the prediction model based on the quantized output, leading to more accurate predictions and reduced quantization errors. This adaptability results in better signal quality and compression efficiency, especially beneficial for signals with varying characteristics. However, the complexity of implementing a feedback loop increases the system's computational requirements and can introduce additional latency, which might be a drawback for time-sensitive applications. Moreover, the feedback loop's design must be carefully managed to avoid instability and ensure the system's robustness.

4.1.4 Open Loop vs Closed Loop DPCM (Matlab)

```
%% OPEN LOOP DPCM
x_tilde=zeros(size(x));
% Encode first sample with PCM.
Delta = (max(x) - min(x)) / M;
x_tilde(1) = delta * floor(x(1)/delta) + delta/2;
% Encode other samples using DPCM
% Calculate prediction residual
d = [NaN; z(2:end)]; % Random signal component
% Quantize prediction residual with PCM:
Delta = (max(d) - min(d)) / M;
d_tilde = Delta * floor(d/Delta) + Delta/2;
% Apply DPCM reconstruction to succ. samples
for j = 2:length(d_tilde)
    x_hat = p*x_tilde(j-1);
    % Get the reconstructed signal:
    x_tilde(j) = d_tilde(j)+x_hat;
end
```

```
%% CLOSED LOOP DPCM
x_tilde = zeros(size(x));
M=Mvec(i);
% Encode first sample using PCM
delta = (max(x) - min(x)) / M;
x_tilde(1) = delta*floor(x(1)/delta)+delta/2;
% Encode other samples using DPCM:
% Use same quantizer (Delta) of Open Loop DPCM
d_OL = x - x_hat_OL;
Delta = (max(d_OL) - min(d_OL)) / M;
for j = 2 : length(x)
    x_hat = p * x_tilde(j-1); % Predicted sample
    d = x(j) - x_hat; % prediction residual
    % quantized prediction residual:
    d_tilde = Delta * floor(d/Delta) + Delta/2;
    % reconstructed signal (booth ENCODER & DECODER):
    x_tilde(j) = d_tilde + x_hat;
end
```

4.2 QUESTIONS FROM PAST EXAMS

4.2.1 September, 12 2016

Consider the following sequence of samples: $x = \{10, 12, 15, 16, 20, 19, 18, 15, 13\}$.

a) Obtain the sequence $\tilde{x}$ encoding x using PCM with the following quantizer:

- $I_1 = [8, 12), y_1 = 10$
- $I_2 = [12, 16), y_2 = 14$
- $I_3 = [16, 20), y_3 = 18$
- $I_4 = [20, 24), y_4 = 22$

It is sufficient to substitute each symbol of the sequence with its correspondent quantized symbol:

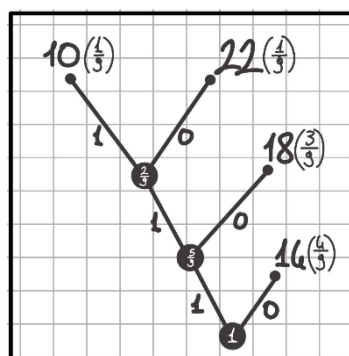
$$\tilde{x} = \{10, 14, 14, 18, 22, 18, 18, 14, 14\}$$

a) Compute the mean squared error (MSE) introduced by the use of PCM.

Firstly, we calculate the error $e = \{0, 2, 1, 2, 2, 1, 0, 1, 1\}$. Then we get the $MSE = \text{mean}(e^2)$:

$$MSE = \frac{0 + 4 + 1 + 4 + 4 + 1 + 0 + 1 + 1}{9} = \frac{16}{9} = 1,77$$

b) Estimate the pmf of each symbol of $\tilde{x}$ and derive the Huffman coding scheme.



c) Obtain the sequence $\hat{x}$ encoding x using a closed-loop DPCM with the following quantizer:

- $I1 = [-\infty, -1.5), y1 = -3$
- $I2 = [-1.5, 1.5), y2 = 0$
- $I3 = [1.5, +\infty), y3 = 3$

Consider that the first symbol is already known by the decoder.

In order to emulate the working of a DPCM encoder-decoder, we can write the following table:

$\hat{x}$	10	10	13	16	16	19	19	19	16
x	10	12	15	16	20	19	18	15	13
d	0	2	2	0	4	0	-1	-4	-3
$\tilde{d}$	0	3	3	0	3	0	0	-3	-3
$\tilde{x}$	10	13	16	16	19	19	19	16	13

d) Compute the mean squared error (MSE) introduced by the use of DPCM.

The total MSE of the DPCM is equal to the MSE introduced by the quantizer:

$$e = \{0, 1, 1, 0, 1, 0, 1, 1, 0\} \rightarrow MSE = \frac{1 + 1 + 1 + 1 + 1}{9} = \frac{5}{9} = 0.5\bar{5}$$

4.2.2 January, 15 2021

Consider a continuous source X that emits the following series of symbols $x = \{3, 0, 6, 6, 6, 3\}$. You are given a quantizer $Q(\cdot)$ that applies the following quantization rule $Q(x) = \left\lceil \frac{x}{\Delta} \right\rceil * \Delta$, where $\lceil \cdot \rceil$ rounds to the least integer greater than or equal to its argument, and $\Delta = 2$ is the quantization step.

a) Compute the sequence $\hat{x}_{PCM}$ obtained by encoding and decoding x using a PCM scheme. Clearly specify which is the encoded and which is the decoded signal.

The encoded signal is the series of codewords that represents $\hat{x}_{PCM} = \{4, 0, 6, 6, 6, 4\}$, which is equal to the decoded signal.

b) Compute the sequence $\hat{x}_{OL-DPCM}$ obtained by encoding and decoding x using an open-loop DPCM scheme. The predictor acts by subtracting half of the previous sample (i.e., $\tilde{x}(n) = 0.5 \cdot x(n-1)$). Use the same quantizer defined above and consider that the first sample is encoded using PCM. Clearly specify which is the encoded and which is the decoded signal.

To emulate the behaviour of the encoder-decoder OL-DPCM we can use the following table:

the predictor of the encoder takes as input the previous sample of x						
$\hat{x}_{enc}$	/	15	0	3	3	3
x	3	0	6	6	6	3
d	/	-15	6	3	3	0
$\tilde{d}$	/	0	6	6	6	0
$\hat{x}_{dec}$	/	2	1	3.5	3.75	3.875
$\tilde{x}$	6	2	7	7.5	7.75	3.875
the predictor of the decoder takes as input the previous sample of the decoded signal $\tilde{x}$						

The encoded signal is $\tilde{d}$ while the decoded signal is $\tilde{x}$.

- c) Compute the sequence $\hat{x}_{CL-DPCM}$ obtained by encoding and decoding x using a closed-loop DPCM scheme. Use the same prediction rule, quantizer and strategy for the first sample applied before. Clearly specify which is the encoded and which is the decoded signal.

in a Closed Loop DPCM the predictor is the same for the encoder and for the decoder

$\hat{x}$	/	2	0	3	3.5	3.75
x	3	0	6	6	6	3
d	/	-2	6	3	2.5	-0.75
$\tilde{d}$	/	-2	6	4	4	0
$\tilde{x}$	4	0	6	7	7.5	3.75

$\tilde{x} = \tilde{d} + \hat{x}$

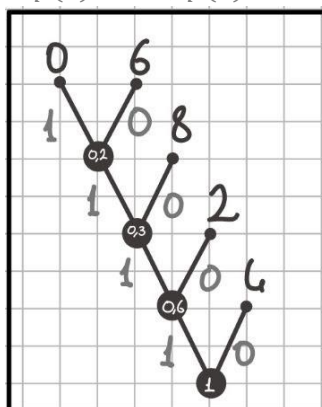
Also in this case, the encoded signal is $\tilde{d}$ and the decoded one is $\tilde{x}$.

- d) Comment on the role of the first sample of the sequence for DPCM schemes. How can you typically deal with it and why?

In both open-loop and closed-loop DPCM schemes, the first sample plays a crucial role. It serves as the reference point from which all subsequent predictions and adjustments are made. Since DPCM encodes the difference between the predicted and actual values, without the first reference sample, the decoder would not have a starting point to reconstruct the original signal.

The first sample is usually encoded using a simple Pulse Code Modulation (PCM) scheme, as we have done in parts (b) and (c). This is because there is no previous value to use to predict the first sample. The first sample needs to be accurately represented because all subsequent samples are predicted based on the previous samples. If the first sample is inaccurately encoded, it could lead to a propagation of errors through the rest of the decoded sequence.

- e) Consider applying lossless coding to $\hat{x}_{PCM}$. Propose a prefix-code, and write the code associated to the sequence of symbols $\hat{x}_{PCM}$. Compute the bitrate in bit per symbol. Consider the following probability mass function $p(0) = 0.1$; $p(2) = 0.3$; $p(4) = 0.4$; $p(6) = 0.1$; $p(8) = 0.1$.



$$\text{Exp. Length} = 1 \cdot 0.1 + 2 \cdot 0.3 + 3 \cdot 0.1 + 2 \cdot 0.4 = 2.1 \text{ bit/symbol}$$

We can use this Huffman diagram to get a code associated to the $\hat{x}_{PCM} = \{0, 1111, 1110, 1110, 1110, 0\}$.

- f) Using MSE as distortion metric and entropy as rate measure, draw the rate-distortion working point of each one of the three coding techniques just applied (i.e., PCM, closed-loop DPCM, open-loop DPCM).

5 LOSSY CODING: TRANSFORM CODING

5.1 TRANSFORM CODING GENERAL CONCEPT

Transform coding is a type of data compression technique used in the field of lossy coding. It works by transforming the original data into a format where the components are decorrelated and then quantizing the transformed data. The transformation aims to concentrate the signal's energy into a few significant components, making it possible to achieve higher compression ratios by quantizing or even discarding the less significant components without significantly affecting the perceived quality of the reconstructed data.

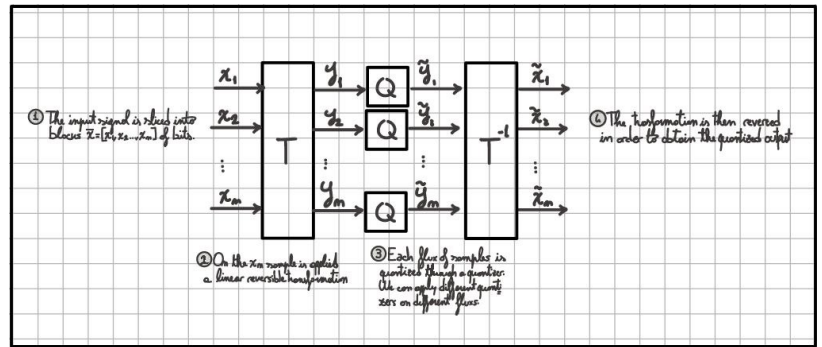


Figure 13

The process typically involves the following steps:

- **Transformation ($x_n \rightarrow y_n$):** The original data (for example, an image or audio signal) is transformed into a different domain (such as the frequency domain) using a mathematical transform such as the Discrete Cosine Transform (DCT), the Fourier Transform, or the Wavelet Transform. This step aims to represent the data in a form where its components are less correlated than in the original domain.
- **Quantization ($y_n \rightarrow \tilde{y}_n$):** After transformation, the coefficients in the transformed domain are quantized. This step reduces the precision of the components based on their importance to the overall signal quality. Components that have less impact on the perceived quality are quantized more coarsely or even discarded.
- **Encoding (sender) and decoding (receiver):** The quantized values $\tilde{y}_n$ are then encoded by the sender using a coding scheme that exploits the statistical properties of these values to reduce the amount of data further. Techniques such as Huffman coding or arithmetic coding are often used. The receiver will retrieve the quantized transformed signal $\tilde{y}_n$ from the encoded sequence of bits.
- **Inverse transformation ($\tilde{y}_n \rightarrow \tilde{x}_n$):** The inverse transformation is applied to $\tilde{y}_n$ in order to get $\tilde{x}_n$.

The transformations used in this kind of algorithms respect the following properties:

- **Orthogonality:** the rows and the columns of the T matrix are orthogonal vectors: $T^{-1} = T^T$. It is a very useful property because we can use T^T for the anti-transformation instead of T^{-1} .
- **Distance preservation:** the length of the vectors remain the same before and after the transformation. This property is essential for the energy preservation.
- **Energy preservation:** the energy of the signal before and after the transformation is the same.

5.1.1 Geometric interpretation of Transform Coding

Transform coding can be geometrically interpreted as the process of mapping a signal from its original high-dimensional space to a lower-dimensional space where its most significant features are concentrated.

To understand this concept, let's consider a simple geometrical example using transform coding with a 2D signal, such as an image or a simple graphical representation. Imagine we have a set of points in a 2D plane that form a specific shape or pattern. In its original form, each point's position is defined by two coordinates (x, y) . This set of points could be **dense in one region and sparse in another, indicating variation in signal energy or information content across the plane**. For transform coding, we apply a transformation, like the Discrete Cosine Transform (DCT), to these points. Geometrically, this is akin to **rotating and scaling the axes of the 2D plane so that the majority of the points (or signal energy) aligns along one axis more than the other**. In this transformed space, one axis might now contain the bulk of the signal's information, while the other has significantly less. *This process can be visualized by imagining a cloud of points spread out over the original xy plane.*

After transformation, these points are repositioned so that when viewed from the perspective of the new axes, most points are close to one of these axes (say, the x axis), making the y values (now representing less critical information) smaller and more uniform.

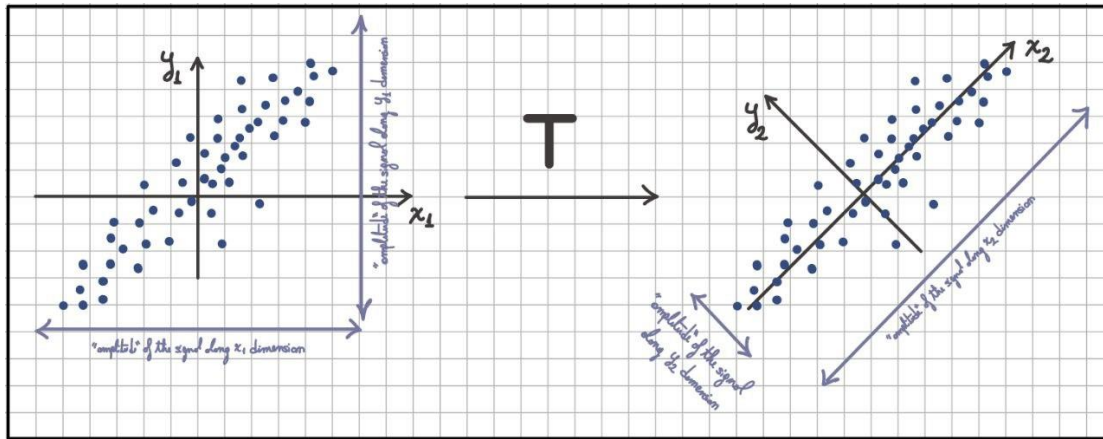


Figure 14

Quantization then comes into play, simplifying the less significant components (smaller y values) which can be represented with less precision or even discarded without substantially affecting the overall pattern's integrity when viewed from our original perspective.

In essence, the transformation has allowed us to **identify and prioritize the components of our signal that carry the most information**, enabling us to **compress the data by focusing on these components and reducing the precision of the less significant ones**. The inverse transformation would then approximate the original pattern or shape, with most of the original information intact despite the compression.

5.1.2 Bit allocation

Bit allocation in transform coding is a crucial step that determines how the available bits are distributed among the various components (or coefficients) of the transformed signal. This step directly impacts the efficiency and effectiveness of the compression and the quality of the reconstructed signal. Bit allocation is guided by the goal of optimizing the trade-off between compression ratio and signal quality, **taking into account the perceptual relevance of different components**. We can classify the coefficients in two classes:

- High-energy coefficients: they contains more information (Signal Energy) and are crucial for the quality of the reconstructed signal. They are consequently quantized at an higher bit-rate.
- Low-energy coefficients: they carry less information and may not significantly affect the perceived quality of the reconstructed signal. They are quantized with a lower bitrate or even discarded without substantially degrading the signal quality.

Practically, the coefficients are the rows of the matrix that we get after the transformation ($Y = T * X$).

We can quantize these rows with different amounts of bits, and the optimal bit allocation for a coefficient k is given

by the formula $R_k = R + \frac{1}{2} * \log_2 \left(\frac{\sigma_{y_k}^2}{\left(\prod_{i=1}^N \sigma_{y_i}^2 \right)^{\frac{1}{N}}} \right)$, in which:

- R is the average bit-rate of the quantizers (equal to $\frac{\text{total amount of bits}}{\text{number of quantizers}}$).
- $\sigma_{y_n}^2$ is the variance of the n -esimal coefficient (variance of elements of a single row in the Y matrix) and so
 - $\sigma_{y_k}^2$ is the variance of the k -esimal coefficient, the one for which we are calculating the optimal bit allocation.
 - $\left(\prod_{i=1}^N \sigma_{y_i}^2 \right)^{\frac{1}{N}}$ is the geometric mean over all the coefficients of the Y matrix.

5.1.3 MATLAB Implementation of a general Transform Coding Algorithm

```
% Define Number of Coefficients
NumOfCoef = length(T(:,1)); % T is a NxN matrix
% Define columns of the input matrix= elements per coefficient
% (Apply a floor operator to crop the signal)
ElemPerCoef = floor(length(x)/NumOfCoef);
% Reshape the input vector:
x = x(1:NumOfCoef*ElemPerCoef);
X = reshape(x,NumOfCoef,ElemPerCoef);
% Perform the transformation:
Y = T * X;
% Calculate the optimal bit allocation for each coefficient:
varPerCoef = var(Y, 0 , 2); % 2 is the second dimension (columns)
geomMeanOfVariances = prod(varPerCoef)^(1/NumOfCoef);
R_opt = round(R_avr+ 0.5*log2(varPerCoef./geomMeanOfVariances));
%% Quantize the coefficients (Y rows) using the opt.bit allocation
M_vec = 2.^R_opt;
delta_vec = (max(Y, [], 2) - min(Y, [], 2))./M_vec;
Y_tilde = delta_vec.*floor(Y./delta_vec)+delta_vec/2;
for i = 1:NumOfCoef
    y=Y_tilde(i,:);
    delta = delta_vec(i);
    M = M_vec(i);
    y(y>delta/2*(M-1)) = delta/2*(M-1);
    y(y<-delta/2*(M-1)) = -delta/2*(M-1);
    Y_tilde(i,:) = y;
end
%% Anti-transform Y_tilde
T_inverse = T'; % T' = T^-1 (Orthogonality)
X_tilde = T_inverse * Y_tilde;
x_encoded = X_tilde(:); % Flat the matrix
```

5.2 DIFFERENT TYPES OF TRANSFORMATION MATRICES

Many mathematical techniques are used in signal processing for transforming data from the time or spatial domain into another domain in which the data's inherent structure can be more efficiently represented or analyzed.

5.2.1 Karhunen-Loève Transform (KLT)

The Karhunen-Loève Transform is a method for decorrelating data or transforming it into a set of orthogonal (uncorrelated) components that best represent the variance in the data. The KLT is optimal in the sense that it minimizes the mean square error (MSE) of the compressed data, making it the best transform for signal representation in terms of energy compaction for a given number of coefficients.

There are two ways to find the KLT transformation matrix's coefficients:

- **Covariance matrix** (used in MATLAB): $C_{xx} = E[(\bar{x} - \bar{\mu}_x) * (\bar{x} - \bar{\mu}_x)^T]$ (over all the columns)
- **Autocorrelation matrix** (used in written exercises): $R_x = E[\bar{x} * \bar{x}^T]$

Where $\bar{x}$ is a column of the input matrix X and $\bar{\mu}_x$ is the average value of the signal x.

5.2.2 MATLAB implementation of KLT

The MATLAB code to get the KLT transformation matrix is the following:

```

% Reshape x to get X.
NumOfCoef = 8;
ElemPerColumns = floor(length(x)/NumOfCoef);
x = x(1:NumOfCoef*ElemPerColumns);
X = reshape(x, NumOfCoef, ElemPerColumns);
% Calculate the average of x.
avg = mean(x);
% Initialize the Cxx 'cube'.
Cxx = zeros(NumOfCoef, NumOfCoef, ElemPerColumns);
% Calculate the Cxx matrix for each column of X and save
% it into the corrispondent 3th dimension of the Cxx cube.
for i = 1 : ElemPerColumns
    Cxx(:, :, i) = (X(:, i) - avg) * (X(:, i) - avg)';
end
% "Compress" the 3th dimension of the Cxx cube:
Cxx = mean(Cxx, 3);
% Calculate the eigen vectors matrix:
[T_t, ~] = eig(Cxx);
% Get the actual transformation matrix.
T = T_t';
% Use T in transform coding.

```

It is possible to see a graphical explanation of the code in the picture below.

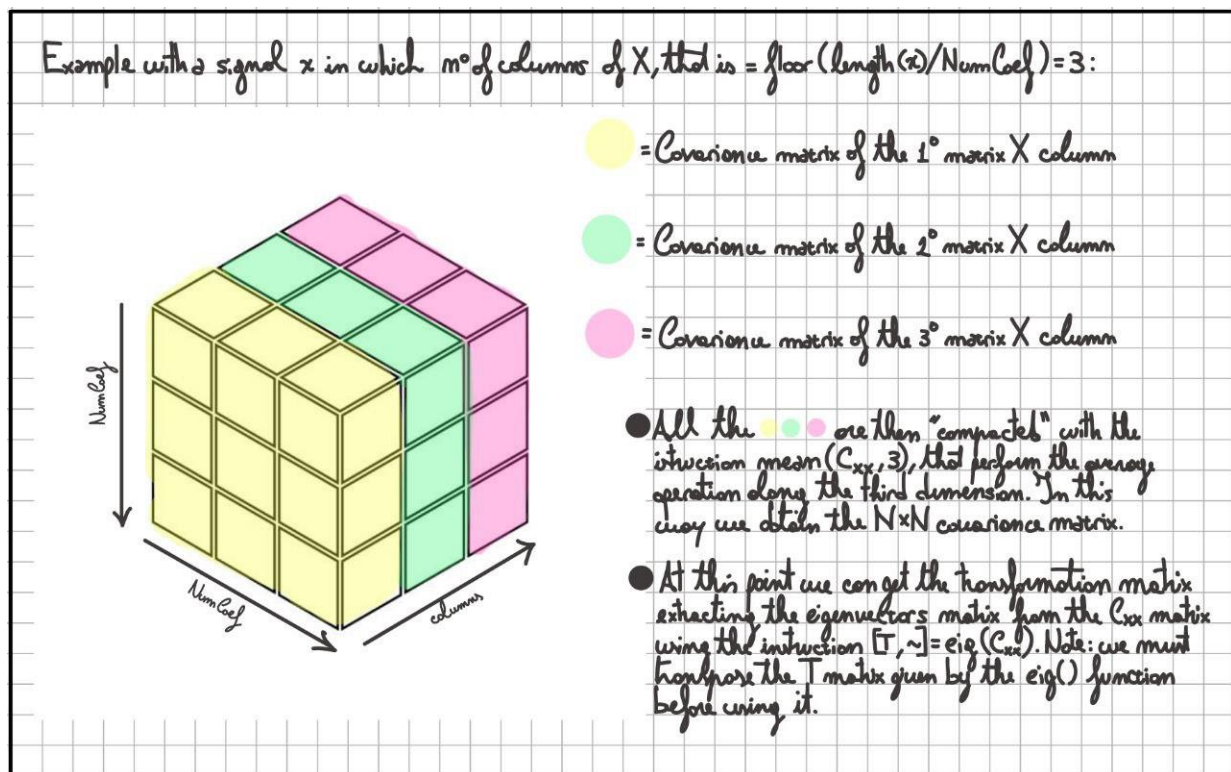


Figure 15

5.2.3 Discrete Cosine Transform (DCT)

The Discrete Cosine Transform is a technique used to transform a sequence of data points (signal or image pixels) from the spatial or temporal domain to the frequency domain. The transformation is based on cosine functions oscillating at different frequencies (similar to the Discrete Fourier Transform (DFT), but it uses only cosine functions).

