Audio search - Song recognition

Guide

BME

Table of contents

1 Introduction

Nowadays, people create and use tons of multimedia contents worldwide. There are several practical problems, where comparison/detection of audio signals are needed. For example:

- Identification of a song from its partial recording through the microphone of the user's phone. (E.g.: Shazam or SoundHound)
- Identification of songs from the radio, TV or internet stream to support copyright protection.
- Identification of signals and ads, to verify their number of occurrences defined in the contract or for other statistics.
- Searching for duplicates in audio databases to reduce their size, and to ensure that all the algorithms are working properly.
- Simple speech recognition system trained by the user

Each problem requires pattern matching solutions with different kinds of distortion and noise tolerance. In the first example, where we record the music with our own device, we should expect additive noises from the environment and distortions caused by the microphone, loudspeaker characteristics and reverberation. Where the signal is processed only electronically, the distortion can originate from the transmission channel or from the coding, and signal frames can be lost, as well. If the users train a speech recognition system with their own words they are not necessarily speaking with the same speed, so the different rhythm of the speech makes a kind of distortion not easy to compensate.

Another important aspect is speed. Even if we could implement a precise algorithm that can recognize songs from a large database, but it is much slower than real-time our solution would be practically useless. In order to improve the speed we can make a hash – or audio fingerprint – of the signals that do not preserve all the original information of the songs but are distinctive and are much smaller than the original files.

In the next exercises we are going to test some audio feature extraction and pattern matching methods which can be used for the first 4 problems. These methods tolerate most of the noises and the distortions, but not the time variances.

One of the feature extraction methods is called MFCC, based on Discrete Fourier Transformation, because we need to work in the frequency domain. We are going to examine how the DFT can be used for audio identification.

Another method we are going to use is the audio fingerprint creation and matching. This starts with a DFT as well, but it works in a completely different way afterwards. It looks for the peaks in the 2D spectrogram and then calculates the so-called landmarks. The fingerprinting method of the original Shazam algorithm is based on these landmarks.

In the 2. and 3. Sections we summarize the MFCC and fingerprinting methods.

2. MFCC feature extraction

The aim of audio feature extraction is to process the input signal so that it is applicable for comparisons (pattern matching). Ideally, all the noises and distortions should be removed but practically it is only possible to reduce their effects. Typically, some properties of human perception are mimicked but general “mathematical” algorithms (such as DFT, DCT, LDA or PCA) are also applied.

One of the most often used feature extraction technique is Mel-Frequency Cepstral Coefficients (MFCC). The block diagram of MFCC is shown in Figure 2.1.

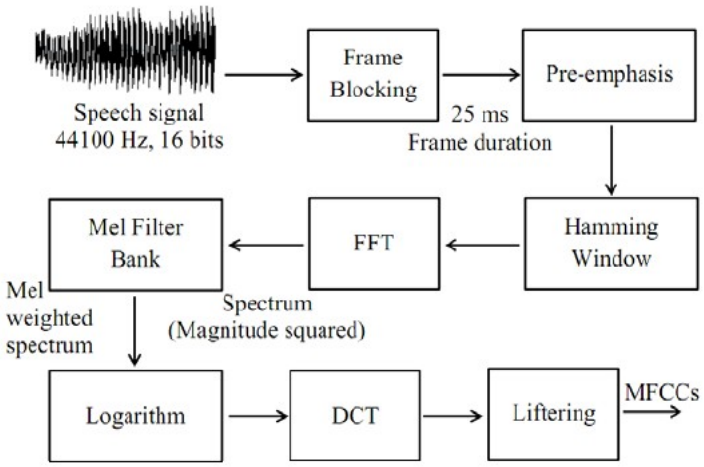

Figure 2.1. Block diagram of MFCC feature extraction

The samples of the audio data (speech or other sounds like music) are processed in the proper order. The feature vectors appear on the output of the whole chain. The feature vectors are usually 13-dimensional, and they follow each other in every 10 ms, but these numbers can vary.

Steps of MFCC feature extraction:

2.1 Framing

The input signal’s sampling rate is usually 16 kHz, 16 bit, but if it is coming from the telephone network then 8 kHz is the default value. We cut the signal into 25 ms long blocks. The signal is changing continuously, and we want to monitor these changes, so we process the signal block by block. If the size of the block is too big, then we cannot track fast changes, but if the size of the block is too small, then the frequency resolution will be low and low frequency components can be lost.

The frames follow each other in every 10 ms, so they are overlapping, which means the start of the next frame is coming earlier than the end of the actual frame. We need this because of the following steps.

