Exercise 3: Image and Audio

Course: Computer Networking 2, DVGC02

Responsible teacher: Kerstin Andersson

Exercise group: Sebastian Machat

Marek Jesensky

# Audio Coding

1. *If we encode the samples independently, how many codewords are needed? (0.25 point)*

If we encode the sample independently, we need just one code word for each sample. The final number of Code words depends on length of audio track and on sample rate. It means .

1. *If we encode the difference between each sample and the previous sample, how many codewords are needed? (0.25 point)*

If we encode differences between each sample and the previous sample there will be one code word less than for coding independent samples. It means .

1. *Under what conditions the method in b) is supposed to create shorter average code length than method a)? (0.5 point)*

If the samples of audio signal are highly correlated with each other. The differential coding technique takes advantages of this correlation in with difference between adjacent samples is encoded.

# Differential Coding

Consider the sequence

(3.4, 5.2, 3, 1, -1.5, -2, -0.7)

1. In order to encode this sequence using differential coding and using the following model calculate a suitable constant a. There is no need to encode the sequence. (0.5 point)

Hint: A good constant is the one, which minimizes the sum of the squared residual errors over the whole sequence. You can calculate this by writing down the sum of the residual errors and taking the derivative with respect to a, which needs to be zero in order to yield the minimum. Then you can solve for a.

As the sequence we got in assignment contains always only one value for every member, mode of each of the members is equal to its value. Therefore we can simplify the model to:

We’ve followed the hint in the assignment, which lead us to calculating sum of the squared residual errors of the whole sequence:

Which, after substituting the real values, led to:

Deriving this function with respect to took us to the following function, which we solved as:

Therefore we can say that the most suitable constant for the assigned model and sequence would be approximately equal to .

1. What general observation can you make in order to achieve a high compression ratio for differential coding? (0.25 point)

As for the general observations leading to achieving as low bitrate (or bandwidth) as possible, the most obvious option is to make sure all the members of the encoded sequence were the same (and in the best case all of them equal to 0 or any other previous value).

Another such high-compressible sequence would be one where value of the following member was always just value of the previous member multiplied by constant number. In such a case we could use the model from question get the compression ratio as high as possible.

As neither of those cases is very often in signal processing, it seems to us that the idea used in encoding images (to go through the whole block in such order that we always encode adjacent pixels right after each other) is very smart and should be very helpful in trying to achieve higher compression ratio.

# MPEG Audio Coding

1. *Explain the effect of temporal masking and frequency masking, and how they are used in MP3 lossy audio compression. (0.25 point)*

Temporal masking - As we can see on Figure 1 there are surrounding areas around masking sound. In these areas human ear is not able to perceive sounds with lower intensity than masking tone has. For pre-masking it is approximately 20ms and for post-masking 200ms. It is the reason why it is not necessary to encode these masked tones in lossy audio coding.

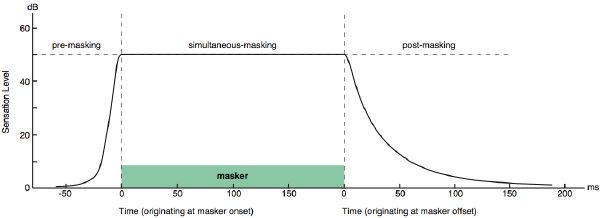


Figure 1 - Schematic drawing of temporal masking, including pre-masking, simultaneous masking, and post-masking (source: http://web.media.mit.edu/~tristan/phd/dissertation/chapter3.html )

Frequency masking – In case there are two or more tones in the same time in audio track with different intensity, the human ear is able to perceive only one with higher intensity. This is used for lossy audio coding. Tone with lower intensity does not encode and listener can’t recognize the difference.

1. *Assume an audio signal is divided into 16 frequency bands with energy in different bands as follows*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Band* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *14* | *15* | *16* |
| *Level (dB)* | *0* | *8* | *9* | *12* | *6* | *2* | *15* | *55* | *8* | *15* | *11* | *2* | *3* | *5* | *3* | *1* |

*Assume that if the level of the 8th band is 55dB, it gives a masking of 12 dB in the 7th band, 15dB in the 9th. How many bits you would need to code the 7th band and 9th band respectively? Suppose original signal is represented with 8 bits/sample/band. Remember, when uniformly quantizing a signal, the improvement obtained by adding one more bit per sample is around 6dB. (0.5 point)*

The level of 7th band is 15dB > 12dB given masking. This one we have to encode. We choose the quantization factor so that the quantization error will be less than 1 bit (6dB).

The level of 9th band is 8dB > 15dB given masking. It means we can ignore this band.

# DCT Coding Implementation

1. *Transform the block according to the forward 2D DCT equation presented in the lecture. The resulting coefficients may be rounded to full integer values by cutting-off values after the decimal point (i.e. 3.14 becomes 3 and 3.79 becomes 3). Write down the truncated coefficients. You may implement in Java or C++, use Matlab or Excel, etc. (0.5 pts)*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 477 | -127 | -24 | -11 | -2 | -11 | -10 | -9 |
| 64 | -22 | -21 | 20 | 9 | 5 | -1 | -8 |
| 54 | 24 | -43 | -38 | -6 | -7 | 0 | -5 |
| 55 | 69 | 23 | -23 | -15 | -11 | -4 | -8 |
| -25 | -6 | 13 | 0 | 4 | 1 | 16 | 15 |
| -18 | -13 | 9 | 19 | 6 | -5 | -6 | 0 |
| -9 | -18 | -4 | 3 | 14 | 3 | -5 | -9 |
| -5 | -7 | -9 | -5 | 0 | -1 | 0 | -3 |

Table 1 - Truncated coefficients

1. *Now do the quantization of the coefficients from the previous exercise according to the following quantization table. Again, round the values like you did before. You may implement in Java or C++, use Matlab or Excel, etc. (0.25 pts)*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 29 | -11 | -2 | 0 | 0 | 0 | 0 | 0 |
| 5 | -1 | -1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | -2 | -1 | 0 | 0 | 0 | 0 |
| 3 | 4 | 1 | 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 |

Table 2 - Quantized coefficients

1. *Note down the quantized DC value. Explain how it is encoded. (0.25 pts)*

Quantized DC value is 11101. This number is encoded by using DC Codes table (Table 3).