One of the most valuable properties of the DCT is its ability to compact energy, meaning that for many natural signals (like images and audio), most of the signal's information tends to be concentrated in a few low-frequency

components of the DCT. This characteristic is crucial for compression, allowing high compression ratios without significantly degrading perceived quality.

In DCT, the elements t_{kl} of the transformation matrix T coefficient are calculated using the following formula:

$$t_{kl} = \begin{cases} \sqrt{\frac{1}{N}} * \cos\left(\frac{\pi}{2N} * (k-1) * (2l-1)\right) & \text{for } k = 1 \\ \sqrt{\frac{2}{N}} * \cos\left(\frac{\pi}{2N} * (k-1) * (2l-1)\right) & \text{for } k > 1 \end{cases}$$

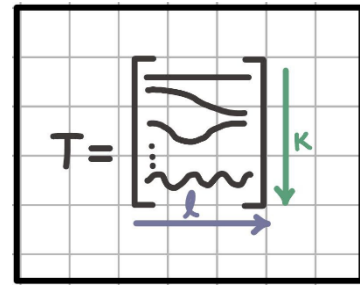


Figure 16

5.2.4 MATLAB implementation of DCT

```
N= 8; % Number of coefficients
% Define k and l
l = 1 : N; % Orizontal array
k = 1'; % Vertical array
% Define the transformation matrix
T = sqrt(1/N)*cos(pi/(2*N)*(k-1)*(2*l-1));
T(2:end, :) = sqrt(2)*T(2:end, :);
% Use the T matrix in Transform Coding
```

5.3 QUESTIONS FROM PAST EXAMS

5.3.1 January, 15 2021

(True/False section):

- The matrix $T = \begin{bmatrix} 1 & 1 & 0.5 \\ 2 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ cannot be used in a transform coding scheme (TRUE or FALSE?).

To answer this question, we must verify that this matrix is orthogonal. In order to do that we can:

- Verify that the dot product between any two different column is = 0 (es. $\begin{bmatrix} 1 & 1 & 0.5 \end{bmatrix} * \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix} = 4.5 \neq 0$).
- Verify that the determinant is = ± 1 (es. $\text{Det}(T) = 1 * (0 - 1) + 1 * (0 - 0) + 0.5 * (2 - 0) = 0 \neq \pm 1$).
- Verify that $T * T^T = I$, where I is the identity matrix (in this case $T * T^T = \begin{bmatrix} 2.25 & 4.5 & 1 \\ 4.5 & 9 & 2 \\ 1 & 2 & 1 \end{bmatrix}$).

It is sufficient that one of the previous conditions is not true to make the matrix not compatible with a transform coding scheme. So we deduce that this matrix cannot be used in such a scheme (TRUE).

- In a practical situation transform coding can provide more distortion than normal PCM at the same rate (TRUE or FALSE?).

It is false: transform coding is expected to have better performances in practical situations because it transform the data from the time/space domain to the frequencies domain in order to allocate more bits to more the important coefficients, incrementing the SNR.

5.3.2 February, 18 2021

Consider a continuous source X that emits the symbols $x = [\sqrt{2}, \sqrt{2}, -2\sqrt{2}, -2\sqrt{2}, -\sqrt{2}, -\sqrt{2}, 2\sqrt{2}, 2\sqrt{2}]$. You are given a quantizer $Q(\cdot)$ that applies the following quantization rule $Q(x) = \left\lceil \frac{x}{\Delta} \right\rceil * \Delta$, where $\lceil \cdot \rceil$ rounds to the closest

integer, and $\Delta = 3$ is the quantization step. Additionally, you are given the transform matrix $T = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}$.

- 1) Compute the probability mass function of x directly from the sequence.

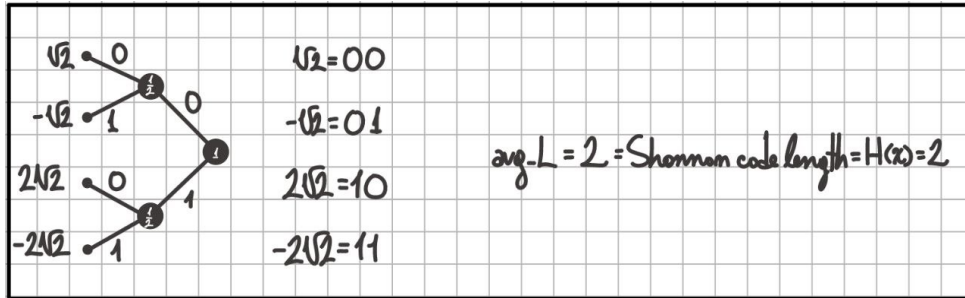
$$P(\sqrt{2}) = \frac{1}{4} \quad P(-\sqrt{2}) = \frac{1}{4} \quad P(2\sqrt{2}) = \frac{1}{4} \quad P(-2\sqrt{2}) = \frac{1}{4}$$

- 2) Compute the entropy of x .

$$H(x) = \sum_x -P(x) \cdot \log_2(P(x)) = -4 \cdot \frac{1}{4} \cdot \log_2\left(\frac{1}{4}\right) = 2$$

- 3) Compute a possible Huffman code to encode x .

- 4) Compute the average code length of the Huffman code and compare it with the Shannon code length.



- 5) Check that the matrix T is orthogonal, thus can be used for transform coding.

$$T^* T^T = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} * \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} \left(\frac{\sqrt{2}}{2}\right)^2 + \left(\frac{\sqrt{2}}{2}\right)^2 & \left(\frac{\sqrt{2}}{2}\right)^2 - \left(\frac{\sqrt{2}}{2}\right)^2 \\ \left(\frac{\sqrt{2}}{2}\right)^2 - \left(\frac{\sqrt{2}}{2}\right)^2 & \left(\frac{\sqrt{2}}{2}\right)^2 + \left(\frac{\sqrt{2}}{2}\right)^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \rightarrow \text{Orthogonality}$$

- 6) Consider a transform coding scheme that 1) select a block of two samples; 2) apply the matrix T ; 3) quantize the obtained coefficients with the quantizer Q .

- Compute the sequence y of transform coefficients obtained applying T to x (hint: consider non-overlapping blocks).
- Compute the sequence $\tilde{y}$ obtained quantizing y with Q .

$$\begin{aligned}
 X &= \begin{bmatrix} \sqrt{2} & -2\sqrt{2} & -\sqrt{2} & 2\sqrt{2} \\ \sqrt{2} & -2\sqrt{2} & -\sqrt{2} & 2\sqrt{2} \end{bmatrix} \begin{matrix} \uparrow \\ \uparrow \\ \uparrow \\ \uparrow \\ x_1 & x_2 & x_3 & x_4 \end{matrix} \\
 &\xrightarrow{T} \begin{aligned} y_1 &= \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} * \begin{bmatrix} \sqrt{2} \\ \sqrt{2} \end{bmatrix} = \begin{bmatrix} 1+1 \\ 1-1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} & y_3 &= \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} * \begin{bmatrix} -\sqrt{2} \\ \sqrt{2} \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix} \\ y_2 &= \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} * \begin{bmatrix} -2\sqrt{2} \\ 2\sqrt{2} \end{bmatrix} = \begin{bmatrix} -2-2 \\ -2+2 \end{bmatrix} = \begin{bmatrix} -4 \\ 0 \end{bmatrix} & y_4 &= \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} * \begin{bmatrix} -\sqrt{2} \\ 2\sqrt{2} \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \end{aligned} \\
 &\xrightarrow{Q} \begin{aligned} \tilde{y}_1 &= \begin{bmatrix} 3 \\ 0 \end{bmatrix} & \tilde{y}_3 &= \begin{bmatrix} -3 \\ 0 \end{bmatrix} \\ \tilde{y}_2 &= \begin{bmatrix} -3 \\ 0 \end{bmatrix} & \tilde{y}_4 &= \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{aligned} \\
 &\rightarrow \tilde{Y} = [3 \ 0 \ -3 \ 0 \ 3 \ 0]
 \end{aligned}$$

6 SPEECH CODING

Speech coding in multimedia signal processing refers to the techniques used for representing speech signals efficiently for storage or transmission.

Speech coding algorithms work by exploiting the characteristics of human speech and auditory perception. These algorithms can remove redundant and less perceptually significant parts of the speech signal, thereby achieving compression.

6.1 SPEECH SIGNALS CHARACTERISTICS

Speech signals are inherently non-stationary, meaning their statistical properties change over time. However, for practical analysis and processing, they can be considered quasi-stationary over short segments, typically ranging from 5 to 20 milliseconds. This approximation is valid because, within these brief intervals, the speech signal's characteristics do not significantly change, allowing for effective analysis and processing techniques to be applied.

Speech sounds can be broadly classified into three categories: voiced, unvoiced, and mixed. Each category has distinct acoustic and production characteristics:

- **Voiced Speech:**
 - **Quasi-periodic in the time domain:** Voiced sounds have a repeating pattern over time due to the vibration of the vocal cords.
 - **Harmonically structured in the frequency domain:** The frequency components of voiced sounds are integer multiples of a fundamental frequency, leading to a harmonic series. This structure is a result of the regular vibration of the vocal cords.
 - **Power spectrum characteristics:** The power spectrum of voiced speech is marked by its *fine structure* (the individual harmonic components) and *formant structure* (spectral peaks which are shaped by the resonant frequencies of the vocal tract). These elements combine to give voiced speech its distinctive tonal quality.
 - **Production mechanism:** Voiced sounds are produced when the vocal cords vibrate as air passes through them from the lungs. The vibration of the vocal cords modulates the airflow, creating the sound waves associated with voiced speech.
- **Unvoiced Speech:**
 - **Characterized as noisy sounds:** Unvoiced sounds lack the periodic structure of voiced sounds and have a more random, noise-like quality.
 - **Production mechanism:** Unvoiced sounds are produced by forcing air through a constriction in the vocal tract, such as between the tongue and the roof of the mouth or between the teeth. This airflow does not involve the vibration of the vocal cords.
- **Mixed Speech:** contains elements of both voiced and unvoiced sounds. Mixed speech sounds occur when there is a combination of voiced and unvoiced production mechanisms, resulting in speech sounds that have characteristics of both. For example, certain consonants may involve both a voicing component and turbulent airflow, making them neither purely voiced nor purely unvoiced.

6.2 LINEAR PREDICTING CODING FOR SPEECH

Linear Predictive Coding (LPC) is a tool used in speech signal processing to represent the spectral envelope of a digital signal of speech in compressed form, using the information of a linear predictive model.

6.2.1 Data Model

The data model in LPC assumes that a speech sample at a particular time can be predicted as a linear combination of its previous samples. This is expressed mathematically as $s(n) \approx \sum_{k=1}^p a_k * s(n - k)$, where:

- $s(n)$ is the signal at the time n .
- $s(n - k)$ represents the signal at past time instants $t = n - k$.

- a_k is the predictor coefficient: these coefficients are used to weight the past p samples of the signal in order to predict the current sample $s(n)$. Each a_k is a scalar factor that multiplies the corresponding past speech sample $s(n - k)$, where k ranges from 1 to p . The goal of these coefficients is to create a linear combination of the past values that best approximates the current value.
- p is the predictive model order: it is the number of past samples of the signal that are used to predict the current sample $s(n)$. It indicates how many previous values of the signal are considered in the linear predictive coding (LPC) model. A higher order means that the model uses more past samples to make the prediction, which can potentially lead to a more accurate representation of the signal, but it also increases the computational complexity and the number of coefficients that need to be estimated.

If we consider the signal to be an autoregressive (AR) stochastic process (an AR process assumes that the current value can be explained as a linear combination of its past values plus a stochastic error term. The process is "stochastic" because of the presence of the random error term ϵ_t that in our case is $G * U(n)$, which introduces uncertainty and variability into the model), this equation is modified in the following way:

$$s(n) = \sum_{k=1}^p a_k * s(n - k) + G * u(n) \text{ where } \begin{cases} s(n - k) \text{ represents the signal at past time instants } t = n - k \\ p = \text{model order} = \text{numb. of past samples used to predict } s(n). \\ a_k = \text{predictor coefficient that weight past } p \text{ samples of } s(n) \\ G = \text{gain parameter.} \\ u(n) = \text{Exciter Signal.} \end{cases}$$

This formulation allows us to view voice production as an excitation (modelled as white noise or train impulses) that passes through a filter represented by the predictor coefficients.

6.2.2 Source-filter Interpretation.

By taking the Z-transform of the previous time-domain equation and extracting its transfer function, we get the equation:

$$H(z) = \frac{S(z)}{G * U(z)} = \frac{1}{A(z)} \text{ where } \begin{cases} A(z) = 1 - \sum_{k=1}^p a_k * z^{-k} . \end{cases}$$

In the picture on the right is provided a simplified block diagram of the source-filter model used in speech processing, particularly in the context of Linear Predictive Coding (LPC). Here's a breakdown of each component shown in the diagram:

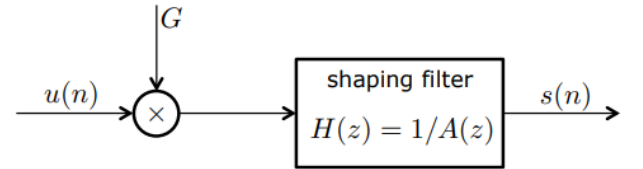


Figure 17

1. **Excitation Source $u(n)$:** this represents the input to the system, which can be thought of as a series of impulses or white noise, simulating the excitation of the vocal cords in human speech. In the context of LPC, this signal is often modelled as a series of impulses corresponding to the glottal pulses during voiced speech.
2. **Gain G :** This is a scaling factor applied (multiplication) to the excitation source to control the amplitude of the signal, adjusting the energy of the excitation to match that of the actual speech signal.
3. **Shaping Filter $H(z)$:** This is the digital filter that shapes the spectral properties of the excitation signal to produce the speech signal $s(n)$. The filter is represented in the z-domain by $H(z)$, which is the inverse of the polynomial $A(z)$.
4. **Output $s(n)$:** The output of the system is the synthesized speech signal $s(n)$.

6.2.3 Application of this Model into LPC

We can use this type of signal prediction into Linear Predictive Coding. In that case, the predicted signal $\hat{s}(n)$ would be: $\hat{s}(n) \approx \sum_{k=1}^p a_k * s(n - k)$. This means that the prediction error is equal to $e(n) = s(n) - \hat{s}(n)$, that in the Z domain becomes:

$$E(Z) = \frac{S(Z)}{H(Z)} = S(Z) * \tilde{A}(Z), \text{ that in time domain is } e(n) = G * u(n)$$

which is the 'non deterministic component' of the signal.

This leads to the following implementation:

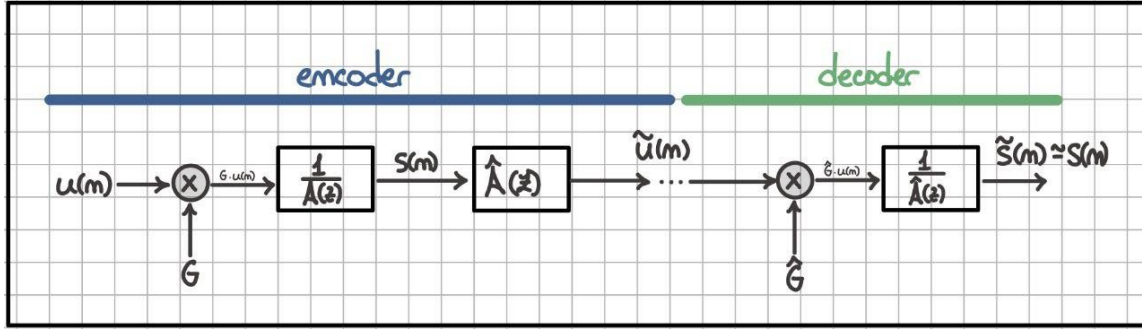


Figure 18

In which:

- $u(n)$ is the excitation signal (vibration of vocal cords), that is multiplied by the gain parameter G (*volume of our voice*). $\frac{1}{A(z)}$ is the actual shaping filter $H(z)$ (*our vocal tract*). The result $S(n)$ is the actual speech signal.
- $\hat{A}(n)$ is the estimated shaping filter. We have seen that $E(Z) = S(Z) * \tilde{A}(Z)$, so the output of this filter (when its input is $s(n)$) is $\tilde{u}(n)$: the signal that represents the pitch information of the speech. It is the output signal of the encoder and the input signal of the vocoder, and it's retrieved from the residual prediction error $e(n) = G * u(n)$. The signal $\tilde{u}(n)$ can be:
 - The actual residual prediction error $e(n) = G * u(n)$.
 - Just a Δ time representing the distance between two consecutive impulses (in this case the excitation signal is represented as a train of impulses).
- $\hat{G}$ and $\frac{1}{\hat{A}(n)}$ are the estimated gain and shaping filters at the decoder side. The input signal (the pitch information) passes through these two modules in order to reconstruct the encoded signal $\tilde{s}(n)$.

The coefficients of the estimated shaping filter $\hat{A}(n)$ can be founded in the following way:

1. We calculate the autocorrelation C_{xx} of the input signal $s(n)$;
2. Build the Toeplitz matrix of the autocorrelation:

$$R = \begin{bmatrix} r(0) & r(1) & \dots & r(p-1) \\ r(1) & r(0) & \dots & r(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ r(p-1) & r(p-2) & \dots & r(0) \end{bmatrix}$$

3. We build a column vector with p elements of the C_{xx} matrix (from the second element to the $p + 1$ one):

$$r = \begin{bmatrix} r(1) \\ r(2) \\ \vdots \\ r(p) \end{bmatrix}$$

4. Get $\hat{a}(n) = R^{-1} * r$.
5. Get the estimated shaping filter $\hat{A}(n) = 1 - \hat{a}(n)^{Transpose}$.

6.3 WAVEFORM CODERS

Waveform coders are a class of algorithms used in digital signal processing that directly encode the amplitude of the audio waveform at uniformly spaced intervals. They aim to replicate the exact waveform of the original signal during playback, which contrasts with vocoders or parametric coders that model the signal using parameters like pitch and formants. For this reason, this type of speech coding algorithm performs well with a wide range of audio signals (not only speech). Waveform coders are commonly used for applications where audio quality is of paramount importance.

We can implement waveform coders using one of the algorithms that we have studied in the previous sections:

6.3.1 [PCM](#) in Waveform Coders

5. **Uniform PCM:** This is the most straightforward type of PCM where the analog signal is sampled at uniform intervals and each sample is quantized into a fixed number of levels, with the same step size between each level. The quantized values are then binary encoded for transmission or storage. Uniform PCM is simple but can be inefficient for signals with a wide dynamic range because the same quantization resolution is applied across the signal, regardless of its complexity.
6. **Non-Uniform PCM:** To address the inefficiency of uniform quantization for signals with varying amplitude distributions, non-uniform PCM applies quantization with varying step sizes (*es. Lloyd-Max*). This approach is often implemented using algorithms like μ -law, in which a logarithmic transformation function is used to assign smaller step sizes for low amplitude signals, and larger step sizes are used for high amplitude signals, following the human ear's sensitivity curve.
7. **APCM (Adaptive PCM):** APCM stands for Adaptive Pulse Code Modulation, where the step size of the quantizer is adapted based on the input signal. This adaptation can be based on the amplitude of the previous samples or other properties of the signal. The adaptive nature allows APCM to allocate bits more dynamically, offering [better resolution during rapidly changing or high-frequency parts of the signal, and reducing resolution when the signal is relatively constant or slow-moving](#).

6.3.2 [DPCM](#) in Waveform Coders

There are two main type of DPCM that are used in Speech Coding:

8. **DPCM (Differential PCM):** DPCM is Differential Pulse Code Modulation, a variant of PCM that encodes the difference between successive samples rather than the absolute sample values. This technique takes advantage of the fact that successive samples of audio signals are often highly correlated. By encoding the differences, which tend to be smaller than the absolute values, DPCM can often use fewer bits per sample compared to standard PCM, leading to data compression.
9. **ADPCM (Adaptive DPCM):** ADPCM stands for Adaptive Differential Pulse Code Modulation. It extends the concept of DPCM by adapting the quantization step size according to the signal's variation. [The step size is increased when larger differences between samples are detected \(indicating rapid changes in the signal\) and decreased for smaller differences](#). This adaptation allows ADPCM to maintain signal quality while reducing the bit rate further than DPCM, offering an effective balance between compression and quality. The adaptation process in ADPCM can be:
 - 1) Forward adaptive: The encoder adapts its parameters based on the analysis of past signal segments and sends information about these adaptations forward to the decoder.
 - 2) Backward adaptive: The encoder adapts its parameters based on the previously encoded signal segments, and since the decoder has access to the same previously decoded segments, no additional information needs to be sent for these adaptations.

Practical ADPCM schemes typically incorporate a prediction process similar to that used in Linear Predictive Coding (LPC). In LPC, a speech sample is estimated using a linear combination of previous samples. The coefficients that multiply these previous samples are optimized to minimize the prediction error. In the context of ADPCM, a similar predictive model is used, but it focuses on predicting the difference between samples rather than the absolute sample values.

Here below there is a detailed description of the ADPCM process.

Encoder: the following operations are performed (only) in the encoder.