2.2 Pre-emphasis

The pre-emphasis represses the low frequency components in the signal and highlight the high frequency ones. We use a first order FIR filter for pre-emphasis, the transfer function is:

2.3 Windowing

Before the Discrete Fourier Transform (DFT) we need to use windowing or else the spectrum will be glossy. The Hamming window sharpens (filters) the spectrum. Multiplying with the window function in time domain means convolution in the frequency domain, and the finite convolution means filtering with the same FIR, so the windowing filters the signal's Fourier transformed form.

Function of the Hamming window:

$$h[n] = 0.54 - 0.46 \cos \frac{2\pi n}{N}$$

where n=0…N-1, and N is the size of the window.

2.4 Discrete Fourier Transform

We can convert the (discrete) time domain signals into (discrete) frequency domain spectrums using the Discrete Fourier Transform. This is necessary because the features of the speech or music can be represented only in the spectral domain. In addition, the distortions and noises in the input signal can be reduced much better in the frequency domain – especially if followed by other transformation.

We use the fast version of DFT called as FFT because it is significantly faster than DFT and can be always applied after zero padding. We keep only the square of the absolute value of the complex spectrum (power spectrum). The phase information is irrelevant in our problems.

2.5 Mel Filter Bank

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

The formula of Mel scale:

$$f_{mel} = 2595 * \log_{10}\left(1 + \frac{f_{lin}}{700\,Hz}\right)$$

There are bandfilters distributed along the frequency according to the Mel scale. To calculate the MFCC, we have to transform the energy spectrum using these filters. A bandfilter is like a triangle window, applying the filter means we multiply the performance spectrum and the window and then we summarize it in the end.

2.6 Logarithm

The last 2 steps of signal processing are for calculating the cepstrum. The traditional cepstrum is calculated with inverse DFT from logarithmic spectrum with linear scale. Cepstrum is an artificial word to show it is formatted from the spectrum. The mel-cepstrum is calculated from the output of the above written bandfilters, with DFT or Discrete Cosine Transform (DCT). DCT is usually used in image processing, and it gives only real values. (The input signal is from the logarithmic spectrum domain instead of the time domain. The phase of the sinus-components of the signal's time function is irrelevant, but the phase of the sinus-components of the logarithmic spectrum contains important information about the tune of the music.)

Calculating the DCT coefficients:

$$W(z) = 1 - 0.95\,z^{-1}$$

where M is the number of bandfilters. We only need the first 13 coefficients. This is how we create the vector for the comparison of the signals. (The DCT also decorrelates the input vector, so we can discard the coefficients of the higher dimensions and it still contains all of the important information of the original vector.)

2.7. Comparing signals based on the MFCC

We can compare vectors by calculating Euclidean distance between them:

$$d(x, y) = \left(\sum_{i=1}^{n} \left(|x_i - y_i|\right)^2\right)^{1/2}$$

where x and y are the two vectors and n is the number of dimensions. We want to compare two signals, so we create two sequences of vectors with MFCC feature extraction. Moving to the next index means the same time delay in both cases, so the same indexes are corresponding to each other. We put the sequences next to each other, and then we find out the difference between them by calculating the Euclidean distance for each pair of vectors and taking the arithmetic mean.

We are sliding the sequences one by one and we calculate the differences in every case. So, we have a number for every frame which represents the difference between the input signal and the reference signal. We can have a match at the minimum of these numbers. Of course, there is going to be a minimum for every reference, so we define a threshold to decide if it is a real match or not. The result of an example of this method is shown on Figure 2.2. where we can see a minimum at the 700. frame. This means the reference signal contains the input from 7 s.