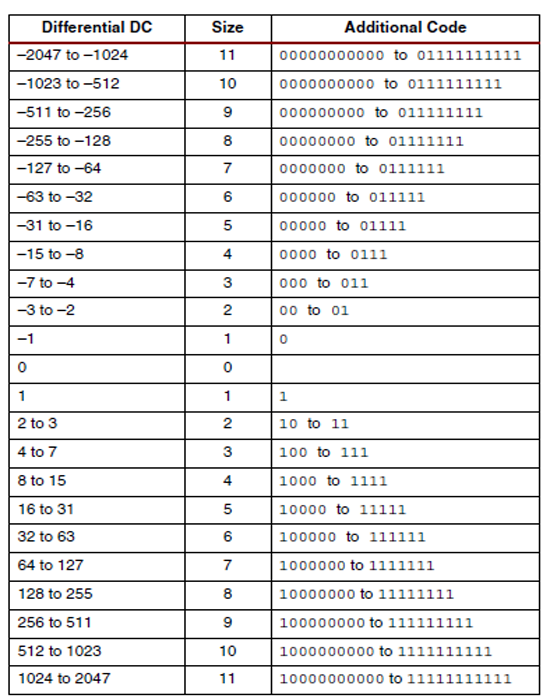


Table 3 - DC Codes (source: http://appliedelectronicsengineering.blogspot.se/2013/10/implementation-of-huffman-coding.html)

1. *For the zero based run-length encoding of the AC coefficients, use the notation <runlength, character>, where runlength denotes the number of preceeding zeros. Calculate the resulting string after zero based run-length encoding for the quantised transformed block as resulting from the previous step. Use the zigzag scanning for the AC coefficients. (0.25 pts)*

29, -11, 5, 3, -1, -2, 0, -1, 1, 3, -1, 4, -2, 1, 0, 0, 0, -1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

↓

<0,29>; <0,-11>; <0,5>; <0,3>; <0,-1>; <0,-2>; <1,-1>; <0,1>; <0,3>; <0,-1>; <0,4>; <0,-2>; <0,1>; <3,-1>; <0,1>; EOB

↓

<0,29>; <0,-11>; <0,5>; <0,3>; <0,-1>; <0,-2>; <1,-1>; <0,1>; <0,3>; <0,-1>; <0,4>; <0,-2>; <0,1>; <3,-1>; <0,1>; <0,0>

↓

(0,5,11101); (0,4,0100); (0,3,101); (0,2,11); (0,1,0); (0,2,01); (1,1,0); (0,1,1); (0,2,11); (0,1,0); (0,3,100); (0,2,01); (0,1,1); (3,1,0); (0,1,1); (0,0)

↓

11010 11101, 1011 0100, 100 101, 01 11, 00 0, 01 01, 1100 0, 00 1, 01 11, 00 0, 100 100, 01 01, 00 1, 111010 0, 00 1, 1010

1. *Decode the quantized coefficients from the last exercise and apply the inverse DCT. Note down the reconstructed values. You may implement in Java or C++, use Matlab or Excel, etc.. (0.5 pts)*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 62 | 69 | 79 | 87 | 89 | 86 | 81 | 77 |
| 36 | 44 | 58 | 74 | 86 | 90 | 88 | 84 |
| 12 | 17 | 31 | 52 | 73 | 86 | 88 | 84 |
| 18 | 16 | 21 | 36 | 57 | 72 | 76 | 75 |
| 47 | 41 | 35 | 38 | 49 | 60 | 66 | 68 |
| 60 | 57 | 54 | 51 | 53 | 58 | 64 | 69 |
| 34 | 44 | 54 | 58 | 57 | 58 | 65 | 72 |
| -2 | 18 | 44 | 57 | 57 | 57 | 64 | 72 |

Table 4 - Reconstructed values

1. *Compare the reconstructed values with the original values and interpret the results. (0.25 pts)*

There is lot of differences in reconstructed values and original values. They are mainly caused by cutting-off values between operations. There are still significant areas of similar numbers in the table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 47 | 77 | 80 | 96 | 91 | 90 | 80 | 94 |
| 37 | 66 | 78 | 95 | 90 | 95 | 90 | 84 |
| 4 | 4 | 4 | 65 | 82 | 90 | 90 | 84 |
| 31 | 4 | 4 | 17 | 59 | 71 | 82 | 78 |
| 41 | 45 | 50 | 38 | 46 | 59 | 69 | 78 |
| 50 | 77 | 52 | 62 | 52 | 59 | 69 | 78 |
| 26 | 65 | 57 | 55 | 59 | 59 | 64 | 69 |
| 0 | 20 | 55 | 57 | 60 | 55 | 59 | 74 |

Table 5 - Original values

1. *Calculate the MSE and the PSNR for the reconstructed block. (0.25 pts)*
2. *Plot the quantized coefficients, but this time multiplying the Quantization matrix with the factor of 2. Decode the quantized coefficients and apply reverse DCT. Note down reconstructed values. Calculate MSE and PSNR and compare to g). Interpret the results. (0.25 pts)*