1. **Get the prediction error $d(n)$:** The predicted signal $\hat{x}(n)$ is subtracted from the original input speech signal $x(n)$ obtaining the residual difference $d(n)$, which is what gets encoded and transmitted because it contains the new information that was not predictable from the past samples.
2. **Quantization:** The prediction error $d(n)$ is then quantized, reducing the number of possible values it can take to a finite set, which is necessary for digital encoding. The quantization step is adjusted following the $v(n)$ signal generated by the step_update module. $d(n)$ quantized is the signal that the encoder sends to the decoder.

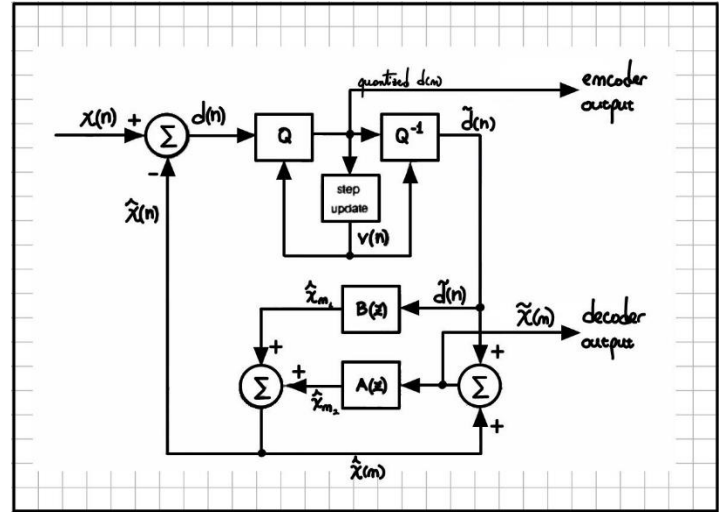


Figure 19

Encoder & Decoder: the following operations are

performed both in the decoder (to reconstruct the signal) and in the encoder (to emulate the behaviour of the decoder in order to get a prediction more similar to the one of the decoder):

3. **De-quantization $\tilde{d}(n)$:** the quantized $d(n)$ is then de-quantized in order to get the reconstructed prediction error $\tilde{d}(n)$. The de-quantization step is adjusted following the $v(n)$ signal generated by the step_update module. There is a loss of information here due to quantization.
4. **Prediction ($\hat{x}(n)$):** the prediction is performed through two parallel predictors:
 - 3) **Short term predictor ($\hat{x}_1(n)$):** the short-term predictor is used in DPCM systems to model the rapidly changing spectral characteristics of speech, which are influenced by the vocal tract's shape. The short-term prediction typically uses a [linear predictive coding \(LPC\) model](#), which can be represented by the filter $A(z)$ in the z-domain:

$$A(z) = 1 - \sum_{k=1}^p a_k * z^{-k} \text{ where } \begin{cases} a_k \text{ are the short term predictor coefficients.} \\ p \text{ is the order of the predictor.} \\ z^{-k} \text{ is the delay elements in the z domain.} \end{cases}$$

- 4) **Long term predictor ($\hat{x}_2(n)$):** the long-term prediction is used to exploit the periodic nature of voiced speech to achieve additional coding gain. Voiced speech, such as vowels and certain consonants, exhibits periodicity corresponding to the vocal cords' vibration rate, which is perceived as pitch. The long-term predictor $A_L(z)$ is defined as:

$$A(z) = \sum_{k=-j}^j a_j * z^{-i-\tau} \text{ where } \begin{cases} a_j \text{ are the long term predictor coefficients.} \\ \tau \text{ represents the pitch period.} \\ z^{-i-\tau} \text{ are the delay elements in the z domain.} \end{cases}$$

The two predictions $\hat{x}_1(n)$ and $\hat{x}_2(n)$ are summed up to get the total prediction $\hat{x}(n)$.

This prediction will be ...

- 5) **Encoded** $\rightarrow$... subtracted to the input signal to obtain $x(n) - \hat{x}(n) = d(n)$, and so the cycle of encoding restarts.
- 6) **Decoder** $\rightarrow$... summed to the de-quantized prediction error in order to get the decoded signal $\tilde{d}(n) + \hat{x}(n) = \tilde{x}(n) \cong x(n)$.

6.4 VOCODERS

Vocoders are a category of speech processing tools designed to analyze and synthesize the human voice signal efficiently. They operate on the principle of separating the speech signal into two distinct components:

10. The source (excitation): this component represents the source of the sound generation. It includes information about the vibration of the vocal cord and the sound produced by them.
11. The system (vocal-tract): this component represents the filter that the source signal passes through.

Unlike waveform coders, the performance of vocoders generally degrades for non-speech signals.

6.4.1 Channel Vocoder

Channel vocoder is one of the earliest types of vocoders, which splits the speech signal into different frequency bands using a series of bandpass filters. Here's how it works:

12. **Analysis:** The incoming speech signal is divided into a number of frequency bands. Each band captures a specific portion of the speech spectrum. The energy content in each band is then measured, typically by rectifying and low-pass filtering the output of each bandpass filter, which yields the amplitude envelope of that band over time.
13. **Carrier Signal:** A noise or tone generator creates a carrier signal, which is then modulated by the amplitude envelopes derived from the speech signal.
14. **Synthesis:** The modulated signals from all bands are summed together to reconstruct the speech signal. The result retains the temporal characteristics of the original speech but loses some of the finer spectral details, as the carrier signal does not carry the phase information of the original signal.

Channel vocoders are known for producing robotic-sounding speech, which is why they are often used for [musical effects](#) rather than accurate speech reproduction.

6.4.2 Linear Predictive Vocoder

A linear predictive vocoder is a type of vocoder that utilizes the concept of linear prediction to analyze and synthesize speech signals. It exploits the source-system model scheme by estimating the parameters of the vocal tract filter and the excitation source. In a linear predictive vocoder, the encoder and decoder work together to analyze and synthesize speech signals. The encoder analyzes the input speech signal and extracts the relevant parameters, while the decoder uses these parameters to synthesize a reconstructed speech signal.

The main parameters sent from the encoder to the decoder in a linear predictive vocoder include:

1. **Linear prediction coefficients ($A(z)$):** parameters that represent the vocal tract filter transfer function.
2. **Pitch period (time lag Δ):** represents the fundamental frequency of the voiced speech.
3. **Voicing decision (zcr & ste):** determines whether a particular segment of speech is voiced or unvoiced.

LPC vocoders are more efficient in terms of data bandwidth than channel vocoders and can produce more natural-sounding speech. They are widely used in telecommunications and speech synthesis systems due to their ability to produce intelligible speech at very low bit rates.

6.4.2.1 Source-System Model

In speech coding, a source-system model is fundamental for understanding how speech signals are generated and processed. This model is based on the theory that the production of speech sounds can be divided into two main components: the source and the system.

- a) **Excitation:** The excitation represents the source component of speech production. It can be voiced or unvoiced (**V/UV**).
- b) **Glottal Model:** It specifically refers to modelling the excitation signal generated by the vocal cords. It represents the vibrations and characteristics of the glottis, which is responsible for the voiced speech sounds. The glottal model helps in generating a periodic pulse-like waveform to represent the pitch and periodicity of the voiced sounds.

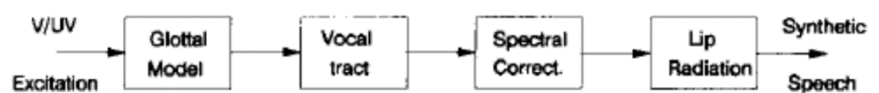


Figure 20

(Excitation and Glottal Model are incorporated into the $U(Z)$ in source-system model equation).

- c) **Vocal Tract:** It represents the filter component in the source-system model. It corresponds to the physical structure of the vocal tract, including the

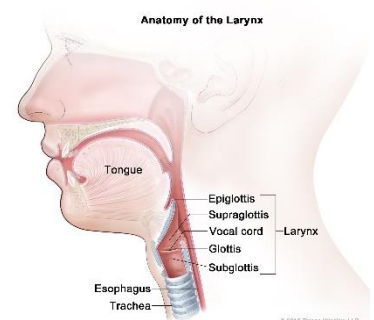


Figure 21

throat, mouth and nasal cavity. The vocal tract act as a resonant filter that shapes the excitation signal from the glottal model, determining the spectral characteristic of the speech sound.

- d) **Spectral Correction:** It relates to the process of obtaining the spectral envelope of the speech signal. This involves using a bank of bandpass filters to capture the resonant characteristics of the vocal tract.
- e) **Lip Radiation:** This block accounts for the effects of radiation from the lips and mouth during speech production. It considers the acoustic radiation pattern of the vocal tract and how it affects the final sound that is radiated into the surrounding air.
- f) **Synthetic Speech:** is the output of our system.

(Vocal tract, Spectral Correlation and Lip Radiation are incorporated into the $H(z) = \frac{1}{1-A(z)}$ filter in source-system model equation).

This model can be represented using the following z-domain function:

$$S(z) = \frac{G}{1 - A(z)} * U(z) \text{ where } \begin{cases} S(z) = \text{Z transform of the synthesized speech signal.} \\ G = \text{gain factor that adjusts the overall amplitude of the signal.} \\ \frac{1}{1 - A(z)} = H(z) = \text{LPC synthesis filter (vocal tract).} \\ U(z) = \text{Z transform of the Excitation signal (pulse train or white noise)} \end{cases}$$

The system and excitation parameters, specifically G and $A(z)$, are not inherent to the speech signal and must be estimated from a finite set of speech samples. The estimation process typically involves analyzing a window of speech, which is often 20 – 30ms long, updating the parameters every 10-30ms to track changes in the speech signal over time.

For vocal tract representation, generally, 8 – 14 linear prediction (LP) parameters are sufficient, providing a balance between computational efficiency and the ability to accurately model the intricacies of human speech. The specific number of parameters can vary depending on factors like the desired quality of synthesis and the characteristics of the speaker's voice.

6.4.2.2 Quantization of LP parameters

In Linear Predictive Coding (LPC), one major issue is the quantization of LP parameters that are transmitted to the decoder. This process can introduce errors that may affect the stability and quality of the synthesized speech.

The LPC synthesis filter, represented as $\frac{1}{1-A(z)}$, is sensitive to the quantization of its coefficients. Directly quantizing the coefficients of the filter $A(z)$ can lead to instability in the synthesis filter, which is why alternative methods of representing the LPC parameters are often used:

- 7) **Reflection coefficients:** one such method is to use reflection coefficients, denoted as k_i , which are inherently stable when quantized because they are bounded between -1 and +1. These coefficients can be ordered and exploited for encoding, with the first reflection coefficients typically being encoded with higher precision because they have more influence on the filter's response. Additionally, transformations of reflection coefficients can lead to parameters that are less sensitive to quantization errors. The **Log Area Ratios (LARs)** are one such transformation, calculated as: $LAR(m) = \log\left(\frac{1+k_m}{1-k_m}\right)$. LARs have been used in many coding applications because they provide a more robust representation that can handle quantization better.
- 8) **Line Spectrum Pairs:** Line Spectrum Pairs (LSPs) are an alternative representation of LPC coefficients that provide some advantages, particularly in the quantization and transmission of LPC parameters for speech coding. The LSPs are frequencies that represent the spectral envelope of a speech signal. For a 10th order LPC polynomial, typically written as $A_{10} = 1 + a_z * z^{-1} + \dots + a_{10} * z^{-10}$ LSPs are derived by forming two new polynomials that are symmetric and antisymmetric around the unit circle in the z-plane. These polynomials are: $\begin{cases} A_{101}(z) = A_{10}(z) + z^{-11} * A_{10}(z^{-1}) \text{ that contains the even order coefficients.} \\ A_{102}(z) = A_{10}(z) - z^{-11} * A_{10}(z^{-1}) \text{ that contains the odd order coefficients.} \end{cases}$ Each of these new polynomials has five complex conjugate pairs of zeros on the unit circle. The angles corresponding to these zeros are the LSP frequencies. They can be used to reconstruct the original LPC polynomial by the relation $A_{10}(z) = \frac{A_{101}(z) + A_{102}(z)}{2}$.

One significant advantage of LSPs is that they can be coded using perceptual quantization rules, which take into account the human ear's sensitivity to different frequencies. This means that the quantization noise introduced during the transmission can be minimized in the perceptually important regions of the speech spectrum, leading to better-sounding decoded speech.

6.4.3 Open Loop Excitation Models

In Linear Predictive Coding (LPC) for vocoders, open-loop excitation models refer to methods of generating the excitation signal without feedback from the system output. These models determine how the vocal tract filter is excited in speech synthesis. Here are the types of open-loop excitation models.

6.4.3.1 Two-state Excitation Model

The two-state excitation model in speech processing is a method used to generate the excitation signal for Linear Predictive Coding (LPC) synthesis. It operates on the principle that speech can be categorized into two primary types: voiced (*typically a pitch-synchronous pulse train*) and unvoiced (*typically a random noise generator*).

The parameters that define this model are:

- 9) **Gain:** Adjusts the amplitude of the excitation signal to match the energy of the analyzed speech segment.
- 10) **Pitch Period:** Determines the fundamental frequency of the voiced speech. In the case of voiced sounds, the excitation signal is typically a periodic pulse train corresponding to the pitch of the speech.
- 11) **Binary Voicing Parameter:** A flag that indicates whether the speech is voiced or unvoiced. If the parameter is set for voicing, a pitch-synchronous pulse train is used. If not, random noise is used to represent unvoiced speech.

6.4.3.2 Mixed Excitation Model

These models provide a combination of periodic (voiced) and aperiodic (unvoiced) excitations simultaneously. This allows for a more natural representation of sounds like fricatives or mixed voicing, which can have characteristics of both voiced and unvoiced sounds.

The mixed excitation model works as follow:

1. **Generation of the sound:**
 - a. the **Impulse Train Generator** produces a series of impulses at the pitch period for voiced speech. The impulses are spaced at intervals corresponding to the fundamental frequency of the voice.
 - b. Simultaneously, the **Noise Generator** provides a random noise signal for unvoiced speech parts. The noise generator ensures that unvoiced sounds, which lack a clear pitch, are represented accurately. The noise signal is often passed through a **lowpass filter f_c** to limit its bandwidth to match that of typical unvoiced speech sounds. The cutoff frequency f_c is chosen based on the spectral content of unvoiced sounds.
2. **Sum of the two signals:** The filtered noise and impulse train are combined in the **summation module Σ** . The proportion of each signal in the mix can vary, allowing for fine-tuning of the excitation signal to match the natural characteristics of the speech being modeled.
3. **Amplification of the sound:** the **Amplifier (g)** sets the final mixed excitation signal to the appropriate level.

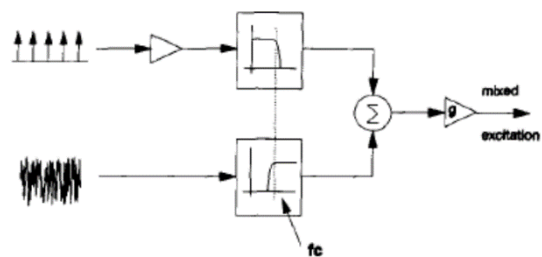


Figure 22

The output signal will be then used to excite the LPC filter in the synthesis process.

6.4.3.3 Residual excited linear prediction

Residual Excited Linear Prediction (RELP) is a technique in linear predictive vocoders where the excitation signal for the synthesis filter is the residual signal from the LPC analysis. The residual signal is the difference between the actual speech waveform and the waveform predicted by the LPC filter. This difference contains the finer details of the speech signal that the LPC model doesn't capture, including phase and pitch information, and characteristics

from nasal and other sounds not modeled by the LPC filter. By using this signal as the excitation for the LPC synthesis filter, RELP maintains the unique characteristics of a speaker's voice, resulting in more natural-sounding synthesized speech.

In RELP, typically, only the low-frequency components of the residual are transmitted, as they carry most of the speech intelligibility. The high-frequency components, less critical for understanding speech, are not transmitted but rather synthesized at the decoder to reconstruct the full speech signal. This approach is beneficial for reducing the amount of data that needs to be transmitted, thus making it a bandwidth-efficient method of speech coding.

6.5 ANALYSIS LINEAR PREDICTIVE CODERS MATLAB IMPLEMENTATION

In order to implement an Analysis-by-synthesis Linear Predictive Coder in MATLAB you must follow the steps below:

6.5.1 Windowing Configuration

```
%% Analysis Window Configuration.
w_dur_sec = 0.04; % window duration in seconds
w_dur_sampl = w_dur_sec * Fs; % window duration in samples
w_shift_sec = 0.01; % shift from a window and the next (stride)
w_shift_sampl = w_shift_sec * Fs; % stride in samples
window = hamming(w_dur_sampl);
numOfWindows = floor((length(x)-w_dur_sampl)/w_shift_sampl)+1;
```

6.5.2 Voiced vs Unvoiced detection

```
%% Voiced vs Unvoiced detection
ZCR_STE = zeros(numOfWindows, 2);
for i = 1 : numOfWindows
    idxs = 1+(i-1)*w_shift_sampl:(i-1)*w_shift_sampl+w_dur_sampl;
    windowedFrame = x(idxs).*window;
    ZCR = sum(abs(diff(windowedFrame>0)))/w_dur_sampl;
    STE = sum(windowedFrame.^2);
    ZCR_STE(i,1)=ZCR;
    ZCR_STE(i,2)=STE;
end
threshold = median(ZCR_STE,1); % 1 = along first dimension (rows).
voiced = ZCR_STE(:,1)<threshold(1) & ZCR_STE(:,2)>threshold(2);
```

6.5.3 Speech Synthesis

```
%% Speech Synthesis
% Set up the initial variables
p = 12; % LP order
% Pitch detection initial parameters
min_f = 60;
max_f = 500;
max_lag = ceil(Fs/min_f);
min_lag = floor(Fs/max_f);
% Initialize the vector that will contains the synthesized speech
synth_voice = zeros(size(x));
for i = 1 : numOfWindows
    idxs= 1+(i-1)*w_shift_sampl : (i-1)*w_shift_sampl+w_dur_sampl;
    currWin = x(idxs).*hamming(length(numOfWindows));
    % Extract the LP coefficients (A_hat)
    [Cxx, lags] = xcorr(currWin);
    Cxx_pos = Cxx(lags>=0); % only positive values
    R = toeplitz(Cxx_pos (1:p));
    r = Cxx_pos (2:p+1);
    a_hat = R\r;
    A_hat = [1; - a_hat];
    % Get the prediction error= u(n)*G (consider delay introduced by conv).
    if i == 1
        [error, errDelay] = filter(A_hat, 1, currWin);
    else
        [error, errDelay] = filter(A_hat, 1, currWin, errDelay);
    end
    % Generation of the exciter (output_encoder → input_decoder).
    if voiced(i) == 1
        % Pitch Detection.
        Cxx_crop_pos = Cxx_pos(min_lag+1:max_lag+1);
        [~, relative_lag] = max(Cxx_crop_pos);
        time_lag = min_lag + relative_lag - 1;
        e = zeros(size(currWin));
        e(1:time_lag:end)=1; % train of pulses (voiced).
    else
        e = randn(size(currWin)); % random noise (unvoiced).
    end
    e = e/std(e)*std(error);
    % Synthesize speech (consider delay introduced by convolution).
    if i == 1
        [synth_voice_frame, synthDelay] = filter(1, A_hat, e);
    else
        [synth_voice_frame, synthDelay] = filter(1, A_hat, e, synthDelay);
    end
    synth_voice(idxs) = synth_voice(idxs) + synth_voice_frame;
end
% Normalize sound from -1 to 1
synth_voice = (synth_voice-min(synth_voice))/(max(synth_voice)-min(synth_voice)) * 2 - 1;
```

6.6 ANALYSIS-BY-SYNTHESIS LINEAR PREDICTIVE CODERS

Analysis-by-synthesis works by generating a synthetic version of the speech signal based on a set of parameters, then comparing this synthesized speech to the original speech signal. The difference between the two is analyzed, and the parameters are adjusted to minimize this difference.

The modules of an analysis by synthesis linear predictive coder are:

1. **Prediction section:** this module is inherited from [waveform coders](#). In this module, **System Parameters** are determined by linear prediction, which is a technique used to estimate the spectral envelope of speech by

predicting current speech samples based on past samples. The system comprises a short-term LP synthesis filter that models the formant structure of speech, a long-term LP synthesis filter that captures the pitch.

2. **Excitation Sequence:** this module is inherited from [vocoders](#). As we have seen in the case of [open loop excitation models](#), the excitation sequence determine how the vocal tract filter is excited in speech synthesis. In Analysis By Synthesis coders, the excitation sequence is optimized through a closed-loop process that involves selecting an excitation sequence that minimizes a weighted difference between the original speech signal and the synthesized speech signal. The error is weighted to reflect the sensitivity of the human ear, ensuring that perceptually important aspects of speech are preserved.

The way in which these components are combined is explained below:

1. **Sound Generation:** the **Exciter** is where the excitation signal $\tilde{d}(n)$ for the LPC filter is chosen or generated. It could be a residual signal, a noise source, or a combination, depending on the speech being modeled. The amplitude of $\tilde{d}(n)$ is adjusted to match the energy level of the target speech signal through the **gain** module.

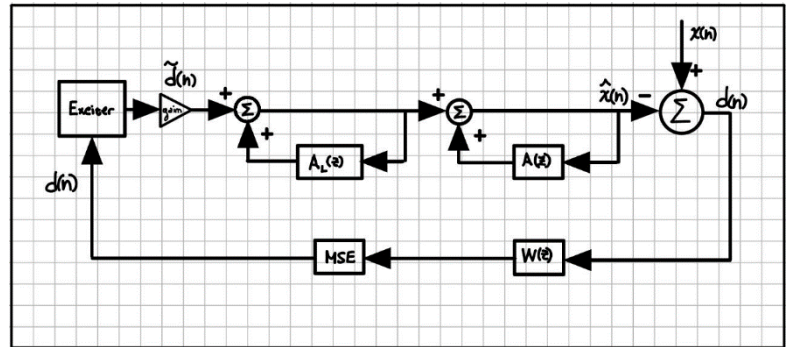


Figure 23

2. **Prediction:** The excitation signal $\tilde{d}(n)$ is fed into the **LPC filters**:
 - c. $A_L(z)$ is the long-term LPC filter representing pitch.
 - d. $A(z)$ is the short-term LPC filter modelling the vocal tract (formant structure).

These filters process the excitation signal to produce a synthesized speech signal $\hat{x}(n)$.

3. **Feedback generation:** the difference $x(n) - \hat{x}(n)$ is calculated to get the **residual error** $d(n)$. This error passes through a **W(z)** filter that weights the error coefficient following the sensitivity curve of the human ear (more importance to the lower frequencies coefficients). The weighted error passes then into a **MSE** module that calculates the Mean Square Error of the predicted signal compared to the original one. The final feedback signal is then passed to the exciter, that will optimize the excitation sequence in order to make the predicted speech signal closer to the actual speech signal.