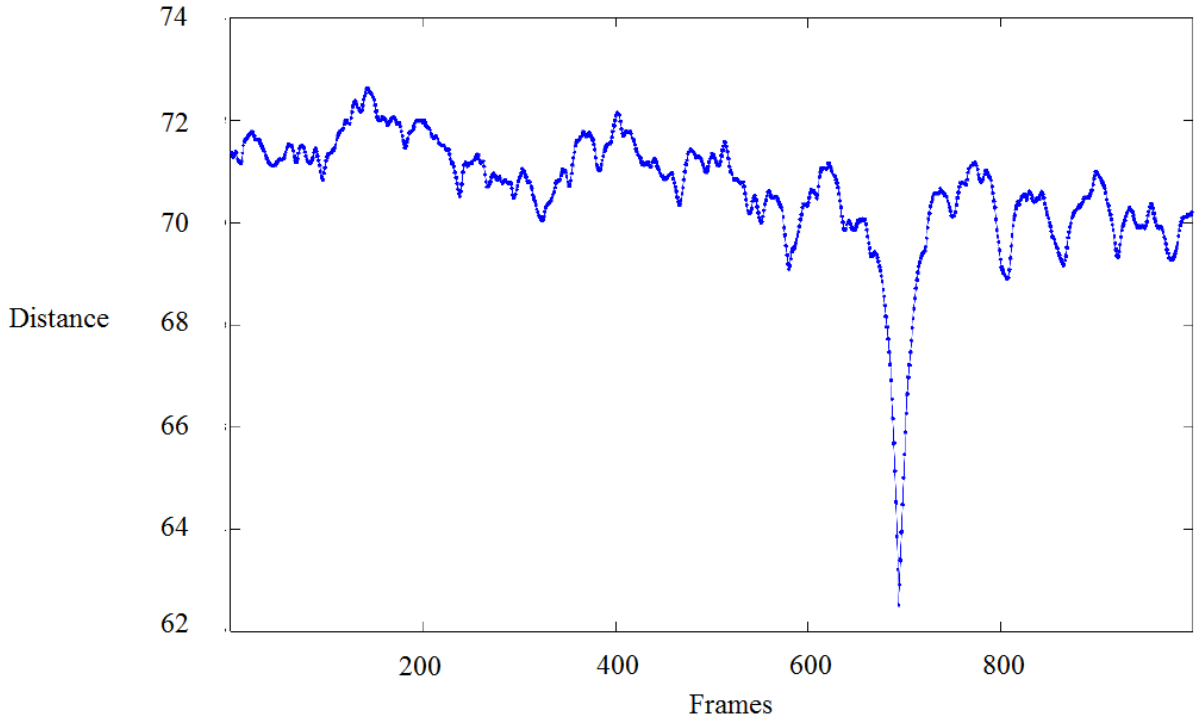

Figure 2.2. Result of a MFCC match

3 Fingerprinting method for signal identification

3.1 Audio fingerprinting

There are several versions of fingerprinting, but basically it is a unique ID for the audio signal. The usual requirements of the audio fingerprint:

- Accuracy (identification without too many false negative or false positive errors)
- Robustness (can be used with different devices and with different methods)
- Good time resolution (from a few seconds to a whole song)
- Security (defense against intentional attacks)
- Flexibility (can be used for different type of signals – speech, rock, pop, classical music – and in different environments)
- Scalability (can be used real time or with a huge number of reference signals)
- Speed (the amount of necessary resources for all the calculations)

Audio fingerprinting can be used in the next fields:

- Creating meta-databases (to easily managing audio databases)
- Identifying unknown recordings (or looking for duplicates in databases)
- Verifying integrity (checking if the reference signal was modified)
- Supporting watermarks

Audio fingerprint is very concise but unique characterization of the signal, which can be used for identification. The feature extraction techniques can be used for identification as well, but they contain way more information than necessary and they were not originally created for this usage. The main reason was to contain as much important information for the human ear as possible without noise. More difficult problems can be solved with the feature extraction techniques like speech recognition, speaker recognition, emotion recognition from the speech, classification of songs based on the genre or instruments, or classification of other non-speech sounds.

3.2 Wang's audio fingerprint

The most known application for song identification is Shazam. We introduce this technology based on [1].

