TSBK35 Lab2

Johnny Elmér - jonel107@student.liu.se

February 2022

1 Method

The audio files were loaded into MATLAB using audioread. Then the highest and lowest frequencies were calculated using max and min respectively on the input signal (audio). Setting the number of steps that should be taken the step size of the quantization was calculated by subtracting the lowest frequency from the highest frequency and then dividing by the number of steps.

To perform the transform coding of the signal the given function for MDCT was used with the input signal and window size as inputs, the window size was chosen as 256. The output of the transform coding was then quantized by dividing it with the step size calculated previously and rounded with the help of MATLAB's *round* function. To get the index of the values of the quantized components (with the rows of the vector in sorted order) MATLAB's *unique* function was used.

The index array was then reshaped into a vector format with the help of MATLAB's *accumarray* to make it easier to create a probability matrix. Said probability matrix was calculated by simply dividing the index array by the sum of all it's values (using *sum(sum(idxarray))*). The resulting probability matrix was then given as the input for the given Huffman coding function used in lab 1 to calculate the data rate.

To be able to listen to the signal it was decoded using the given inverse MDCT function with the input being the quantized sound multiplied with the step size. To check the SNR two different approaches were used with the same result. Firstly the SNR function was implemented by simply calculating the difference between the original and compressed audio, then calculating the distortion/mean square error (D) and then calculating $10 * log10(var(y)/D)$ where y is the original sound. The second approach was to simply use MATLAB's own *snr* function.

The number of steps were then adjusted continuously until, using the audio *heyhey* as the original audio, the data rate got at close to 2.90 as possible without going over the limit. When this was reached the audio was saved using

MATLAB's *audiowrite* to be able to listen to it. For fun two more audio's were used to be able to compare, using the same number of steps. After that comparison was made two more experiments were made for fun, where instead of matching the number of steps to the *heyhey* audio, the data rate and SNR values respectively were matched as closely as possible. This was mostly done for the sake of personal interest and not necessarily for the sake of the lab.

2 Results

Here the results of the different comparisons, as well as some general information about the different audio files, are presented in four tables. Tables three and four show the results of the extra experiments but are not the really discussed.

Variable	**heyhey**	**hey04**	**nuit04**
Sample frequency	44.1kHz	44.1kHz	44.1kHz
Nr of samples	8.192.000	512.000	739.328

Table 1: General information about the different audio files.

Value calculated	***heyhey***	**hey04**	**nuit04**
Sample frequency	*44.1kHz*	44.1kHz	44.1kHz
Nr of steps	*32.35*	32.35	32.35
Data rate	*2.8999*	3.9574	2.3054
SNR	*41.6335*	42.2632	40.4438

Table 2: Results of the compression of three different audio's, heyhey being the one needed for the lab, all using the same number of steps (32.5).

3 Discussion

Since the transform coding is rather simply done (with Huffman coding and quantization being done on the entire length of the audio and not smaller snippets of it) the amount of space taken up by the extra information is negligible/insignificant and will not affect the data rate enough to really be noticeable. The *heyhey* audio ended up getting a data rate of 2.8999 and and SNR of 41.6335.

Value calculated	**heyhey**	**hey04**	**nuit04**
Sample frequency	44.1kHz	44.1kHz	44.1kHz
Nr of steps	32.35	14.99	52.66
Data rate	*2.8999*	*2.8967*	*2.8991*
SNR	41.6335	36.5009	44.1345

Table 3: Results of trying to match the data rate in heyhey.

Value calculated	**heyhey**	**hey04**	**nuit04**
Sample frequency	44.1kHz	44.1kHz	44.1kHz
Nr of steps	32.35	29.85	38.45
Data rate	2.8999	3.8314	2.4991
SNR	*41.6335*	*41.6310*	*41.6331*

Table 4: Results of trying to match the SNR in heyhey.

Because of how MDCT works it cannot be used on songs that are not able to be separated into equal chunks, meaning it cant have an odd number of samples. This is why the audio's chosen to test alongside *heyhey* were picked so that they contained an evenly separable amount of samples.

When the sound is compressed with a 2.8999 data rate the subjective difference is barely existent when comparing with the original sound. Even though it is known that the sound is indeed compressed it is in no way clear when simply listening to it. If the number of steps is instead changed to a lower value, consequently lowering the SNR the sound fairly quickly starts to have a bit off an odd tinny/distorted quality to it, especially when focusing on the singers words. Of course, when instead changing it to a higher value, and in result getting a higher SNR, the sound doesn't sound distorted at all, but the sound doesn't subjectively sound better than when the number of steps were 32.35.

To see my code, visit the following link: `https://github.com/JohnnyElmer/TSBK35/tree/main/Lab2`