So, Analysis-by-synthesis LPC combines the features of model-based vocoders, which represent the formant and pitch structures, with waveform coders that match the input speech waveform. This hybrid approach, augmented by perceptual weighting, ensures that the encoded speech sounds natural to the listener.

6.7 COMMON EXCITATION MODELS USED IN SPEECH PROCESSING

6.7.1 MultiPulse Excited Linear Prediction (MPLP)

This algorithm creates an excitation **sequence composed of multiple, non-uniformly spaced pulses**. The amplitude and location of each pulse are determined to minimize the weighted mean square error between the synthesized and actual speech. MPLP typically uses 4-6 pulses every 5 milliseconds. It provides a more accurate speech representation but is computationally more intensive because it codes both the amplitudes and the locations of the pulses.

6.7.2 Regular Pulse Excitation (RPE)

In contrast to MPLP, RPE uses **pulses that are uniformly spaced**. The positions of these pulses are determined once the location of the first pulse is known. RPE commonly uses a spacing factor of 3 to 4 and updates the first pulse location every 5 milliseconds. RPE typically includes 10 to 13 pulses in each 5ms segment. This method is less complex than MPLP.

6.7.3 Code Excited Linear Prediction (CELP)

CELP is designed for low-rate, high-quality speech coding. It uses a **codebook of Gaussian sequences** to represent the excitation signal. The codebook contains a large number of vectors, and the CELP algorithm selects the optimal vector that minimizes the perceptually weighted MSE. This approach effectively bridges the gap between waveform coders and vocoders, balancing computational efficiency with speech quality.

7 AUDIO CODING

Audio coding in multimedia signal processing refers to the technology and methods used to compress and decompress digital audio data. The primary goal of audio coding is to reduce the size of audio files or the bandwidth needed for transmission without significantly compromising the audio quality perceived by listeners. This is achieved through various algorithms and techniques that exploit the properties of human hearing (psychoacoustic models) to remove parts of the audio signal that are less likely to be noticed by the average listener.

7.1 PERCEPTUAL CODING GENERAL PROCESS

Unlike speech coding, where the signal has a more predictable structure and can often be effectively encoded by modeling the vocal tract and pitch, music and general audio signals lack a simple generative model. This complexity makes it impractical to rely on a predictive model for efficient encoding. To address this challenge, audio coding algorithms have adopted a strategy known as perceptual coding. Perceptual coding is underpinned by the principle of psychoacoustics, which studies how humans perceive sounds. By understanding the limitations and characteristics of human hearing, perceptual coding aims to eliminate parts of the audio signal that are less critical to our perception. The following block diagram of perceptual audio coding illustrates this process through several key stages.

7.1.1 Encoder

- **Analysis Filter Bank:** this module decomposes the input signal into subsampled spectral components. This means the audio signal is divided into multiple bands of frequencies, each band capturing a portion of the spectrum present in the original signal.
- **Perceptual Model:** It uses the time domain or time/frequency domain representation of the signal to compute an estimate of the actual (time-dependent) masked threshold. The masked threshold represents the point below which a signal becomes inaudible in the presence of another, louder signal (masker). This model helps to identify which parts of the audio can be altered or discarded based on human auditory perception.
- **Quantization and Coding:** The spectral components outputted by the Analysis Filter Bank are quantized and coded. The goal during this stage is to keep the noise introduced by quantization below the masked threshold identified by the perceptual model, ensuring that any degradation in audio quality remains imperceptible to the listener. This is achieved through following the “instruction” from the Perceptual Model, assigning more bits to more important (from the point of view of human perception) spectral coefficients of the signal and less bits (or even discard) to the less perceivable spectral coefficients.
- **Encoding of bitstream:** the spectral coefficients are then encoded using an encoding algorithm (es. *Huffman*) giving as result a bit stream output.
- **Frame Packing:** Frame packing refers to the process of assembling the bitstream of the quantized and coded spectral coefficients along with some side information, such as bit allocation information. This side information is essential for the decoder to accurately reconstruct the audio signal. Frame packing organizes all this information into a structured format, or "frame", which can then be easily handled by transmission and storage systems.

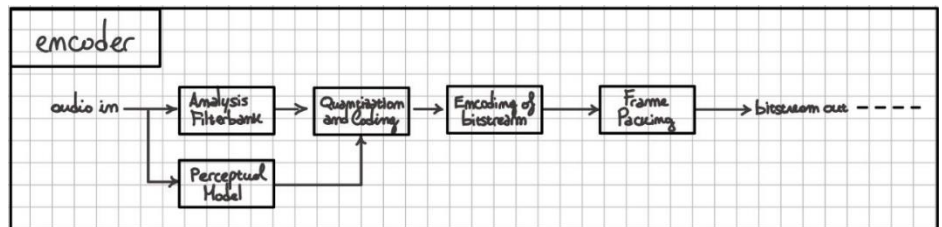


Figure 24

7.1.2 Decoder

- **Decoding of bitstream:** the output bitstream of the encoder is there decoded to retrieve the quantized signal.
- **Inverse quantization:** The inverse of the quantization process, this block takes the quantized values and reconstructs the approximate original sub-band samples.
- **Synthesis Filterbank:** This block recombines the sub-band samples into a full bandwidth audio signal (*using the informations provided by the Frame Packing module in the encoder*). It's the inverse process of the analysis filterbank.
- **Audio Out:** This is the output digital audio signal that has been decompressed. It's an approximation of the original audio in, reconstructed from the compressed bitstream.

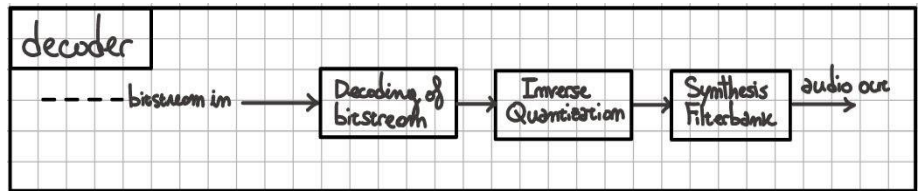


Figure 25

7.2 EAR ANATOMY AND PSYCHOACOUSTIC BASICS

7.2.1 The ear

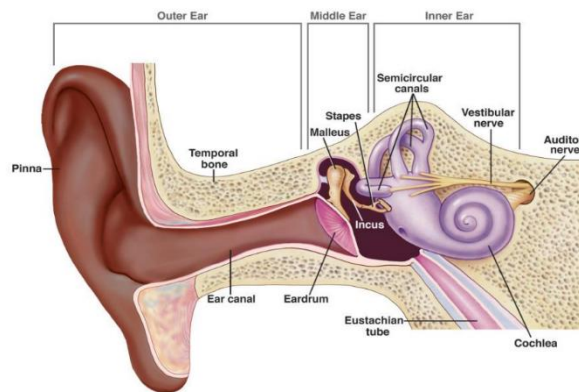


Figure 26

The human ear is a complex and highly specialized organ responsible for hearing. It consists of three main sections: the outer ear, middle ear, and inner ear, each playing a vital role in the process of hearing.

Let's analyze the composition of the human auditory system by following the path of sound from the environment to the brain:

1. Outer Ear:

- **Pinna:** Sound waves enter through the pinna (the visible part of the ear), which captures these waves and funnels (*incanala*) them into the ear canal. The pinna is unique to everyone, similar to a fingerprint. This uniqueness contributes to slight differences in how each person hears.
- **Ear Canal:** The sound waves travel down this canal, reaching the eardrum (tympanic membrane).

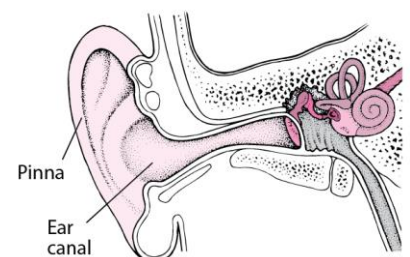


Figure 27

2. Middle Ear:

- **Eardrum:** This membrane vibrates in response to the sound waves.

- **Ossicles:** The middle ear contains three tiny bones named the malleus (hammer), incus (anvil), and stapes (stirrup).
 - **Malleus (*Martello*):** Attached to the eardrum, it vibrates with the eardrum.
 - **Incus (*Incudine*):** Receives vibrations from the malleus and passes them to the stapes.
 - **Stapes (*Staffa*):** The stapes vibrates against the oval window, a membrane-covered opening (*un'apertura coperta da membrana*) to the inner ear, transferring the sound vibrations.

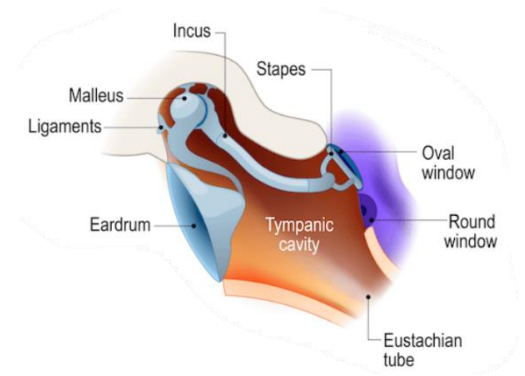


Figure 28

They function like an electrical transformer, amplifying pressure and reducing motion while preserving energy. This process adapts the sound wave from air to the liquid environment of the cochlea.

- **Eustachian Tube:** This tube links the ear canal to the throat (*gola*), helping to equalize pressure between the ear's internal and external environments. This equalization protects the eardrum from sudden changes in air pressure and loud noises.

3. Inner Ear:

- **Cochlear Windows (Oval and Round):** The oval window connects to the stirrup, while the round window acts as a damping mechanism. This setup compensates for pressure changes caused by the oval window's oscillations.
- **Cochlea:** The most crucial part of the inner ear, the cochlea is a labyrinthine structure where vibrations are converted into electrical signals. It is filled with perilymph, a fluid with an ionic composition similar to plasma and cerebrospinal fluid. The vibrations cause the fluid inside the cochlea to move.
- **Hair Cells:** The cochlea's basilar membrane is lined with over 30,000 hair cells. These cells convert the mechanical vibrations into electrical signals as they bend (*mentre si piegano*) in response to fluid movement. Different regions of the basilar membrane are sensitive to different frequencies, following the logarithmic function $\omega_0 = 2^{\frac{x}{3.5}} + a$ (The variable a represents the characteristic frequency location on the basilar membrane of the cochlea closest to the oval window. It is the frequency for which the distance x from the oval window is zero). High-frequency sounds primarily vibrate the base of the membrane (near the oval window), which is stiffer and narrower (*più rigido e più stretto*). Low-frequency sounds, conversely, vibrate the apex (*apice*) of the membrane, which is wider and more flexible.
- **Auditory Nerve:** These electrical signals are transmitted to the brain through this nerve.

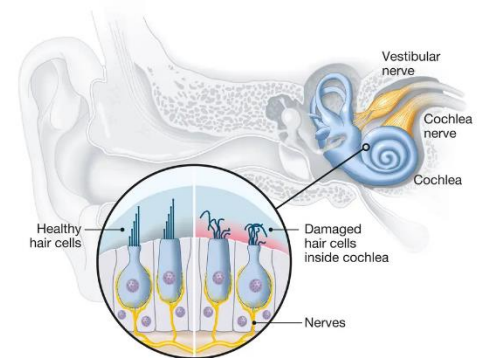


Figure 29

The middle and inner Ear are covered by the **Temporal Bone**, that is one of the hardest parts of the body.

OUR AUDITORY SYSTEM, WHILE COMPLEX, DOES HAVE ITS LIMITATIONS.

7.2.2 SOUNDS ILLUSIONS

7.2.2.1 Superposition of Pure Tones - 1st Order Effect (originated in the cochlea)

When two sounds with closely related frequencies, denoted as f_1 and f_2 , are played together, we don't hear them as separate. Instead, we perceive a single tone that oscillates. This perceived tone has a frequency that is the average of f_1 and f_2 , and it also beats or pulsates at a frequency that is the difference between $\frac{f_1 - f_2}{2}$.

Mathematically, this is represented as follows:

$$\sin(A) + \sin(B) = 2\sin\left(\frac{A+B}{2}\right)\cos\left(\frac{A-B}{2}\right)$$

$$\sin(2\pi f_1 t) + \sin(2\pi f_2 t) = 2\sin\left(2\pi \frac{(f_1 + f_2)}{2} t\right) \cos\left(2\pi \frac{(f_1 - f_2)}{2} t\right)$$

The sine component on the right-hand side of the equation corresponds to the average frequency of the two tones, which is the frequency we perceive. The cosine component represents the beat frequency, which is how fast the sound seems to pulse.

What's fascinating is that we are hearing a frequency that is not actually being played by any of the two sources. This is a testament to the constructive and destructive interference patterns that sound waves can form and how our auditory system interprets these patterns.

7.2.2.2 Superposition of Pure Tones - 2nd Order Effect (originated in the neural system)

This effect is rooted in the neural system. Unlike physical sound wave interference, the 2nd-order effect involves the brain's interpretation of the sound waves: when two pure tones are played with frequencies f_1 and f_2 , where f_2 is slightly different from $2 * f_1$, the brain perceives a tone beat at the frequency $f_2 - 2 * f_1$.

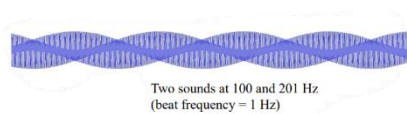


Figure 31

In contrast to the 1st-order effect, where beats are heard as changes in amplitude, the 2nd-order effect maintains a constant amplitude. Instead of amplitude modulation, listeners experience cyclic changes in the shape of the waveform. This can be perceived as a variation in the sound's timbre or quality, without fluctuations in loudness.

7.2.3 Timbral Consonance and Dissonance

Understanding whether two sounds are consonant or dissonant involves analyzing their frequency separation and how this separation is perceived by the human auditory system. In particular, it is important the concept of critical Bandwidth, which refers to the frequency range within which two sounds affect the same group of hair cells in the cochlea. The sounds within a critical bandwidth are more likely to interact, affecting our perception of consonance and dissonance.

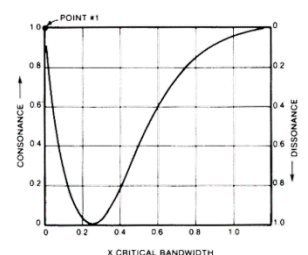


Figure 32

Figure 30

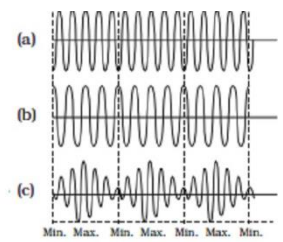


Fig. Graphical representation of beats

When two tones are close in frequency (within a small separation), we perceive a single beat or a combined tone rather than two distinct sounds because these sounds are in the same critical bandwidth and stimulate the same hair cells. As the frequency separation increases, we initially perceive a sensation of roughness (*ruvidosità*, *rozzezza*), which is associated with dissonance. When the frequency separation exceeds the critical bandwidth, we perceive two distinct tones, indicating a transition from dissonance to consonance.

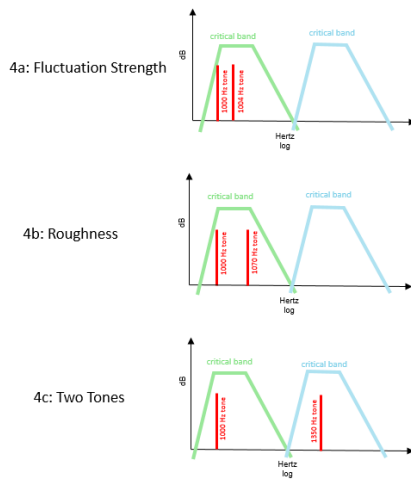


Figure 33

The width of the critical bandwidth changes in function of the frequency: at lower frequencies, the critical bandwidth is larger, which can make intervals sound muddy (*torbido*) or less distinct; at higher frequencies, the critical bandwidth becomes narrower, and intervals sound more consonant and clearer.

This phenomenon explains why certain intervals on a piano sound consonant in higher registers but dissonant in lower registers.

For the same reason, the spacing of chords in Western music tends to become wider in lower registers, which helps avoid sensory dissonance. The left hand of a pianist often plays voicings that span a wider range when the lowest note is around the octave below middle C. In contrast, the right hand can play more closely spaced intervals without creating dissonance because the critical bandwidth in higher registers is narrower.

7.2.3.1 Bark Scale

The definition of critical bandwidth leads to the partitioning of the frequency domain in critical bands similar to the one shown on the right. The frequency scale that is derived by mapping frequencies to critical band numbers is called the **Bark scale**.

z/Bark	f_0/Hz	f_c/Hz	$\Delta f_0/\text{Hz}$	f_w/Hz
0	0	100	100	50
1	100	200	100	150
2	200	300	100	250
3	300	400	100	350
4	400	510	110	450
5	510	630	120	570
6	630	770	140	700
7	770	920	150	840
8	920	1080	160	1000
9	1080	1270	190	1170
10	1270	1480	210	1370
11	1480	1720	240	1600
12	1720	2000	280	1850
13	2000	2320	320	2150
14	2320	2700	380	2500
15	2700	3150	450	2900
16	3150	3700	550	3400
17	3700	4400	700	4000
18	4400	5300	900	4800
19	5300	6400	1100	5800
20	6400	7700	1300	7000
21	7700	9500	1800	8500
22	9500	12000	2500	10500
23	12000	15500	3500	13500

Figure 34

7.2.4 Pitch: The Subjective Impression of Frequency

Pitch is a subjective impression of frequency. It doesn't always correspond to the real frequency of the sound, a phenomenon due to the way our brain processes sound information.

7.2.4.1 Just Noticeable Difference (JND)

It is the smallest variation of a frequency we can perceive. Our hearing is most sensitive in the mid-frequency range of 20Hz to around 20kHz, with the most sensitivity lying in the logarithmic scale from 1-4 kHz, crucial for the frequencies characterizing speech. *Figure 11* presents a graph that illustrates this concept. The difference limen is the smallest change in a stimulus that can be detected. On the y-axis, we have the difference limen for frequency expressed as a percentage, and on the x-axis, we have the frequency of the tones in hertz (Hz). We can observe from the graph that at lower frequencies (below 500 Hz), the percentage change needed for a difference to be noticed is larger. This implies that at lower frequencies, our ears are less sensitive to changes in frequency. As the frequency increases, particularly in the mid-range (around 1000 to 5000 Hz), our sensitivity to frequency changes increases, meaning that smaller percentage changes are detected. However, at very high frequencies (above 5000 Hz), the curves level out, indicating that our ability to discern changes does not improve significantly in this range.

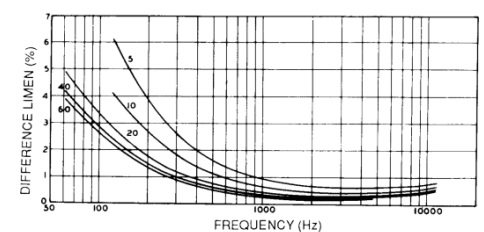


Figure 35

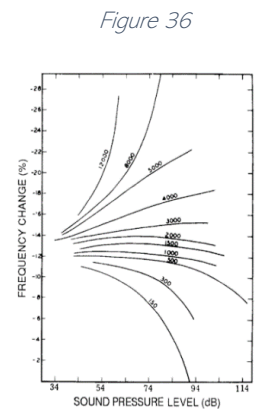
7.2.4.2 Fundamental Note

All sounds that create a sensation of pitch are periodic. Their spectrum consists of harmonics that are integer multiples of the fundamental frequency. The pitch of a complex periodic tone is closely related to the pitch of a sine wave at the fundamental. Helmholtz postulated that pitch is heard at the fundamental frequency because this frequency creates the lowest frequency peak on the basilar membrane in the ear.

Seebeck and later Schouten demonstrated that even complex periodic sounds with no energy at the fundamental frequency can still produce a clear pitch sensation at the fundamental. This phenomenon is called **Missing fundamental**.

7.2.4.3 Pitch and Intensity

Pitch not only depends on the frequency but also on the intensity or amplitude of sound. Even when playing the same pitch at different volume levels, our pitch perception may change, leading us to perceive different notes. The *figure 12* shows how pitch changes in function of the intensity (*when intensity increases: low frequencies are perceived as lower pitch, high frequencies are perceived as higher pitch*).



7.2.4.3.1 Mel Scale

Pitch is measured using the Mel scale (*figure 13*). It is a scale that relates the perceived frequency of a tone to the actual measured frequency. It's based on subjective tests to quantify how listeners perceive pitches: the idea is that equal distances on the Mel scale sound equally distant to the human ear. The scale is non-linear: it is more linear at lower frequencies and more logarithmic at higher frequencies. This reflects how the human ear perceives sound. The unit of measurement on this scale is "Mels." For reference, 1000 Hz is usually set to 1000 Mels, and other frequencies are scaled relative to this.

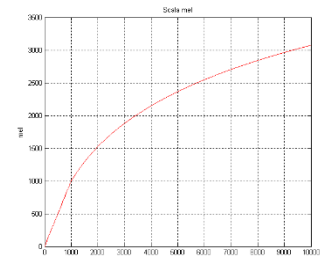


Figure 37

Figure 13 shows what we have just said: the horizontal axis represents the frequency of the sound in Hertz (Hz), while the vertical axis represents the perceived pitch of the sound measured in Mels. As usually, 1000 Hz is set as a reference point where the frequency in Hertz and the perceived pitch in Mels is the same.

7.2.4.4 Acoustic Intensity Measurement

Acoustic intensity, often described as loudness, is how we subjectively experience the power of a sound. It's not just about how strong the sound is; it's about how we perceive its strength. For example, two sounds might have the same Sound Pressure Level (SPL), which is an objective measurement, but we might hear them as being of different loudness if they are at different frequencies.

7.2.4.4.1 Phon scale and Fletcher and Munson's iso-phonic curves

The unit of measurement of perceived intensity is the phon. The phon scale is based on the dB scale (decibels) but adjusted for the ear's varying sensitivity at different frequencies. For instance, our ears are less sensitive to very low/very high frequencies. The relation between perceived and real intensity is:

$$0 \text{ phons} = \text{perceived intensity of a } 1000\text{Hz tone at } 0 \text{ Db (SPL)}$$

So, if a sound is said to have a level of x phons, it is as loud to us as a 1000 Hz tone would be at x decibels SPL.

To better understand this concept, we could use the Fletcher and Munson's iso-phonic curves (*figure 14*). This plot shows several curves, each representing a constant loudness level in phons. The curves swoop upwards at the lower and higher ends of the frequency spectrum, indicating that at these frequencies, a sound must have a higher SPL for us to perceive it as being equally loud to a sound at 1000 Hz and 0dB. The lowest curve marks the threshold of audibility—below this line, sounds are too faint to hear. The highest curve denotes the threshold of pain—above this, sounds are uncomfortably loud. This plot illustrates that our perception of loudness changes with frequency and is not a straight linear relationship with the SPL.