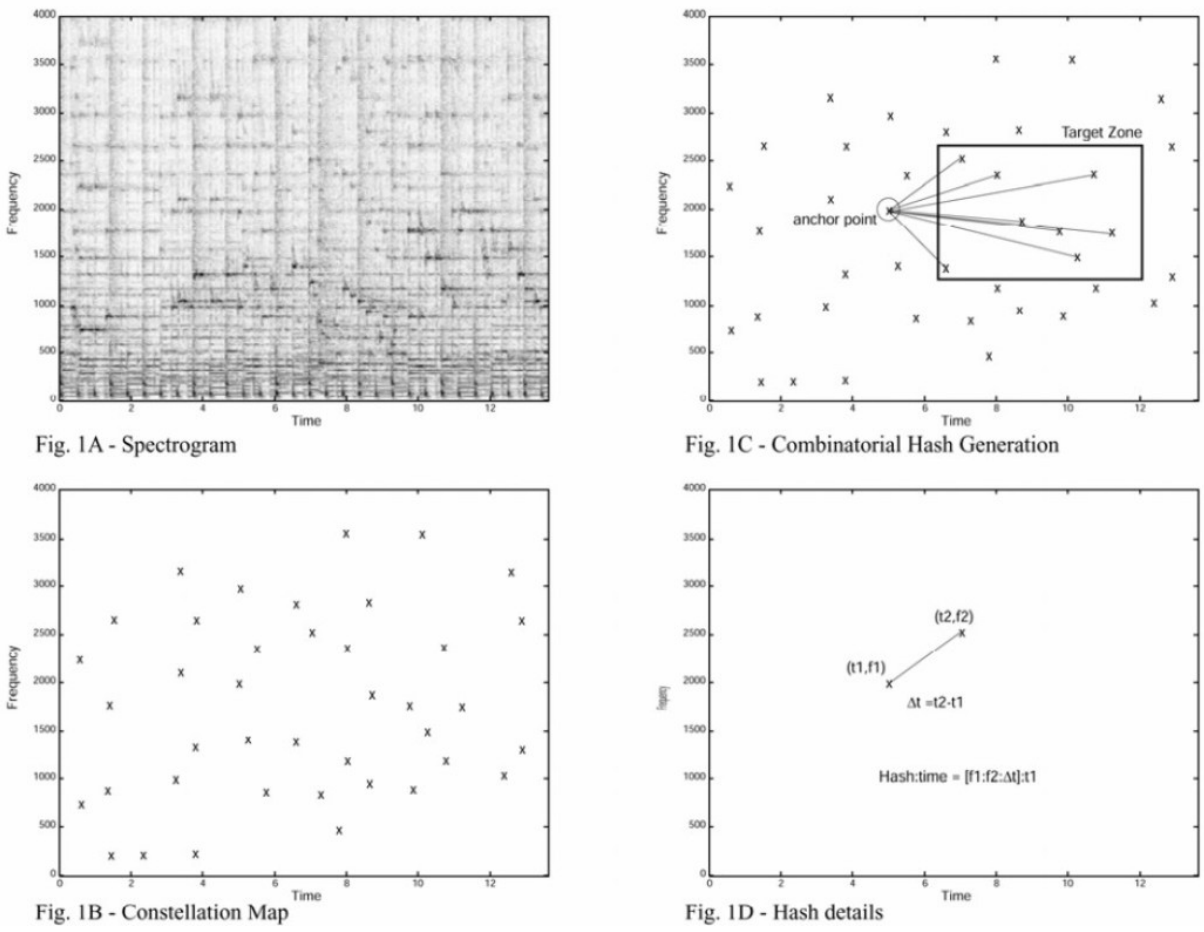

Figure 3.1. Fingerprinting method of Shazam song recognition system

The principle of spectral analysis is shown on Figure 3.1. The Figure 1A shows a spectrogram. According to it the maximum of the analyzed frequency domain is 4 kHz. The explanation is that it has to work for landline or mobile phones as well. The length of a frame is 14 ms. The first step is searching for the peaks in the spectrogram using adaptive threshold. The result of this step is shown on 1B and called constellation map, because it looks like a star field. This step is a huge data reduction. The collection of the peaks of the spectrogram is similar to the sheet music that is why it can be used for the identification.

The constellation map can be used for the recognition by sliding the unknown and the reference signal. If several points are matching it is likely that the source is the same. This solution is insensitive to the differences from the coding, linear distortion or frequency highlighting. This is a really slow process. To speed it up some or all of the peaks are chosen to be anchor points, each anchor point having a target zone associated with it. Each anchor point is sequentially paired with points within its target zone (these pairs are the landmarks), each pair yielding two frequency components plus the time difference between the points (Figure 1C and 1D). The frequencies and the time difference are 10 bit long and they make a 32 bit long hash together. This algorithm is used for every reference song and we save these hashes into a table. The time tag of the first peak is the ID of the song. To make the search as fast as possible they are sorted by the hash code. The audio fingerprints are these values inside the tables.

3.3 Song identification with Wang's method

We use the same algorithm to create the fingerprint for the input unknown song. The system searches for the same hash code in the database, and then it decides if there are enough pairs where the hash is the same and the difference between the time delays are almost the same. This difference means the starting point of the unknown signal in the reference song.

This method is very fast because of the simplicity of the audio fingerprint and the big hash tables. For example, searching in a database with 20 000 reference songs takes 5-500 ms. The speed depends on the quality of the input signal. If it is really bad quality the system has to calculate more fingerprints.

We will use a simplified version of this method with Octave/Matlab. Of course, it is way slower than the original one.