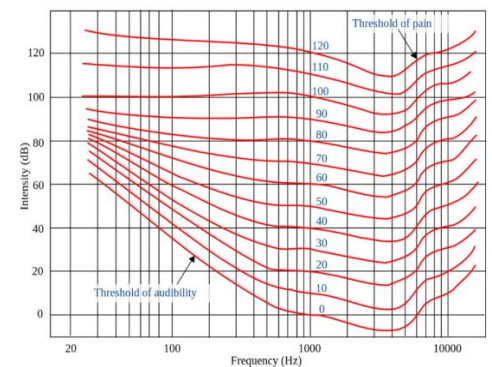


Figure 38

7.2.4.5 Intensity Thresholds

The audibility of sound also depends on intensity thresholds, which are the minimum and maximum SPLs at which sound is perceivable without causing pain. The threshold of pain indicates the SPL beyond which sound becomes painfully loud, while the hearing threshold indicates the lowest SPL at which sound is just audible.

7.2.5 Masking

Masking refers to a process where one sound is rendered inaudible because of the presence of another sound.

It is convenient to distinguish between only three types of simultaneous masking:

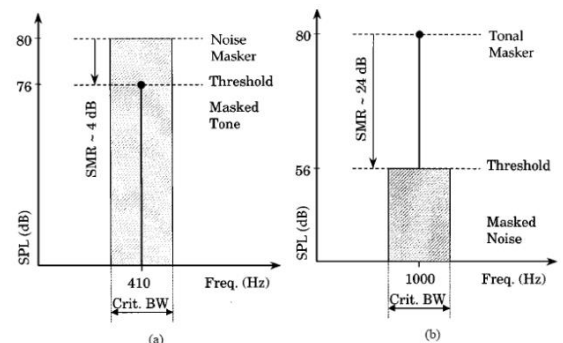
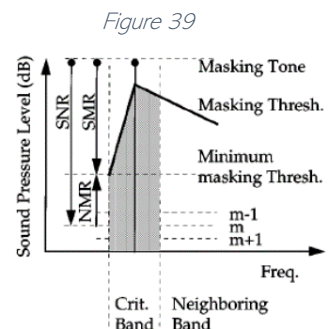
- noise-masking-tone (NMT)
- tone-masking-noise (TMN)
- noise-masking-noise (NMN)

We define the **SMR (Signal-to-Mask ratio)** as the difference between the intensity (SPL) of the masking tone/noise ('signal') and the intensity of the masked tone/noise ('mask'). A positive SMR indicates that the tone is more audible than the masking noise, while a negative SMR suggests that the masking noise dominates and the tone is less audible.

7.2.5.1 Frequency Masking

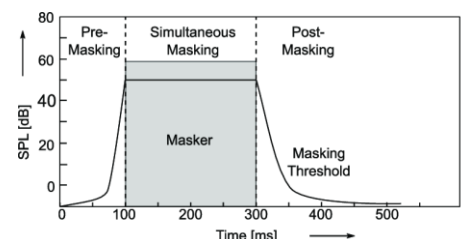
Frequency masking occurs when the presence of a stronger sound, or "masker," makes it difficult to hear a second sound at a similar or neighbouring frequency. This happens because the masker and the masked sound compete at the level of the auditory system. When two sounds are played together, if one is significantly louder (the masker), it can obscure our ability to hear the other sound (the test tone). The point at which the *less loud* tone becomes just audible is known as the threshold of audibility.

- In the **NMT** scenario a narrow-band noise masks a tone within the same critical band, provided that the intensity of the masked tone is below a threshold directly related to the intensity and, to a lesser extent, the center frequency of the masking noise.
- In the case of **TMN** a pure tone occurring at the center of a critical band masks noise of any subcritical bandwidth or shape, provided that the noise spectrum is below a predictable threshold directly related to the strength and, to a lesser extent, to the center frequency, of the masking tone.



7.2.5.2 Temporal Masking

Temporal masking describes the auditory system's delayed response to sounds following a loud noise. After exposure to a loud sound (the masker), the ear requires time to recover and become sensitive again to softer sounds at similar frequencies. This recovery time can involve both a pre-mask period, occurring immediately before the masker is introduced, and a post-mask period, following the cessation of the masker. During these times, our perception of other sounds is 'masked,' and our ability to hear them is diminished.



The concept of auditory masking is used in the development of audio compression algorithms, such as those used in MP3 and AAC encoding, which remove sounds that are likely to be masked and therefore inaudible, allowing for more efficient compression.

7.3 PERCEPTUAL MODELS

Perceptual models, in the context of audio encoding and signal processing, refer to algorithms or frameworks that are designed to approximate the human auditory system's response to sound. The goal of these models is to predict what changes to an audio signal can be made that will be imperceptible or least noticeable to the average listener.

This is crucial in audio compression, where the aim is to reduce the amount of data needed to represent the audio without significantly compromising the perceived quality.

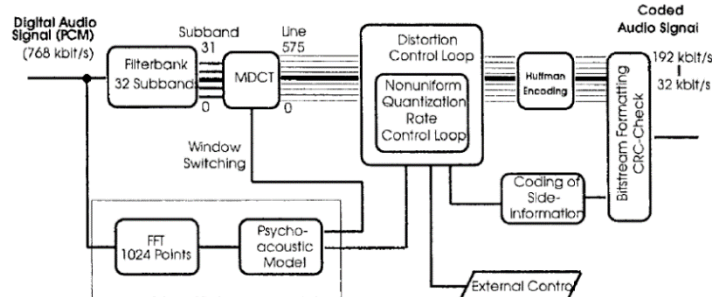
The perceptual model's main task is to estimate the allowed noise level, which is the "just masked threshold". This means the model determines the maximum amount of noise that can be added to the audio signal at each time and frequency location without being detected by the average listener.

The bit allocation according to perceptual criteria works in the following way:

1. For each frequency band, a time-varying Signal-to-Mask Ratio (SMR) is estimated. The SMR is a measure of how much signal is present compared to the noise (masking) threshold at that frequency. A higher SMR means that there is more signal relative to the noise, and therefore, more bits should be allocated to represent this signal accurately.
2. The model then allocates bits to each frequency band based on the SMR. The number of bits allocated to each band i is $R_i = \frac{SMR_i}{6.02dB}$. The constant 6.02 appears because a change of 6.02 dB corresponds to a doubling or halving of the signal power, which in turn corresponds to an increase or decrease of one bit of resolution in a digital signal.

7.4 MPEG-1 PSYCHOACOUSTIC MODEL 1

The MPEG-1 psychoacoustic model 1 is an algorithm used for the perceptual encoding of digital audio. It's part of the MPEG-1 standard, which includes the well-known MP3 audio compression format. This model aims to optimize the encoding process by focusing on the parts of the audio that are most important to the human ear, and minimizing the parts that the ear is less likely to perceive, thus enabling efficient compression without significant loss of perceived audio quality. Here's a complete description of the model using the information provided in the slides:



1. Spectral Analysis and SPL Normalization: the model starts by conducting a spectral analysis to estimate the high-resolution spectrum of the input signal. This analysis breaks down the complex audio signal into its constituent frequencies (spectral components, expressed in SPL). The operation in order are:

- a. The SPL of the input signal is normalized to standardize the spectrum levels across different recordings, ensuring consistent encoding.
- b. The audio input is divided into 12-ms frames (with 512 samples each) using a Hann window that overlaps by 1/16th of the frame. This technique captures the temporal changes in the audio signal.
- c. The Power Spectral Density (PSD) of each time frame is obtained using a Fast Fourier Transform (FFT). The Power Spectral Density (PSD) provides a detailed view of where the signal's power is distributed across different frequencies.

2. Identification of Tonal and Noise Maskers: the identification of tonal and noise maskers is a crucial step for encoding audio data efficiently while maintaining perceived quality. Here's a more detailed explanation of this step:

- Tonal maskers are identified in the PSD by finding peaks or local maxima. These are points where the power at a specific frequency is significantly higher than the power in nearby frequencies. A local maximum is considered a tonal masker if it is at least 7 decibels (dB) louder than its neighboring frequencies within a certain range measured on the Bark scale, which is a psychoacoustic scale that approximates the frequency resolution of human hearing.
- After tonal maskers are identified, the remaining spectral lines are examined to identify noise maskers. Noise maskers are identified in regions where there are no prominent tonal components. These areas may

not have distinct peaks in the PSD, but they can still have enough energy to mask other sounds within their critical band. Any spectral energy that doesn't fall within the neighborhood of a tonal masker is considered as part of a noise masker. Essentially, if it's not a distinct tone (tonal masker), but it's still carrying energy, it's classified as noise.

3. Decimation and Reorganization of Maskers: The process of decimation and reorganization of maskers in the MPEG-1 psychoacoustic model involves simplifying the number of maskers to optimize the encoding process by focusing on sounds that significantly impact perceived audio quality. The model simplifies the number of maskers using two criteria:

1. **Discarding Inaudible Maskers:** Maskers below the absolute threshold of hearing (minimum loudness level that the average human ear can perceive) are discarded.
2. **Comparing and Merging Maskers:** The Bark scale is used to group frequencies into critical bands that correspond to the auditory resolution of the human ear. A sliding window 0.5 Bark wide is moved across the spectrum, comparing the strength of maskers that are close to each other in frequency. Within each window, if two maskers are identified, the quieter one (the one with lower energy) is discarded, and only the stronger masker is retained. This is because, within a critical band, the stronger sound can mask the weaker one, rendering the weaker sound inaudible. Thus, there's no need to encode the weaker sound with as much detail.

This process essentially filters out unnecessary information that does not significantly contribute to the perceived audio quality.

4. Calculation of Individual Masking Thresholds: After the model has identified and simplified the set of maskers (both tonal and noise maskers), it then proceeds to calculate the individual masking thresholds. For each masker identified, the model calculates a threshold of audibility. This threshold represents the sound level below which other sounds will become inaudible when masked by this particular masker. It's a limit below which any additional sound energy in the same critical band will not be perceived by the average human ear. By knowing the individual masking thresholds, the encoder can determine how much it can reduce the bit rate at certain frequencies without the listener noticing a change in quality. Sounds that are below their respective masking threshold will not be encoded with as many bits because the differences would be inaudible under the mask of louder sounds.

5. Calculation of Global Masking Thresholds: After determining the individual masking thresholds for each masker in the signal, the next step is to combine these thresholds to understand the overall masking effect across the audio spectrum. The model treats the masking effects as additive, which means it assumes that the presence of multiple maskers can cumulatively increase the masking effect within a given critical band. These combined effects are used to create a global masking threshold for each frequency bin. A frequency bin is a small range of frequencies, and the entire spectrum is divided into many such bins. The global masking threshold for a frequency bin is the sound level below which a tone would be inaudible due to the masking effects of all the maskers affecting that bin.

The global masking thresholds are used to guide the allocation of bits during the encoding process. They help determine the maximum level of quantization noise that can be added to each frequency bin without the noise being noticeable. By setting the quantization noise level just below the global masking threshold, the encoding process ensures that the noise introduced is imperceptible, thus maintaining the quality of the audio as perceived by the listener.

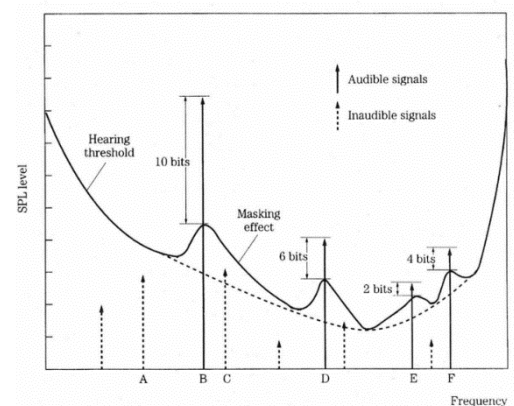


Figure 42

7.5 FILTER BANKS

Filter banks are fundamental components in digital signal processing, particularly in the field of audio coding. They are used to analyze signals in both the time and frequency domains, which is crucial for many forms of audio compression.

A filter bank typically consists of a series of band-pass filters that divide the input signal into multiple components, each one representing a specific portion of the frequency spectrum. This process is essential for codecs that utilize perceptual coding techniques because it allows them to allocate bits more efficiently based on the psychoacoustic model.

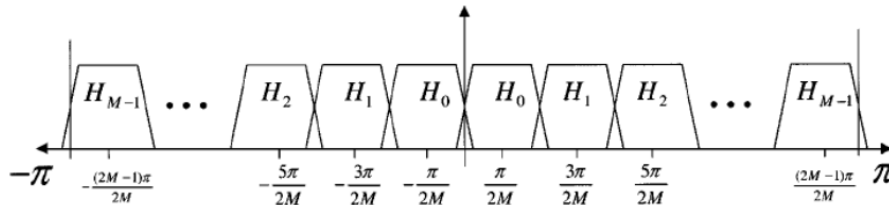


Figure 43

The choice of an appropriate filter bank is critical to the success of a perceptual audio coder: it's necessary to find the trade-off between temporal resolution and frequency resolution. In fact, no single temporal/frequency resolution trade-off is optimal for all signals:

- In **Stationary Harmonic Signals**, where the frequency content does not change over time (es. a sustained note from a musical instrument) high frequency resolution is important because you want to capture the harmonic content very accurately.
- In **Impulse-Like Signals**, that have a short duration and are often of high amplitude (es. a drum hit) high temporal resolution is necessary to accurately capture the rapid changes over time.

7.5.1 Filter Banks Requirements

A number of bank characteristics are highly desirable for filter banks used in audio coding:

- **Signal adaptive time-frequency tiling:** This refers to the filter bank's ability to adapt to the signal's characteristics. For example, it might switch between a low number of bands (e.g., 32) for signals that require more temporal resolution and a high number of bands (e.g., 512) for signals that require more frequency resolution. This adaptability helps to optimize the encoding for different types of audio content.
- **Good channel separation:** The filter bank should be able to isolate the bands well so that there is minimal leakage of audio information between them. Good channel separation ensures that the manipulation of one frequency band does not inadvertently affect adjacent bands.
- **Strong stopband attenuation:** The filter bank should significantly reduce (attenuate) the signal in the stopband regions—frequencies outside the intended passband of each filter. This means that each filter effectively blocks out frequencies it is not responsible for, preventing them from interfering with the analysis.
- **Perfect reconstruction (lossless):** After the signal has been split into different bands, processed, and then recombined, the filter bank should be able to perfectly reconstruct the original signal. This characteristic is essential for codecs that require a lossless format, or for applications where high fidelity is critical.
- **Critical sampling:** This means that the filter bank samples the signal at the Nyquist rate, which is the minimum rate at which the signal can be sampled without introducing aliasing (distortion from undersampling). Critical sampling is efficient because it uses the minimum number of samples necessary to accurately represent the signal.
- **Availability of fast algorithms:** The implementation of the filter bank should be computationally efficient, with fast algorithms that can process the audio in real-time or near real-time, which is essential for many applications such as streaming audio.

7.6 FILTER BANK FAMILIES

7.6.1 Pseudo-Quadrature Mirror Filter (PQMF) filter banks

Pseudo-Quadrature Mirror Filter (PQMF) banks are a type of filter bank used in digital signal processing, specifically in audio coding standards like MPEG-1/2.

They are designed to perform a critical function: breaking down the audio signal into narrower bands for more detailed processing.

In a PQMF filter bank, each individual channel filter is derived from a single prototype filter through a process called modulation. Here's a more detailed explanation of how this works:

- **Prototype Filter $h(n)$:** This is the original low-pass filter from which all channel filters are derived. It's designed to allow frequencies below a certain cutoff while blocking higher frequencies. The length of this filter, denoted by N , determines how sharp the cutoff is between the passed and blocked frequencies.
- **Modulation Process:** To create the different channel filters $h_k(n)$ from the prototype filter, modulation is applied. Modulation in this context involves altering the prototype filter so that it can pass a different range of frequencies. For a filter bank with K channels, k would vary from 0 to $K - 1$. Each value of k results in a different modulation of the prototype filter, shifting its passband to a new range of frequencies.
- **Cosine Modulation:** The actual modulation involves multiplying $h(n)$ by a cosine function. The frequency of this cosine is determined by k , K , and N . This multiplication shifts the frequency response of the prototype filter up and down the frequency spectrum, creating band-pass filters centered around different frequencies.

The filter bank generates $2K$ copies of the prototype filter's frequency response to cover the full range of frequencies in the signal. This extensive coverage is necessary to ensure that all parts of the audio signal are analyzed and encoded.

The PQMF banks approximate the analysis and synthesis process of signals but do not achieve perfect reconstruction. This means that when the signal is split into its component parts (sub-bands) and then recombined, the output is not an exact replica of the original input; there will be some degree of distortion or error. Despite this imperfection, PQMF perform well enough that the distortion is typically below the threshold of perception for human listeners, especially in the context of audio coding for music and speech, where they are commonly used. This is why PQMF banks are utilized in widely adopted standards like MPEG-1 and MPEG-2, where the trade-off between computational efficiency and the quality of signal reconstruction is acceptable for the purposes of audio encoding.

7.6.2 Modified Discrete Cosine Transform (MDCT)

The Modified Discrete Cosine Transform (MDCT) is a transformation technique widely used in audio signal processing, especially for audio compression algorithms like MP3 and AAC. It has several properties that make it well-suited for these applications:

- **Perfect Reconstruction:** Unlike many other transformation methods used in signal processing, the MDCT has the property of perfect reconstruction. This means that, in theory, you can reconstruct the original signal perfectly from the MDCT coefficients without any loss of information. This is ideal for high-fidelity audio applications where quality is paramount.

- No Blocking Artifacts:** The MDCT is designed to avoid blocking artifacts. These are the discontinuities that can occur at the borders between successive blocks of data when they are processed independently, which is common in other block-based transformations. The MDCT's overlapping window technique helps to smooth out these transitions, making the artifacts less noticeable. The MDCT operates on blocks of data with an overlap, typically 50%, which means each block of input samples overlaps the previous block, allowing for a smoother transition between blocks and reducing the risk of blocking artifacts. Despite the overlapping, the MDCT is critically sampled. This means that the transform does not increase the number of samples; it maintains the same rate. For each block of $2M$ samples, the MDCT produces exactly M coefficients. This efficient sampling ensures that the MDCT is not expanding the data, which is crucial for data compression applications.
- Fast Algorithms:** The MDCT can be efficiently implemented using algorithms that are based on the Fast Fourier Transform (FFT), which is a highly optimized process for computing transformations.
- Signal-Adaptive:** The MDCT can be made signal-adaptive, meaning it can adjust its processing based on the characteristics of the input signal. This allows for more efficient coding by adapting to the signal's temporal and spectral content.

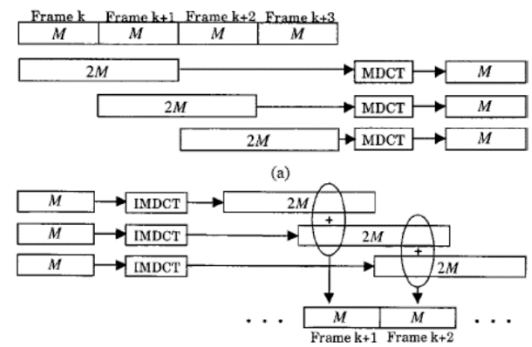


Figure 44

7.6.2.1 Transform Coefficients

The forward MDCT takes an input block of audio samples, $x(n)$, and computes the **transform coefficients**, $X(k)$. This operation is defined by the equation: $X(k) = \sum_{n=0}^{M-1} x(n) * h_k(n)$. Here, $h_k(n)$ represents the basis functions of the MDCT, which are derived from windowed sines and cosines. The index k ranges from 0 to $M - 1$, corresponding to each coefficient in the block of M samples.

The transform coefficients generated by the MDCT are what get quantized and encoded in an audio compression algorithm. The process of quantization and coding is guided by perceptual models, which attempt to preserve the parts of the signal most important to human hearing while allowing for the discarding of parts that are less likely to be noticed.

7.6.2.2 Reconstructed Samples

The inverse MDCT uses the transform coefficients, $X(k)$, to reconstruct the original audio samples, $x(n)$, with the equation: $\sum_{k=0}^{M-1} [X(k) * h_k(n) + X'(k) * h_k(n + M)]$. This equation ensures that the original block of samples can be perfectly reconstructed from the MDCT coefficients, provided the overlapping blocks are handled correctly.

For the MDCT to achieve perfect reconstruction, certain conditions must be met regarding the window function $w(n)$. These conditions ensure that the sum of the squared window function over two consecutive blocks (*adding up the energy contribution from both blocks at each point in time*) is equal to 1, which is a requirement for preventing aliasing and ensuring that the overlap-add process results in a perfect reconstruction of the original signal.

7.6.3 Adaptive Filter Banks

Adaptive filter banks are a sophisticated technology used in signal processing to dynamically alter the time-frequency analysis of signals, particularly in audio encoding. Unlike traditional filter banks, which have a fixed time/frequency decomposition, adaptive filter banks can adjust the resolution of this decomposition over time to suit the varying characteristics of the input signal. The decision to change the window length is driven by a perceptual model, which takes into account how humans hear and process audio. The perceptual model determines

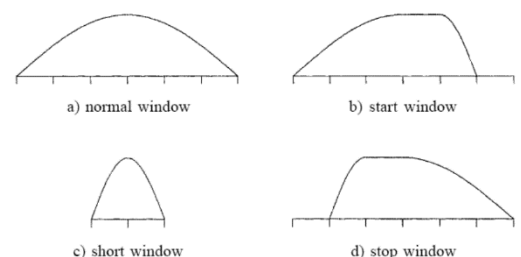


Figure 45

whether the signal at a given moment requires better time resolution or frequency resolution. The adaptive system switches between different window lengths:

- A **long window** is used for stationary signals where frequency resolution is more important.
- A **short window** is used when the signal has transient elements, and increased time resolution is required.
- **Start and stop windows** are special windows used to transition between long and short windows smoothly, ensuring no sudden jumps or artifacts are introduced during the switch.

7.6.4 Hybrid filter Banks

Hybrid filter banks combine different types of filter banks in a cascade, meaning one after the other.

In the hybrid filter bank system, the approach begins with a **polyphase filter bank**. This stage is responsible for breaking down the audio signal into multiple bands, which allows for initial frequency separation. Each band is then processed further for more detailed frequency analysis.

The **MDCT** plays a critical role in this detailed analysis by providing a finer frequency resolution. This is where the hybrid nature of the filter bank shines, as it combines the polyphase filter bank's broad initial separation with the MDCT's precise spectral analysis.

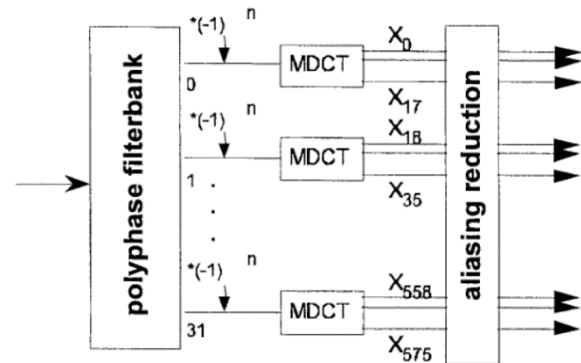


Figure 46

By employing window switching techniques, the filter bank can dynamically adjust the MDCT's window length. The perceptual model drives this adaptive process. It intelligently decides when to switch between long and short windows, optimizing the codec's performance based on the psychoacoustic properties of human hearing. This ensures that the audio is encoded in a way that preserves the most critical auditory details while allowing for efficient data compression.

Despite the sophistication and benefits that hybrid filter banks bring to audio encoding—allowing for variable frequency resolutions and adaptively switching to accommodate the signal's nature—their complexity is a notable drawback. Implementing such systems requires advanced algorithms and can be computationally intensive.

7.7 QUANTIZATION IN AUDIO CODING

Quantization is the process of mapping a large set of input values to a smaller set. In the context of audio coding, it involves reducing the precision of the signal's representation by mapping continuous amplitude values to a limited number of levels. This is a lossy process, meaning that some of the signal's precision is lost, but if done carefully, it can significantly reduce the amount of data required to store or transmit the audio without noticeably affecting the perceived quality. Most systems apply uniform quantization, where each step size between quantized levels is the same across the entire range. However, some systems, like MPEG-1 and MPEG-2 Audio, use non-uniform quantization following a power law, which may allocate more levels to certain ranges of amplitudes based on their auditory importance.

In non-uniform quantization, a crucial process is the Bit allocation, described in the next paragraph.

7.7.1 Bit Allocation

The process of bit allocation is crucial in determining how many bits are assigned to each part of the signal. This is typically done before quantization and can be based on data statistics or a perceptual model. The perceptual model assesses which parts of the audio are more important to the listener and allocates more bits to those parts, ensuring that the most audible parts of the signal are encoded with the highest precision.

There are two approaches to bit allocation:

1. **Direct Structure (Bit Allocation):** This method involves strategizing how to distribute a finite number of bits among various components of an audio signal **before the quantization process begins**. The allocation is guided by a perceptual model, a sophisticated framework that simulates the way human ears perceive sound. The model identifies which parts of the audio spectrum are most sensitive to the ear and allocates more bits to these areas, since any distortion or noise in these regions would be most noticeable to listeners. Conversely, fewer bits are allocated to parts of the spectrum that are less audible, thus optimizing the overall perceived audio quality. This is a proactive approach, as it anticipates the perceptual impact of quantization before it happens.
2. **Indirect Structure (Noise Allocation):** Contrary to the direct structure, the indirect structure **first quantizes the data and then assesses how the quantization has affected the signal, particularly in terms of the noise introduced by quantization. After this initial quantization, the algorithm adjusts the number of bits allocated to different components based on the amount of noise each component has and how perceptible that noise would be** according to a perceptual model. The aim is to mask the quantization noise, keeping it below the thresholds at which the human ear would detect it, known as the masking thresholds.

7.8 ENTROPY CODING IN AUDIO CODING

Entropy coding is a method of lossless compression that involves encoding the quantized values in a way that reduces the number of bits required to represent them. It takes advantage of the statistical distribution of the quantized values, assigning shorter codes to more frequent values and longer codes to less frequent ones. Huffman coding and arithmetic coding are two common entropy coding techniques.

8 IMAGE CODING

Image coding is the process of converting an image into a digital format that can be efficiently stored or transmitted by sampling, quantizing, transforming, compressing, and encoding the visual information into a compact binary form. This allows for the reduction of file size while maintaining image quality, facilitating easy storage and transmission.

8.1 DIGITAL IMAGES

A digital image is a two-dimensional array of samples denoted by $x[n1, n2]$, where $n1$ and $n2$ are spatial coordinates within the image dimensions $N1$ and $N2$, respectively. This array represents the pixel values across the image's width and height.

The pixel values in a digital image are obtained through the quantization of real-valued images. This process involves scaling the real-valued intensities ($x'[n1, n2]$ in the range $[0, 1]$ or $[-1/2, +1/2]$) by 2^B and then rounding to the nearest integer to produce quantized pixel values. Here, B represents the number of bits used per pixel, and the rounding operator is applied to each value. Digital images use a finite number of bits (B) to represent each pixel value. There are two types:

- **Unsigned B-bit imagery:** Pixel values range from 0 to $2^B - 1$.
- **Signed B-bit imagery:** Pixel values range from -2^{B-1} to $2^{B-1} - 1$.

Typically, $B = 8$ for standard images, but higher values of B may be used for increased dynamic range in applications like medical imaging, military, or scientific purposes.

For color images, each pixel is typically represented by three values corresponding to the red, green, and blue (RGB) components ($x_R[n1, n2]$, $x_G[n1, n2]$, $x_B[n1, n2]$). This allows for the representation of a wide range of colors through the combination of these primary colors.

8.2 LOSSLESS CODING IN DIGITAL IMAGES

Lossless coding in digital images is a method of data compression designed to minimize the number of bits required to represent an image while ensuring that there is no loss of information. This means that every single bit of data (all B bits) in the original image sample can be perfectly reconstructed from the compressed image data.

The compression achieved through lossless coding is typically limited compared to lossy compression, as it does not discard any information. Lossless coding is essential in applications where the exact original data is required, such as:

- Binary images (facsimile), where only two possible values are present for each pixel, typically used in scanned documents.
- Medical images, where the precise details are crucial for diagnosis and any loss of data can be unacceptable.
- Master copies of images that will be edited later, as loss of detail could compound through editing processes.

In these scenarios, the integrity of the original image is of utmost importance, and lossless coding ensures that the original and the decompressed images are bit-for-bit identical.

8.3 LOSSY CODING IN DIGITAL IMAGES

Lossy coding in image coding is a compression technique that allows for a reduction in the number of bits required to store or transmit digital images at the cost of some loss of information, leading to a deviation or distortion from the original image. This distortion can be acceptable because the human visual system may not perceive the loss or may tolerate it, and because the digital representation of real-world scenes is often an imperfect approximation to begin with.

The extent of the distortion introduced by lossy coding can be quantified using metrics such as:

- **Mean Square Error:** $MSE = \frac{1}{N_1 * N_2} * \sum_{n_1=0}^{N_2} \sum_{n_2=0}^{N_2} (x[n_1, n_2] - \hat{x}(n_1, n_2))^2$.
- **Peak Signal to Noise Ratio:** $PSNR = 10 * \log_{10} \frac{(2^B - 1)^2}{MSE} dB$. This equation expresses the ratio of the maximum possible pixel value of the image to the power of the distortion noise. The PSNR is often used to measure the quality of reconstruction of lossy compression codecs.

In lossy coding, the measure of compression can be expressed in terms of:

- **Compression ratio:** $C_{ratio} = \frac{N_1 * N_2 * B}{||C||}$. It compares the number of bits required to store the uncompressed image to the number of bits required after compression (bit-stream C of length $||C||$).
- **Bitrate:** $Bit_{rate} = \frac{||C||}{N_1 * N_2} \left[\frac{bit}{pixel} \right]$. It is more meaningful than compression ratio since the bit depth (B) can be somewhat arbitrary for lossy compressed images.

Lossy compression is especially effective for natural images and motion video, where much higher compression ratios can be achieved compared to lossless coding. Common formats that use lossy compression include JPEG for images and MPEG for videos.

Typical bitrates after lossy compression depend on the content of the image and the desired quality level, with higher quality requiring more bits per pixel and lower quality allowing fewer bits per pixel. For example, a high-quality lossy compressed image may have a bitrate of 1 bpp, a moderate quality image may have 0.5 bpp, and a usable quality image may have 0.25 bpp. These bitrates are substantially lower than what is typically achieved with lossless compression.

8.4 REDUNDANCY AND IRRELEVANCY

Compression techniques work by exploiting **statistical redundancy** and introducing acceptable deviation using **visual irrelevancy**.

8.4.1 Statistical Redundancy

Statistical Redundancy refers to the presence of patterns and structures in the image data that can be used to represent the image more efficiently. Redundancy can be seen in the repetition of pixel values or colour patterns within an image. For example, in a picture of the sky, large areas might be a similar shade of blue.

Statistical redundancy is related to the predictability of pixel values. If pixel values are not equally likely, or if there is a correlation between adjacent pixel values, then the image contains statistical redundancy that can be exploited for compression. For example, adjacent pixels in an image often have similar or identical values, which means the information can be compressed by encoding the differences rather than the absolute values. The joint entropy of two adjacent pixels will generally be less than the sum of their individual entropies, allowing for a more efficient representation (*entropy is a measure of the unpredictability or complexity of the data. An image with lower entropy can be compressed more because it has more predictability or less complexity*).

By converting from RGB to a YC_bC_r color space, the statistical redundancy can be exploited more effectively as it often reduces the correlation between the color channels, thereby reducing the overall entropy of the image components.

Lossless coding utilizes only statistical redundancy by encoding the image data in a way that all original pixel values can be perfectly reconstructed.

8.4.2 Visual Irrelevancy

Visual Irrelevancy refers to the data within an image that can be omitted without significant loss of perceived image quality. This concept is based on the limitations of human vision, such as the inability to perceive certain frequencies of colour or brightness variations. The human visual system is less sensitive to colour hue and saturation changes than to brightness variations. This is why colour space transformations are often used in compression to separate luminance (brightness) information from chrominance (color) information.

$$\begin{bmatrix} X_Y \\ X_{C_b} \\ X_{C_r} \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.0813 \end{bmatrix} * \begin{bmatrix} X_R \\ X_G \\ X_B \end{bmatrix} \text{ where } \begin{cases} X_R, X_G, X_B \text{ are the colour component in RGB representation.} \\ X_Y \text{ is the luminance component in } YC_bC_r \text{ representation.} \\ X_{C_b}, X_{C_r} \text{ are the chrominance components.} \end{cases}$$

The luminance component Y represents the brightness levels in the image, while the chrominance components C_b and C_r carry colour information. Due to the visual system's properties, the chrominance components can often be sub-sampled without significantly impacting the perceived image quality (C_b and C_r components are usually sub-sampled 2:1 relative to Y). This means that the resolution for colour information can be lower than the resolution for brightness information, exploiting visual irrelevancy to achieve better compression rates.

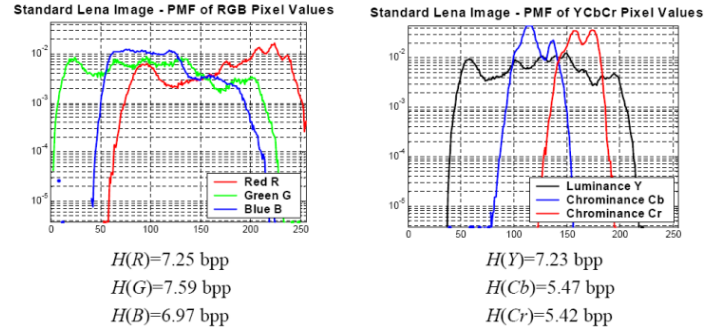


Figure 47

8.4.3 $RGB \rightarrow YC_bC_r$ conversion in MATLAB

```
%% YCrCb conversion.
x = double(x); % Convert to double precision to avoid errors.
R = x(:, :, 1);
G = x(:, :, 2);
B = x(:, :, 3);
YCrCb_matrix = [0.299, 0.587, 0.114; -0.169, -0.331, 0.5; 0.5, -0.419, -0.0813];
Y = YCrCb_matrix(1,1)*R + YCrCb_matrix(1,2)*G + YCrCb_matrix(1,3)*B;
Cr = YCrCb_matrix(2,1)*R + YCrCb_matrix(2,2)*G + YCrCb_matrix(2,3)*B;
Cb = YCrCb_matrix(3,1)*R + YCrCb_matrix(3,2)*G + YCrCb_matrix(3,3)*B;
```

8.5 IMAGE COMPRESSION SYSTEMS

Most of the image coding scheme known in the literature, including the standards JPEG and JPEG2000 follow the architecture shown in figure 48.

1. **Transform $T(x)$:** The original image x is first passed through a transform function T , which converts the image into a set of coefficients y that represent the image in an alternate domain, such as the frequency domain. This step is usually reversible (invertible), meaning the original image can be reconstructed from these coefficients without loss if no further steps introduce irreversibility. Common transforms include the Discrete Cosine Transform (DCT), used in JPEG compression. The transform helps to deal with inter-pixel redundancy by concentrating the image's energy into fewer coefficients, which are easier to compress.

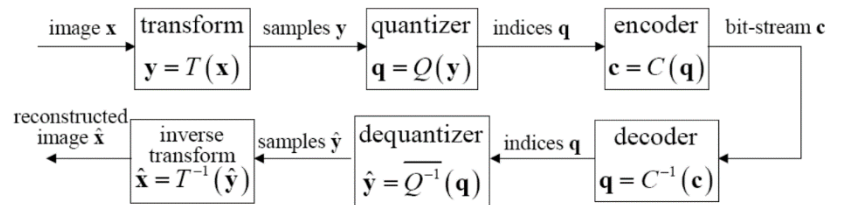


Figure 48

2. **Quantization $Q(y)$:** The samples y from the transform are then quantized using a quantization function Q . This process maps the continuous or large set of values into a smaller set, which can be represented with fewer bits. Quantization is not invertible; it introduces distortion because it approximates the transform coefficients, effectively reducing the precision of the image representation. This step is used only in lossy compression and exploits visual irrelevance by discarding information that is less important to human vision.
3. **Entropy Coding $C(q)$:** After quantization, the indices q (the output of the quantization step) are further compressed using an entropy coding function C . Entropy coding is a reversible process that encodes the quantized values into a bit-stream c in a way that commonly occurring values use fewer bits and less common ones use more bits. This step takes advantage of statistical redundancy within the quantized

values. The goal is to reduce the number of bits needed to represent each quantized value, therefore compressing the image data.

4. **Decoding Process $C^{-1}(c)$:** To reconstruct the image, the compressed bit-stream c is passed through the decoder C^{-1} , which is the inverse of the entropy coding function. The output q is then dequantized by the dequantizer Q^{-1} to approximate the original transform coefficients y . Finally, the inverse transform T^{-1} is applied to the dequantized coefficients $\tilde{y}$ to produce the reconstructed image $\tilde{x}$.

8.6 2D TRANSFORMS IN IMAGE CODING

In image coding, 2D transforms are mathematical operations applied to the two-dimensional array of an image's pixel values to transform the image **from its spatial domain** (where the values represent pixel intensities at specific coordinates) into a different domain, often the **frequency domain**. This transformation process facilitates the compression of the image by enabling the algorithm to operate on the image data more efficiently.

The forward 2D transform $T(k_1, k_2)$ is applied to the original image $x[n_1, n_2]$ of size $N_1 * N_2$. The transform function generates a two-dimensional array of samples y , which represents the image in the transformed domain.

The forward 2D transform can be expressed through the general equation:

$$T(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] * g(n_1, n_2, k_1, k_2) \text{ where } \begin{cases} T(k_1, k_2) \text{ is the frequency - domain image.} \\ x[n_1, n_2] \text{ is the original space - domain image.} \\ g(n_1, n_2, k_1, k_2) \text{ is the function that maps } [n_1, n_2] \text{ to } (k_1, k_2). \end{cases}$$

The inverse transform used to reconstruct the image from the transformed coefficients is:

$$x(n_1, n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} T[k_1, k_2] * h(n_1, n_2, k_1, k_2) \text{ where } \begin{cases} T(k_1, k_2) \text{ is the frequency - domain image.} \\ x[n_1, n_2] \text{ is the original space - domain image.} \\ h(n_1, n_2, k_1, k_2) \text{ is the function that maps } (k_1, k_2) \text{ to } [n_1, n_2]. \end{cases}$$

8.6.1 2D Transforms Properties

8.6.1.1 Symmetry

A transform is symmetric if the two-dimensional basis function is the product of the same one-dimensional function in both dimensions:

$$g(n_1, n_2, k_1, k_2) = g_1(n_1, k_1) * g_2(n_2, k_2)$$

This symmetry can further simplify the computation and the implementation of the transform.

In the next paragraphs we will analyse the most common type of transforms.

8.6.1.2 Separability

A 2D transform is separable if the basis function $g(n_1, n_2, k_1, k_2)$ can be written as the product of two one-dimensional functions:

$$g(n_1, n_2, k_1, k_2) = g_1(n_1, k_1) * g_2(n_2, k_2)$$

This property is beneficial because it allows the 2D transform to be computed as two successive 1D transforms, which can significantly reduce the computational complexity.

2D separable transforms can be conveniently expressed in matrix form. In this way, the transformation of an $N \times N$ block of an image can be represented as

$$y = A * x * A^T \text{ where } \begin{cases} x \text{ is the } N \times N \text{ matrix representing the time domain input signal.} \\ y \text{ is the } N \times N \text{ matrix representing the freq. domain signal.} \\ A \text{ is the matrix of size } N \times N \text{ that contains the basis function of the transform.} \end{cases}$$

and the inverse transformation becomes: $x = A^T * y * A$.

Since A is orthonormal, the inverse transform matrix is simply the transpose of A , leading to the inverse operation.

By using separable transforms, the complexity of the transformation is reduced from $O(N^4)$ to $O(N^3)$, because two $N \times N$ matrix multiplications are performed instead of a $[1xN^2] * [N^2xN^2]$ multiplication. This computational efficiency is significant in image processing, especially for large images or real-time applications.

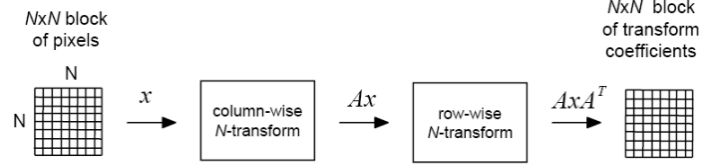


Figure 49

8.6.2 Two important 2D Transforms: DFT and DCT

The Discrete Fourier Transform (DFT) and the Discrete Cosine Transform (DCT) are two common transforms used in the field of image processing, particularly for image compression.

8.6.2.1 Discrete Fourier Transform (DFT)

The DFT is used to analyze the frequency content of signals, including images. For a two-dimensional signal (like an image), the 2D DFT basis function g is given by the formula:

$$g(n_1, n_2, k_1, k_2) = \frac{1}{N_1} * e^{-j * \frac{2\pi}{N_1} * n_1 * k_1} * \frac{1}{N_2} * e^{-j * \frac{2\pi}{N_2} * n_2 * k_2} \text{ where } \begin{cases} n_1 \text{ and } n_2 \text{ are spatial domain indices.} \\ k_1 \text{ and } k_2 \text{ are frequency domain indices.} \\ N_1 \text{ and } N_2 \text{ are the image block dimensions.} \end{cases}$$

The DFT transforms the spatial domain signal into the frequency domain, representing the image as a sum of sinusoids of varying magnitudes, frequencies, and phases. This transform is complex-valued and **contains both real and imaginary parts**, which represent the amplitude and phase of the frequency components.

The inverse 2D DFT uses a similar formula but with a positive exponent, allowing the original image to be reconstructed from its frequency representation: $h(n_1, n_2, k_1, k_2) = e^{j * \frac{2\pi}{N_1} * n_1 * k_1} * e^{-j * \frac{2\pi}{N_2} * n_2 * k_2}$.

8.6.2.2 Discrete Cosine Transform (DCT)

The DCT is similar to the DFT but uses only cosine functions. It's particularly well-suited for image compression due to its ability to pack most of the signal's energy into a small number of coefficients. The 2D DCT is expressed as:

$$g(n_1, n_2, k_1, k_2) = h(n_1, n_2, k_1, k_2) = a(k_1) * \cos\left(\frac{(2 * n_1 + 1) * \pi * k_1}{2 * N_1}\right) * a(k_2) * \cos\left(\frac{(2 * n_2 + 1) * \pi * k_2}{2 * N_2}\right)$$

Where $a(k)$ is a scaling factor that equals $\sqrt{\frac{1}{N}}$ when $k = 0$ and $\sqrt{\frac{2}{N}}$ for $k = 1, \dots, N - 1$. This ensures that the transform matrix is orthonormal. The **DCT is real-valued**, making it more suitable for images that have real-valued pixels.

8.6.2.3 DFT vs. DCT

The DCT is often preferred over the DFT for image coding for several reasons:

- The DCT transforms real pixel values into real coefficients, which is computationally simpler and more intuitive to work with when dealing with images that have real-valued intensities.
- The DCT avoids the boundary artifacts that can be introduced by the DFT. In the DFT, the image is conceptually repeated across its boundaries, leading to discontinuities at the edges, which can manifest as ringing artifacts in the spatial domain. The DCT, by contrast, assumes the signal is even around the boundaries, which often leads to better visual results.

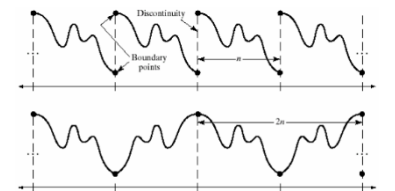


Figure 50

The choice of transform affects the coding gain, which is a measure of how well the transform can compact the energy of the signal. A higher coding gain means more of the image's energy is concentrated in fewer coefficients, which is desirable for compression. The DCT is known for its high coding gain, especially for signals with high spatial correlation like images. This is why it's widely used in standards like JPEG.

8.6.3 DCT for Image Coding in MATLAB

```
%% First luminance's block 8 x 8 DCT coefficients.
Y_crop = Y(1:8, 1:8); % isolate the first block.
N = 8;
% Initialize the DCT transformation matrix.
l = 1:N;
k = 1';
T = sqrt(1/N)*cos(pi/(2*N)*(k-1)*(2*l-1));
T(2:end,:) = T(2:end,:)*sqrt(2);
transf_block = T * Y_crop * T';

%% Visualize the 2D-DCT transform basis.
Subplot_position_counter = 1;
basis = zeros(size(Y_crop));
for i = 1 : N %N = dimension of NxN blocks.
    for j = 1 : N
        basis = T(i,:)'*T(j,:);
        subplot(N,N,counter);
        imagesc(basis);
        counter = counter+1;
    end
end
```

8.6.4 Choice of Block Size

When applying transforms like the DCT, images are typically partitioned into smaller, non-overlapping blocks. These blocks are processed independently. The size of these blocks is a significant factor in the design of a compression algorithm because it influences the trade-off between efficiency and complexity.

Advantages of Larger Blocks: Larger blocks can lead to decreased blocking artifacts. This is because using larger blocks can help in achieving smoother transitions between them (*blocking artifacts are the visible boundaries that can appear between blocks when an image is compressed at high ratios and then decompressed*). A larger block size can result in increased coding gain, which is the concentration of the image's energy into a smaller number of coefficients. This is beneficial for compression, as it can lead to more significant reductions in data size while maintaining the integrity of the image's visual information.

Disadvantages of Larger Blocks: However, increasing the block size also increases computational complexity. Larger blocks require more calculations for both the forward and inverse transform, which means more processing time and potentially greater resource usage, especially in hardware-limited environments.

8.6.5 Quantization of the Transform Coefficients

Quantization allows the transform coefficients to be represented with a finite number of bits. This reduces the amount of data required to store each coefficient, contributing to the compression of the image. As a result of quantization, some of the coefficients are approximated, or "coarsely reconstructed," while others may be set to zero. This means that less important information (*often high-frequency details*) is discarded, which can lead to a loss of some image detail. Effective quantization requires understanding the statistical distribution of the transform coefficients. For example, with the DCT, coefficients are mostly uncorrelated, which allows for each coefficient to be quantized independently:

- The DC coefficient (the first coefficient in a DCT block, representing the average color) is typically uniformly distributed.
- The AC coefficients (all other coefficients, representing changes in pixel values) tend to follow a Laplacian probability density function. This knowledge is used to design quantization tables that more heavily quantize coefficients with a distribution indicating they have less visual impact.

8.7 ZONAL CODING

Zonal coding is a technique used in image compression, particularly following a transformation like the Discrete Cosine Transform (DCT). The method involves selectively encoding transform coefficients based on their importance to the overall image quality.

Here's a detailed explanation of zonal coding:

- **A Priori Decision on Coefficients:** Zonal coding operates on the principle of deciding in advance ("a priori") which DCT coefficients are significant enough to be encoded. This decision is typically based on the expected visual impact of each coefficient. Coefficients that contribute more to the perceived image quality are prioritized for encoding. The importance of each coefficient is estimated by calculating its variance. A higher variance for a coefficient means that it has a more significant impact on the image's content and should therefore be preserved in the encoding process.
- **Zonal Mask:** A zonal mask is used to determine which coefficients are encoded. This mask is essentially a pattern or map that indicates which parts of the DCT block are more important. The mask is designed based on the understanding that lower-frequency coefficients (usually found in the upper-left corner of the DCT block) are generally more important for image reconstruction, while higher-frequency coefficients (toward the bottom-right) can often be discarded without substantially affecting image quality.
- **Bit Allocation Mask:** A bit allocation mask is also employed, which specifies the number of bits allocated to each DCT coefficient that is deemed important enough to be encoded. The allocation of bits is not uniform; more bits are given to coefficients with higher variance, as they are more important for accurately representing the image.

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

8	7	6	4	3	2	1	0
7	6	5	4	3	2	1	0
6	5	4	3	3	1	1	0
4	4	3	3	2	1	0	0
3	3	3	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure: Zonal coding. Left: zonal mask. Right: bit allocation

Figure 51

The entire process of zonal coding, including the design of the zonal and bit allocation masks, depends on the estimated variances of the DCT coefficients. Coefficients with higher variances are allocated more bits, and the zonal mask is created to cover areas with these significant coefficients.

8.8 THRESHOLD CODING

Threshold coding is an adaptive image compression technique designed to address some limitations of zonal coding by determining which DCT coefficients to encode based on a threshold.

Limitations of Zonal Coding

Zonal coding uses a fixed mask to encode certain coefficients across all image blocks, making it a non-adaptive approach that relies on statistics collected during an offline training process. This means that zonal coding treats all image blocks with the same importance across the entire image, which can be inefficient for images with varying content.

How Threshold Coding addresses these limitations

Threshold coding enhances these limitations by being adaptive. It determines whether to encode each DCT coefficient based on if it exceeds a certain threshold. This approach adjusts to the image content dynamically, allowing for more flexibility in encoding.

Here is a step-by-step explanation of how threshold coding works:

1. **Uniform Scalar Quantization:** All DCT coefficients are initially quantized uniformly, meaning they are all divided by a fixed quantization step size. This step size is typically determined by a quantization matrix which can assign different step sizes to different coefficients, depending on their importance to image quality. The quantization process effectively reduces the precision of the coefficients, which is a form of lossy compression.

2. **Threshold Application:** A threshold is set to determine the significance level of the DCT coefficients. If a coefficient's absolute value is below this threshold, it is considered insignificant to the overall image quality and is set to zero. This step reduces the number of coefficients that need to be encoded, thereby reducing the amount of data that will be stored or transmitted.
3. **Recording Non-Zero Coefficients:** The positions (coordinates) of the non-zero coefficients are recorded. This step is crucial because, during the decompression phase, the decoder needs to know where to place these non-zero coefficients within the DCT block. Along with their positions, the amplitude values of the non-zero coefficients are also kept for the reconstruction of the image.
4. **Quantization Index Calculation:** For each non-zero DCT coefficient, a quantization index is computed. This index is derived by dividing the coefficient's value by the product of a global quantization parameter and the value from the quantization matrix corresponding to that coefficient. The result is then rounded to the nearest integer. This index will be used during the encoding process to represent the coefficient's value in a compressed form.
5. **Quantization Matrix Adjustment:** The quantization matrix, which controls the step sizes for different coefficients, can be adjusted to fine-tune the compression. For example, higher quantization steps can be used for coefficients that the human eye is less sensitive to (typically high-frequency components), and lower steps for more visually significant components (low-frequency components). This part of the process can incorporate psychovisual considerations, acknowledging that the human visual system is more sensitive to quantization noise at low frequencies. Therefore, less quantization (smaller step sizes) is applied to low-frequency coefficients to maintain image quality.

8.9 JPEG

The JPEG algorithm is a widely used method for digital image compression. The algorithm involves several steps to efficiently compress images. Here is a detailed description of the JPEG compression algorithm:

1. **Prepare the image:** The JPEG compression process starts by preparing the *input image data*:

- **Colorspace Conversion:** The image is converted from the *RGB* colorspace to the *YCbCr* colorspace. This step separates the image into a luminance component (*Y*) and two chrominance components (*C_b* and *C_r*). The *Y* component represents the brightness levels in the image, while *C_b* and *C_r* represent the colour information.
- **Level Offset:** If the image is quantized using *B* bits, an offset equal to 2^{B-1} is subtracted from each pixel value of each colour matrix (*Y*, *C_r*, *C_b*). This step is known as level shifting and is done to *center the pixel values around zero*, which is necessary for the subsequent DCT process.
- **Subsampling of Chrominance Components:** The *C_b* and *C_r* components are often subsampled because the human eye is less sensitive to fine color details than to fine brightness details. Subsampling reduces the resolution of the chrominance information, which contributes to the compression process.

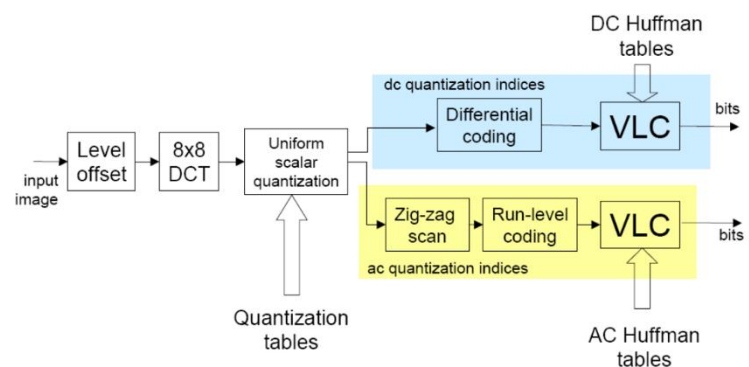


Figure 52

2. **Partitioning and DCT:** Each colour matrix is then partitioned into four non-overlapping 8x8 blocks. For each block, the DCT (Discrete Cosine Transform) is applied, which converts the spatial domain data into frequency domain data.
3. **Threshold Quantization:** Each DCT coefficient is quantized using a uniform scalar quantizer. This involves using quantization matrices that define how much each coefficient should be scaled down. The quantization matrices can vary for different colour components, and they are designed to be more aggressive for higher frequency components which are less visually significant.

4. **Coding:** this step is different for DC and AC coefficients:
 - **DC coefficients:** For the DC coefficient (*the first coefficient in each block, which represents the average colour of the block*), **Differential Coding** is used. The DC coefficient of each block is encoded as the difference from the previous block's DC coefficient, a method known as [Differential Pulse Code Modulation \(DPCM\)](#).
 - **AC Coefficients:** The AC coefficients (*all other coefficients in the block*) are ordered in a **zig-zag sequence** to group non-zero coefficients together. This scanning order prioritizes lower-frequency components over higher-frequency ones. After the zig-zag scan, the AC coefficients are **run-level encoded**, which combines the sequence of coefficients with a scheme that encodes the length of runs of zeros for further compression.
5. **Variable Length Coding (VLC):** The VLC (Variable Length Coding) blocks apply the Huffman tables to the DC and AC quantization indices. VLC is a statistical coding method that assigns shorter codes to more frequent sequences, which results in compressed bitstreams for both DC and AC components.

8.10 JPEG MATLAB IMPLEMENTATION

8.10.1 JPEG compression (DCT)

```

%% JPEG compression (DCT) of the luminance component.
B = 8;
load('Qjpeg.mat'); %(NxN matrix indicating coeff. quantizing precision).
s_factor = 1; % factor that increase/reduce the amount of quantization.
quant_matrix = QJPEG.* s_factor;
% 1) Split the image in YCrCb format.
Y = YCrCb_matrix(1,1)*R + YCrCb_matrix(1,1)*G + YCrCb_matrix(1,1)*B;
% 2) Level offset.
Y_shifted = Y-2^(B-1);
% 3) Divide the image into blocks.
JPEG_Y = zeros(size(Y_shifted));
for i = 1 : N : length(Y_shifted(1,:))-(N-1)
    for j = 1 : N : length(Y_shifted(:,1))-(N-1)
        curr_block = Y_shifted(i:i+N-1 , j:j+N-1);
        % 4) Transform coding (DCT).
        l = 1:N;
        k = 1';
        T = sqrt(1/(2*N))*cos(pi/(2*N)*(k-1)*(2*l-1));
        T(2:end,:) = T(2:end, :)*sqrt(2);
        transformed_block = T * curr_block * T';
        % 5) Threshold quantization: y = round(x(k1,k2)/Q(k1,k2)).
        quantized_block = round(transformed_block./quant_matrix);
        % 6) Entropy coding and encoding. We don't do that here (e menomale).
        % 7) Inverse quantization: re-multiply the quantized values for Q.
        deQuantized_block = quantized_block.*quant_matrix;
        % 8) Inverse transform T^-1.
        deTransformed_block = T'*deQuantized_block*T;
        % 9) Put the blocks together.
        JPEG_Y(i:i+N-1 , j:j+N-1) = deTransformed_block;
    end
end
10) Remove level offset.
JPEG_Y = JPEG_Y + 2^(N-1);

```

8.10.2 JPEG compression (KLT)

```
%% JPEG compression (KLT) of the luminance component.
% 1) Split the image in YCrCb format.
Y = YCrCb_matrix(1,1)*R + YCrCb_matrix(1,1)*G + YCrCb_matrix(1,1) * B;
% Calculate the number of blocks as "total_area/block_area".
numblocks = floor( (length(Y(1,:)))*length(Y(:,1)) / (N*N) );
% Reshape the matrix so that it has the blocks along the columns.
Y_blocks_along_columns = im2col(Y, [N,N], 'distinct');
% Initialize the Cxx matrix
Cxx = zeros(N*N, N*N, numblocks); % each cube "slice" is a 8x8 block.
% Calculate the Cxx for each block:
avrg = mean(Y(:));
for i = 1 : numblocks
    Cxx(:,:,i) = (Y_blocks_along_columns(:, i)-avrg) *
    (Y_blocks_along_columns(:, i)-avrg)';
end
% average along the third dimension.
Cxx = mean(Cxx,3);
% Take the eigenvectors_matrix = T'.
[eig_matrix, ~] = eig(Cxx);
T = eig_matrix';
% Start with the JPEG algorithm.
% 2) Level offset.
Y_shifted = Y-2^(B-1);
JPEG_Y = zeros(size(Y));
% 3) Divide the Y matrix into NxN blocks:
for i = 1 : N : length(Y(1,:)) - (N-1)
    for j = 1 : N : length(Y(:,1)) - (N-1)
        curr_block = Y_shifted(i:i+N-1,j:j+N-1);
        % 4) Transform Coding (KLT).
        transformed_block = T * curr_block(:);
        transformed_block = reshape(transformed_block,N,N);
        % 5) Threshold Quantization.
        quantized_block = round(transformed_block./quant_matrix);
        % 6) Encoding part (not implemented).
        % 7) De-quantize the signal.
        deQuantized_block = quantized_block .* quant_matrix;
        % 8) Invert the KLT transformation.
        deTransformed_block = T'*deQuantized_block(:);
        deTransformed_block = reshape(deTransformed_block, N,N);
        % 9) Put the blocks together.
        JPEG_Y(i:i+N-1, j:j+N-1)=deTransformed_block;
    end
end
end
10) Remove level offset.
JPEG_Y = JPEG_Y + 2^(N-1);
```

9 VIDEO CODING

Video coding refers to the process of compressing and decompressing digital video data. The primary goal of video coding is to reduce the amount of data required to represent a digital video stream, making it easier to store and transmit over bandwidth-limited networks while maintaining an acceptable level of quality.

The process of converting a continuous video signal into a digital format involves several steps. To understand them, consider the mathematical representation of an analogical video signal luminance $s(x, y, t)$, where x and y are the spatial coordinates in a video frame, and t represents time. To convert this continuous function into a digital signal, we must do the following:

1. **Sampling:** $s(x, y, t)$ is sampled at discrete intervals in both space (x, y) and time (t) . This is often done using a 3D lattice (*reticolo 3D*), which is a grid that extends in three dimensions. The spatial dimensions (x, y) are on the plane of the video frame, while the third dimension (t) extends along the time axis. By sampling the signal at points on this lattice, a sequence of discrete values is obtained, which can then be processed by digital systems.
2. **Quantizing:** After sampling, each sample value is quantized. Quantization is the process of mapping a large set of input values to a smaller set — often this involves rounding off to the nearest value within a set range. For instance, while the actual luminance might be a value with a high degree of precision, quantization would involve approximating this value to the nearest value that can be represented within the bit depth of the digital system (like 8 bits per sample).
3. **Frame of the Video Sequence:** The sampled and quantized luminance values from a single time instant t are collected together to form a single frame of the video. A frame is essentially a still image, and a sequence of these frames, each representing a different time instant, makes up the video stream. When these frames are played back at a certain frame rate (e.g., 24, 30, or 60 frames per second), it creates the illusion of motion which is the essence of a video.

9.1 MOTION ESTIMATION

Motion estimation is a critical concept in the field of video processing. It involves the estimation of the movement of objects within a scene from a sequence of video frames. The goal is to understand how the objects are moving in the scene over time, usually from one frame to the next.

9.1.1 Issues in Motion Estimation

9.1.1.1 3D vs. 2D Motion

Real-world objects move in three dimensions, but a video captures only the 2D projection of this movement onto the image plane (x, y) . Estimating the true 3D motion requires additional information, typically depth, which can be obtained through techniques like stereo vision or depth sensors.

9.1.1.2 Covered/Uncovered Areas

When an object moves, it may uncover previously hidden parts of the scene or cover parts that were visible before. These areas do not have corresponding pixels in the previous or next frame, complicating motion estimation.

9.1.1.3 True Motion vs. Apparent Motion

True motion is the actual physical displacement of objects between frames, whereas apparent motion may be caused by changes in lighting or shadows rather than object movement. Distinguishing between these is essential for accurate motion estimation.

9.1.1.4 Aperture Problem

This problem arises when motion is observed through a small window (aperture). The observer can only detect [the component of motion that is parallel to the local edge or gradient](#). This is problematic when there's not enough texture in the image to determine the perpendicular component of the motion.

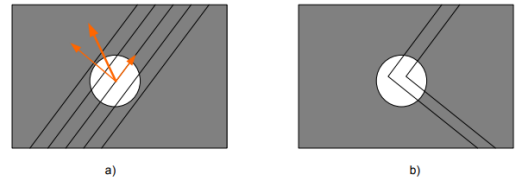


Figure 53

The aperture problem is further addressed by modifying the size of the observation window. A larger window can reduce the aperture problem by including more textured areas, which helps in estimating the motion vector more accurately. However, it also increases the risk of including multiple objects with different motions within the same window. A multi-resolution approach is typically used to resolve this by analyzing the motion at different scales.

9.1.1.5 Noise

Various types of noise, including thermal noise from the camera sensor, quantization noise from digital conversion, and aliasing from sampling, can corrupt the motion information and must be accounted for. Filtering methods, such as low-pass filters, are often used to reduce the impact of noise on motion estimation.

9.1.2 Block Matching

Block matching is a motion estimation technique widely used in video coding due to its simplicity and effectiveness, which makes it suitable for hardware implementation. It forms the basis for most video compression algorithms found in standard video codecs. This algorithm is made of the following steps:

1. **Block Division:** a video frame at time t is divided into a grid of non-overlapping $N \times N$ blocks. Each block typically consists of a square group of pixels, for example, an 8×8 or 16×16 pixel block. Selecting the appropriate block size is a critical decision that impacts the efficiency and effectiveness of the motion estimation:
 - Using smaller blocks can lead to more precise motion models. Because smaller blocks can more closely follow the movement of individual objects or parts of objects, the motion estimation can be more accurate, especially in scenes with complex motion or non-uniform motion within objects. However, smaller blocks come with a trade-off. They increase the number of motion vectors that need to be computed and stored since more blocks mean more vectors. Consequently, this requires more data to be encoded, leading to an increase in the bit rate. In other words, while the motion estimation might be more accurate, the video might not compress as efficiently because you need to store more motion vectors.
 - Conversely, larger blocks reduce the number of motion vectors and can lead to better compression rates. However, they may not capture motion as accurately, especially if the block contains multiple objects or parts of objects with different motions.

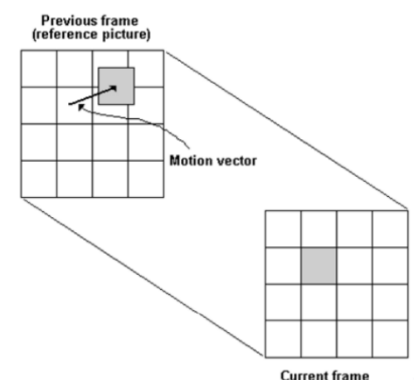


Figure 54

The choice of block size is, therefore, a balance between the desired precision of motion estimation and the efficiency of video coding.

2. **Motion Estimation:** for each block in the current frame, the algorithm searches for a block with the most similar content in the reference frame (s) — usually the immediate previous or the next frame. The performance of the motion estimation algorithm is directly tied to the chosen block size.
3. **Assumption of Uniform Motion:** block matching operates under the assumption that all pixels within a block undergo the same motion. This simplifies the calculation but can sometimes lead to inaccuracies, especially in areas with complex or non-uniform motion.

4. **Searching for the Best Match:** the algorithm tests different positions within a predefined search range S in the reference frame to find the block that best matches the current block.
5. **Motion Vector Calculation:** the displacement of the best matching block from the original position in the current frame is represented as a motion vector (v_x^*, v_y^*) . This vector indicates how far and in what direction the block has moved from one frame to the next.
6. **Error Minimization:** the best motion vector is determined by minimizing a cost function, typically the Mean Square Error (MSE). The MSE measures the difference between the pixel values of the original block and the candidate block from the reference frame. The motion vector that results in the lowest MSE is chosen as the best match. The mathematical formula (using the MSE) is:

$$(v_x^*, v_y^*) = \arg \min_{(v_x, v_y) \in S} \left[\frac{1}{N^2} * \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |I(x+i, y+j, t) - I(x+i+dx, y+j+dy, t-1)|^2 \right]$$

In which $\left\{ \begin{array}{l} \arg \min_{(v_x, v_y) \in S} \text{ means "the vector } (v_x, v_y) \text{ belonging to the search range } S \text{ that minimizes MSE.} \\ I(x+i, y+j, t) \text{ is the current intensity value of the pixel at coordinates } (x+i, y+j). \\ I(x+i, y+j, t) \text{ is the intensity value at } t = t-1 \text{ of the pixel at coordinates } (x+i+dx, y+j+dy). \end{array} \right.$

9.1.3 Motion compensation

Once motion vectors have been estimated, the process of motion compensation begins. Motion Compensation uses the vectors obtained from Motion Estimation to create a prediction of the current frame based on the previous frames.

9.1.4 Displaced Frame Difference (DFD)

This predicted frame from motion compensation is then compared to the actual current frame, and the difference between them is computed. This difference is known as the prediction residual or **Displaced Frame Difference (DFD)**. The DFD represents the error or changes that cannot be predicted by motion estimation alone: it is the information that must be encoded and transmitted to allow the decoder to reconstruct the current frame using the prediction.

The DFD is crucial because it typically contains much less data than the original frame or the predicted frame, allowing for efficient compression. In the context of video compression, only the motion vectors and the DFD need to be stored or transmitted, rather than the entire frame. This significantly reduces the amount of data required and is a key factor in the effectiveness of modern video compression standards like H.264 and HEVC.

The Displaced Frame Difference for a single pixel can be calculated as:

$$DFD(x, y, t) = I(x, y, t) - I(x + v_x(x, y), y + v_y(x, y), t - 1)$$

where $\left\{ \begin{array}{l} I(x, y, t) \text{ is the luminance of the pixel in the current frame (current time } t). \\ I(x + v_x(x, y), y + v_y(x, y), t - 1) \text{ is the luminance of the correspondent pixel of the past video frame.} \\ v_x(x, y) \text{ and } v_y(x, y) \text{ represents the space shift along the two axys indicated by the motion vector.} \end{array} \right.$

9.1.5 Mean Square Error (MSE) and Peak Signal Noise Ratio (PSNR)

We can easily calculate the MSE starting from DFD as:

$$MSE = \frac{1}{N_1 * N_2} * \sum_{x=0}^{N_1-1} \sum_{y=0}^{N_2-1} DFD(x, y, t)^2$$

and also, the PSNR as:

$$PSNR = 10 * \log_{10} \frac{2^{B-1}}{MSE}$$

9.1.6 Matching Criteria

When performing block matching in motion estimation for video processing, reducing the computational load is a significant concern. To achieve this without compromising too much on accuracy, alternative distance measures to the Mean Square Error (MSE) can be used:

9.1.6.1 Mean Absolute Difference (MAD)

Mean Absolute Difference MAD is a simpler computation compared to MSE and is defined as the average of the absolute differences between the pixel intensities of the current frame and the reference frame. The formula for MAD is:

$$MAD(v_x, v_y) = \frac{1}{N^2} * \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |I(x+i, y+j, t) - I(x+i+dx, y+j+dy, t-1)|$$

where $\begin{cases} I(x+i, y+j, t) \text{ is the luminance of the pixel in the current frame (current time } t). \\ I(x+i+dx, y+j+dy, t-1) \text{ is the luminance of the correspondent pixel of the past video frame.} \end{cases}$

9.1.6.2 Sum of Absolute Differences (SAD)

Sum of Absolute Differences SAD is another measure that, like MAD, uses the absolute difference between pixel intensities. However, instead of averaging these differences, SAD sums them up. The formula for SAD is:

$$SAD(v_x, v_y) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |I(x+i, y+j, t) - I(x+i+dx, y+j+dy, t-1)|$$

where $\begin{cases} I(x+i, y+j, t) \text{ is the luminance of the pixel in the current frame (current time } t). \\ I(x+i+dx, y+j+dy, t-1) \text{ is the luminance of the correspondent pixel of the past video frame.} \end{cases}$

9.2 SEARCH CRITERIA

In the context of motion estimation for video compression, there are two primary search strategies for determining the best match for a block: full search and fast search. Each method has its advantages and constraints.

9.2.1 Full Search

Full search, also known as exhaustive search, is a methodical approach that checks every possible position within a predefined search window to find the best matching block.

The search window is defined in a range $\pm W$, where W is the maximum distance in pixels that the search will move from the original block position in all directions (horizontally and vertically). In full search, the total number of candidate motion vectors is $(2W + 1)^2$. This is because the search checks every possible position within the window, one pixel at a time, across both dimensions. The optimal value for W is determined by the specific requirements of the video content and application, such as:

- Amount of motion: Videos with fast-moving objects may require a larger W to capture larger movements.
- Spatial resolution: Higher resolution videos may benefit from a larger W to maintain quality.
- Computational complexity: A larger W increases the number of computations, which can be intensive.

The search can be conducted in various patterns, such as raster order (*left-to-right, top-to-bottom*) or spiral order (*starting from the center and moving outward in a spiral*).

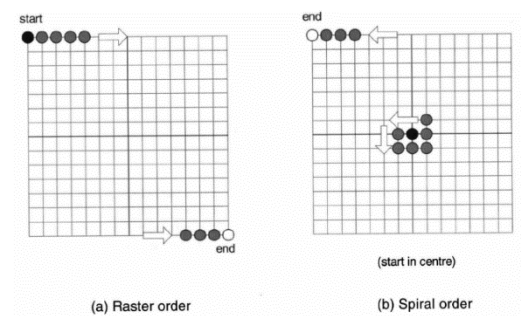


Figure 55

9.2.2 Fast Search

Fast search algorithms aim to reduce the computational complexity of block matching by avoiding a full search. They seek (*attempt*) to find a good enough match more quickly by following 'smart' patterns or heuristics. Examples of Fast Search Algorithms are:

1. **Three Step Search (TSS):** Iteratively refines the search area in three steps, each time narrowing down the search to a smaller region around the best match found in the previous step.
2. **Logarithmic Search:** Begins with a large step size and progressively reduces the step size in a logarithmic manner until the best match is found.
3. **Cross Search:** Evaluates a cross-shaped pattern within the search area to quickly locate a potential match with fewer computations.

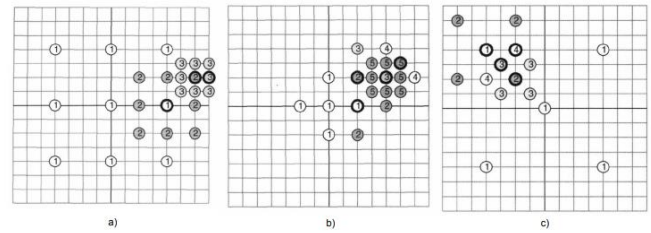


Figure: Fast search algorithms. a) Three Step Search. b) Logarithmic Search. c) Cross Search

Figure 56

These fast search methods are particularly useful in real-time video processing where computational resources are limited or when encoding speed is crucial. They trade off some accuracy for a significant reduction in computation time compared to full search.

9.3 SUB-PIXEL MOTION ESTIMATION

Sub-pixel motion estimation is a technique used to enhance the precision of motion estimation in video processing beyond the limitations of integer-pixel accuracy. Typically, motion estimation identifies how much a block of pixels has moved from one frame to another, with the motion vector representing this movement in terms of (*whole*) pixel displacements. However, by utilizing sub-pixel accuracy, it's possible to achieve a more refined estimation of motion, pinpointing (*localizzando*) the displacement to fractions of a pixel, such as $\frac{1}{2}$, $\frac{1}{4}$, or $\frac{1}{8}$ of a pixel.

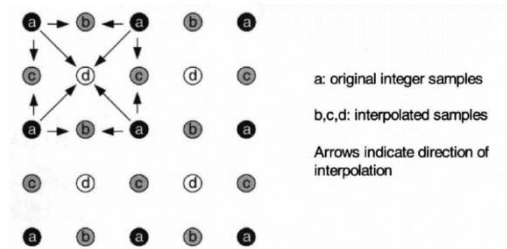


Figure 57

To achieve this higher accuracy, the reference frame from which the motion is being estimated must be interpolated. **Interpolation** is a process of estimating intermediate values between pixel intensities to create 'virtual' pixels at sub-pixel positions. This creates a denser motion field that can be searched for matches with finer granularity. The factor of interpolation, denoted by M , determines the level of sub-pixel accuracy, with $\frac{1}{M}$ representing the smallest detectable motion.

Higher sub-pixel accuracy generally leads to better prediction performance, often reflected in higher Peak Signal-to-Noise Ratio (PSNR) values.

However, with sub-pixel accuracy, the number of potential candidate predictors grows exponentially. For example, if the search window for integer-pixel motion estimation is defined by $\pm W$, the number of candidate vectors tested is $(2W + 1)^2$. When $M = 2$ sub-pixel estimation is introduced, the search space can effectively quadruple to $(4W + 1)^2$ for each sub-pixel position.

A practical approach often employed is to **first perform an integer pixel search** to find a rough motion vector and then **refine this estimate at the sub-pixel level**. This two-step process provides a good compromise between computational efficiency and the precision of the motion estimation.

9.4 PROGRESSIVE SCANNING AND INTERLACED SCANNING (*TIME SAMPLING*)

In digital video technology, there are two primary methods for sampling frames in time domain: **progressive scanning** and **interlaced scanning**. These methods impact how images are captured and displayed, influencing the quality and smoothness of the video playback.

9.4.1 Progressive Scan

In progressive scan each frame is captured and displayed in its entirety, with all lines of resolution presented in sequence. In other words, each frame is a full snapshot of a single moment in time. This method is similar to how a computer monitor displays images and is ideal for fast-moving scenes because it reduces motion blur and maintains image sharpness.

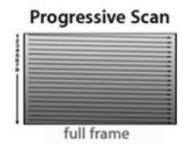


Figure 58

9.4.2 Interlaced Scan

Interlaced scanning captures and displays each frame in two fields. The first field (odd sampling times $t = 2K + 1$) consists of all the odd-numbered lines of resolution, and the second field (even sampling times $t = 2K$) consists of all the even-numbered lines. These two fields are captured at slightly different times and then interwoven to create a single frame. This technique was traditionally used in broadcast television to reduce bandwidth usage and flicker perception on CRT (*Cathode Ray Tube*) displays.

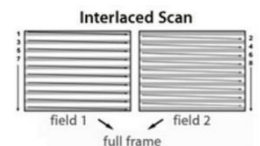


Figure 59

9.5 COLOUR VIDEO SEQUENCES

Colour video sequences require three colour components, and uncoded video material is typically sampled in the *YUV 4:2:0* format:

- **Y Component:** represents luma (brightness) information.
- **U and V Components:** represent chroma (colour) information and are subsampled at half the resolution of the luma component, both horizontally and vertically. This reflects the human visual system's reduced sensitivity to colour details compared to brightness.

The *YUV 4:2:0* format is efficient because it reduces the amount of data by focusing on the more perceptually significant luma information while using less data for the color information, which has a lower impact on perceived image quality.

Typically, $8 \frac{\text{bit}}{\text{pixel}}$ are used for each color component.

9.6 HYBRID DPCM/DCT VIDEO CODING

Hybrid DPCM/DCT video coding is a foundational technique used in many video compression standards, including MPEG and H.264. It combines [Differential Pulse Code Modulation \(DPCM\)](#) for temporal redundancy reduction and [Discrete Cosine Transform \(DCT\)](#) for spatial redundancy reduction. This hybrid approach takes advantage of temporal correlations between successive frames through motion estimation and compensation, while also exploiting spatial correlations within a frame using the DCT. The result is a highly efficient video compression method that significantly reduces the amount of data needed to represent high-quality video.

Below is a detailed explanation of the encoding and decoding processes in such systems.

9.6.1 Encoding Process

1. Prediction Error Calculation (e_t):

the encoder computes the prediction error, which is the difference between the current block and the motion-compensated predictor: $e_t = f_t - \hat{f}_t^{MC}$.

2. DCT Transformation (d_t):

the prediction error e_t is then transformed using the DCT obtaining d_t . The DCT helps to compact the block's energy into fewer coefficients, which are easier to compress.

3. Quantization ($\tilde{d}_t$):

the DCT coefficients are quantized, effectively reducing the precision of the coefficients and thereby removing some of the less perceptually important information, which contributes to compression. The quantized DCT coefficients $\tilde{d}_t$ are:

- **Entropy-coded** using techniques like Huffman coding or arithmetic coding. This step further compresses the data by encoding the values with variable length codes based on their frequency of occurrence.
- Given in input to the inverse quantizer so that the encoder can emulate the reconstruction process of the encoder (see next steps) to ensure the decoder can reconstruct the frame ([closed loop DPCM](#)).

4. Inverse Quantization ($\hat{d}_t$):

the range of the values is restored through de-quantization obtaining $\hat{d}_t$.

5. Inverse DCT ($\tilde{e}_t$):

the DCT transformation is reversed to retrieve the prediction residuals from the DCT coefficients, obtaining the reconstructed prediction residual $\tilde{e}_t$.

6. Prediction block ($\hat{f}_t^{MC}$):

now let's look inside the sub-diagram that predicts the next frame:

- The reconstructed prediction residual $\tilde{e}_t$ is summed up to the predicted value $\hat{f}_t^{MC}$ to get the reconstructed actual value $\tilde{e}_t + \hat{f}_t^{MC} = \tilde{f}_t$.
- The **Frame Buffer** 'save' the frame $\tilde{f}_t$ giving it in output at the time $t + 1$. In this way, the MC and ME modules will receive at their input the frame $\tilde{f}_{t-1}$.
- The **Motion Estimation** module takes in input the actual current frame f_t and the reconstructed past one $\tilde{f}_{t-1}$ and, comparing the two frames (*block matching, search criteria...*), performs the motion estimation to get the Motion Vectors MV_t . This result is:
 1. **Entropy Coded** and sent or stored.
 2. Given to the Motion Compensation module.
- The **Motion Compensation** module will 'create' a new frame starting from the reconstructed past one ($\tilde{f}_{t-1}$) and applying to it the changes indicated in the MV_t vectors. The result $\hat{f}_t^{MC}$ is:
 1. Re-summed at the beginning of the 'prediction cycle'.
 2. Used as predicted frame at the very beginning of the Hybrid DPCM/DCT encoding process.

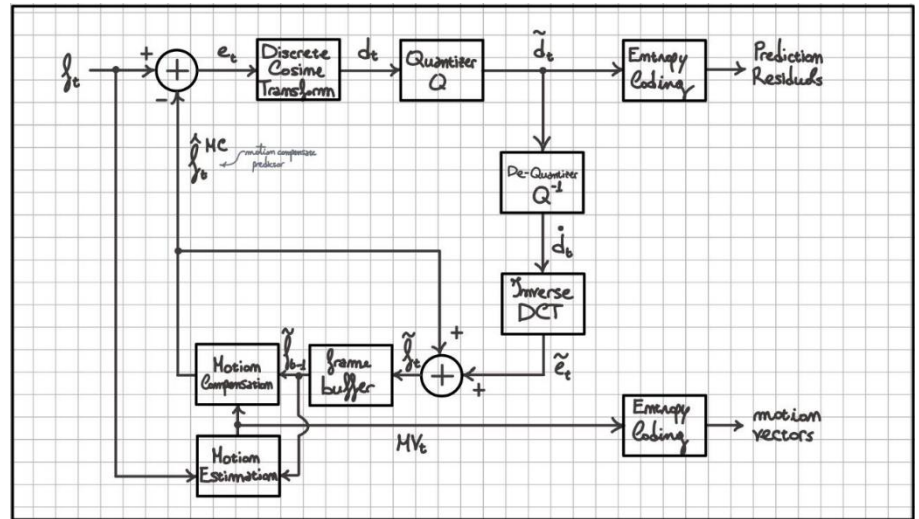


Figure 60

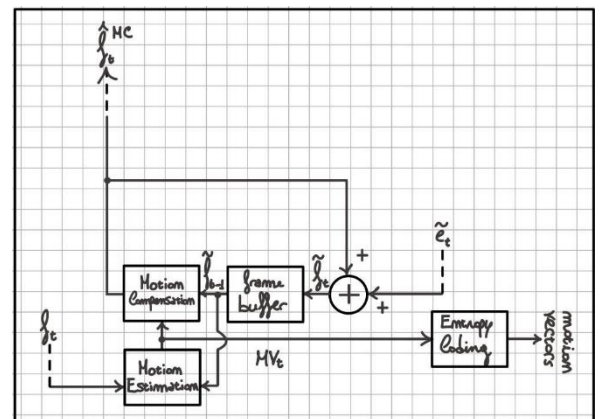


Figure 61

9.6.2 Decoding Process

1. **Entropy Decoding:** the decoder take in input the encoded prediction residuals and motion vectors and apply an Inverse Entropy Coding in order to retrieve the quantized DCT coefficients $\tilde{d}_t$ and the motion vectors MV_t .

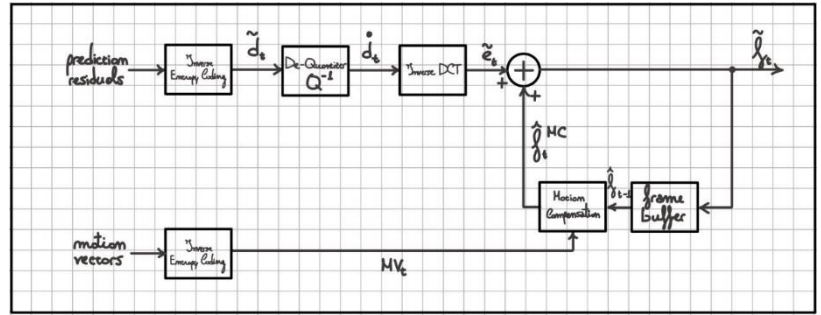


Figure 62

2. **Reconstruct the prediction error ($\tilde{e}_t$):** the quantized DCT coefficients $\tilde{d}_t$ passes through an **Inverse Quantization Filter** obtaining the reconstructed DCT coefficients $\hat{d}_t$. These coefficients are then anti-transformed by an Inverse DCT filtered, that give in output the reconstructed prediction error $\tilde{e}_t$.
3. **Reconstruct the frame $\tilde{f}_t$:** the prediction error is summed up to the predicted frame $\hat{f}_t^{MC}$ (see next steps) in order to obtain the reconstructed frame $\tilde{e}_t + \hat{f}_t^{MC} = \tilde{f}_t$. The reconstructed frame is then:
 - Given as output of the decoder.
 - Given in input to the frame buffer, in order to perform the next prediction.
4. **Predict the next frame ($\hat{f}_t^{MC}$):**
 - the **Frame Buffer** takes in input the reconstructed frame $\tilde{f}_t$ and 'store it' to be used in the next iteration, giving in output $\tilde{f}_{t-1}$.
 - The **Motion Compensation module** takes in input the past reconstructed frame $\tilde{f}_{t-1}$ and the motion vectors MV_t and 'generate' a new predicted frame starting from $\tilde{f}_{t-1}$ and applying the 'changes' indicated from MV_t . The output of this module is the next predicted frame $\hat{f}_t^{MC}$, which will be summed up to the reconstructed prediction error $\tilde{e}_t$ in the next algorithm iteration to get the next reconstructed frame.

9.6.3 Key Concepts in Hybrid DPCM/DCT Video Coding

In video coding, particularly in formats using hybrid DPCM/DCT (Differential Pulse Code Modulation/Discrete Cosine Transform) methods, there are several key concepts and components involved in encoding and decoding video sequences.

9.6.3.1 Frame Types

INTRA-frame Coding: This is used for the first frame in a sequence, coding it as a standalone image without reference to other frames. This type of coding is also inserted periodically to enable random access and provide robustness against errors. INTRA-frames are coded using only spatial information within the frame itself.

INTER-frame Coding: Subsequent frames are typically coded using INTER-frame coding, which involves coding the differences between the current frame and the previously coded frames. These differences are often represented as motion-compensated prediction errors.

9.6.3.2 Synchronization

The encoder and decoder must be synchronized, meaning that they both maintain an identical copy of previously decoded frames. This synchronization allows the encoder to work in a closed-loop fashion, using feedback from the functional modules of the decoder.

Synchronization is preserved as long as the communication channel is noiseless. If errors occur during transmission, the decoder may not reconstruct the same frame as the encoder, leading to drift over time.

9.6.3.3 Coding Efficiency

The efficiency of coding a frame, represented by e_t for the displaced frame difference, is better when the energy of e_t is small. Lower energy in the prediction error results in higher coding efficiency because less information needs to be transmitted.

9.7 VIDEO CODING IN MATLAB

9.7.1 Motion Estimation on a single image block

```
% Motion Estimation on a single block of an img
% Select the 8 x 8 block from the img starting at (x, y) = (150, 35) of the
second frame.
N = 8;
x_pos = 150;
y_pos = 35;
block = img(y_pos:y_pos+N-1 , x_pos:x_pos+N-1 , 2);
% ME using window W = 16 pixels using SAD as distance measure
W = 16;
rowRng = -W:W;
columnRng = -W:W;
SAD = zeros(length(rowRng), length(columnRng));
for i = 1:length(rowRng)
    for j = 1:length(columnRng)
        comparison_block = img(y_pos+rowRng(i):y_pos+rowRng(i)+N-1, ...
                                ... x_pos+columnRng(j):x_pos+columnRng(j)+N-1,1);
        SAD(i,j) = sum(abs(comparison_block(:)-block(:)));
    end
end
[~, min_idx] = min(SAD(:));
[row_idx,column_idx] = ind2sub([size(SAD)],min_idx);
% Store the motion vector.
Vx = columnRng(column_idx);
Vy = rowRng(row_idx);
% Display the cost function.
imagesc(columnRng,rowRng,SAD);
% Visualize the motion vector.
quiver (x_pos+N/2,y_pos+N/2,Vx,Vy);
```

9.7.2 Motion Estimation and Motion Compensation on a whole image

```
%% Perform MC using 8 x 8 blocks performing full-search, using a search window W = 16.
N = 8;
W = 16;
Nrows = length(img(:,1,2))/N;
Ncolms = length(img(1,:,2))/N;
motionVectors = zeros(Nrows, Ncolms,2);
motCompensatedImg= zeros(size(img(:,:,2)));
% Perform MC
for i = 1: Nrows
    for j = 1 : Ncolms
        r_Idx = (i-1)*N+1;
        c_Idx = (j-1)*N+1;
        currBlock = img(r_Idx:r_Idx+N-1,c_Idx:c_Idx+N-1,2);
        r_Range = -W:W;
        c_Range = -W:W;
        SAD=nan(length(r_Range),length(c_Range)); % better initialize as nan
        % Perform full search
        for l = 1 : length(r_Range)
            for m = 1 : length(c_Range)
                row_idx = r_Range(l);
                col_idx = c_Range(m);
                if(r_Idx+row_idx>0 && r_Idx+row_idx+N-1<length(img(:,1,2))&& ...
                    ... c_Idx+col_idx>0 && c_Idx+col_idx+N-1<length(img(1,:,2)))
                    nextBlock = img(r_Idx+row_idx:r_Idx+row_idx+N-1, ...
                        ... c_Idx+col_idx:c_Idx+col_idx+N-1,1);
                    SAD(l,m) = sum(abs(currBlock(:)-nextBlock(:)));
                end
            end
        end
        [~,min_idx]=min(SAD(:));
        [row_idx,col_idx] = ind2sub(size(SAD),min_idx);
        % Save Motion Vector
        vectors(r_Idx,c_Idx,1)=r_Range(row_idx);
        vectors(r_Idx,c_Idx,2)=c_Range(col_idx);
        % Put the motion compensated block into the predicted image.
        motCompensatedImg(r_Idx+vectors(r_Idx,c_Idx,1):r_Idx+vectors(r_Idx,c_Idx,1)+N-1, ...
            ... c_Idx+vectors(r_Idx,c_Idx,2):c_Idx+vectors(r_Idx,c_Idx,2)+N-1) = currBlock;
    end
end
%% Compute DFD and calculate its mean and variance.
DFD = img(:,:,2)-motCompensatedImg;
variance = var(DFD(:));
average = mean(DFD(:));
```

10 OTHER MATLAB UTILITIES

10.1 LINEAR REGRESSION

Useful when the exercise asks something like “quantize y with DPCM using as predictor $y_{\text{hat}}(n) = a * x(n) + b$ where ‘ a ’ and ‘ b ’ are obtained by linear regression (least squares on $y = ax + b$)”.

```
% Linear regression between two sources x and y to
% get the a and b coefficients
alpha = [x, ones(size(x, 1), 1)] \ y;
a = alpha(1);
b = alpha(2);

% In this specific case the signal is:
y_hat = a * x + b;
```

10.2 DEFINE x_{axys} IN FREQUENCY DOMAIN

```
% X_AXYS DEFINITION:
x_axys = 0 : Fs/N_samples : (N_samples-1)*Fs/N_samples;
% When We represent the frequencies domain, we usually represent it from 0
to Fs. % For this reason, we have that:
%     - The step size is Fs/N_samples (total length/number_of_segments).
%     - The “final step” is (N_length.1)*Fs/N_samples
%       (total_length-one_segment)because we start from 0.
```

11 MIND MAP