4 Evaluation of the system's performance

4.1 Evaluation of the classification

We have an input unknown signal, and we have to decide whether it is the same as the reference or not. This is a simple binary classification problem, because we have to choose between 2 classes. These 2 classes are:

- The test signal matches with the reference signal (positive)
- The test signal does not match with the reference signal (negative)

There are 4 possibilities depending on our decision:

- True Positive (TP): Our decision was positive, and it was correct
- False Positive (FP): Our decision was positive, and it was incorrect
- True Negative (TN): Our decision was negative, and it was correct
- False Negative (FN): Our decision was negative, and it was incorrect

We usually evaluate the binary classification problems with the following phrases:

- Accuracy = (TP+TN)/(TP+TN+FP+FN): rate of the correctly recognized data
- Recall = TP/(TP+FN): rate of the correctly recognized positive data
- Precision = TP/(TP+FP): rate of the real positives out of all the positives
- F-measure = 2*Precision*Recall/(Precision + Recall) Harmonic mean of precision and recall

4.2 Evaluation of the speed

If the time requirement of a method depends linearly on the amount of the input data and the input data is originated from some kind of time data we use this linear dependence to define the speed of the method. Calculating the MFCC is a method like this. Real time factor is the time needed for the calculation divided by the time length of the input data.

$$RTF = \frac{t_{calculation}}{t_{data}}$$

A small value means a fast system.

5 Scripts and functions for the lab exercises

5.1 Octave functions

We have to do most of the exercises in GNU Octave environment. This is an interpreter system similar to Matlab. It has all the basic functions of Matlab, but it is free.

All of the functions necessary to complete the lab tasks are available in a directory or installed as on Octave package. The following chapter describes these functions in a short, not fully detailed way. For most of the functions there is a detailed description that is accessible by typing “help *function-name*”.

The MFCC feature extraction and audio fingerprinting methods are from the following pages:

https://labrosa.ee.columbia.edu/matlab/rastamat/

https://labrosa.ee.columbia.edu/matlab/fingerprint/

Y = wawread (FILENAME)

Loads a waw file with the name FILENAME, the samples are going to the Y vector. If the wav is a multi-channel file Y is going to be a matrix.

For example: ***y_lvnp = wavread('tracks/lvnp.wav');***

PLAYER = audioplayer (Y, FS)

Creates a player object, which plays the waveforms from Y vector with FS sampling frequency.

For example: ***player_lvnp=audioplayer(y_lvnp, 16000);***

play (PLAYER)

Starts an audio playback with the given object that is created with the audioplayer() function. During the lab work this command or a sound playback software can be used to listen to the samples.

plot (X, Y, FMT)

Draws a diagram on the current subplot of the current figure. The coordinates of the displayed points are the elements of X and Y vector. This command can run with different syntaxes, more info in help. For example the X parameter can be omitted.

For example: ***plot(e, '.-')***

tic

Starts a timer.

VAL = toc

Returns or - when not used in assignment statement - writes to the screen the time elapsed since the timer was started. We can measure the execution time of functions.

For example: ***tic;e=match(mfcc_whole_clean, mfcc_part_mic);toc;***

[S, F, T] = specgram (X, N, FS, WINDOW, OVERLAP)

Creates a spectrogram. N is the length of the frames and the number of point for FFT. The step of the sliding frames is OVERLAP samples or if it is missing then N/2. It is doing a WINDOW type of windowing or “Hanning” by default before FFT.

For example: ***as_whole_clean=abs(specgram(y_whole_clean, 512));***

specgram_plot (AS)

Displays the absolute value of the spectrum created by specgram function.

For example: ***specgram_plot(as_whole_clean)***

[cepstra, aspectrum, pspectrum] = melfcc (samples, sr, varagin)

Performs MFCC feature extraction of the given waveform. It needs the sampling frequency of the waveform vector, and several other parameters of the feature extraction process, see help. The 2. and 3. output is the output of the filterbank and power spectrum, can be omitted, if we do not need them.

Note: .The output of this function needs to be transposed, when using it in the match() function, like the following example shows.

For example: ***mfcc_whole_clean=melfcc (y_whole_clean, 16000)'***

E = match (REF, TEST)

Slides the TEST vector along the 1st dimension of the REF vector and calculates the Euclidean distance for each alignment. If the 2nd dimension of TEST and REF is greater than 1, so they are matrices, it calculates the distance between vectors.

For example: ***e_wnm=match(mfcc_whole_clean, mfcc_part_noise)***

I = edpeak(ED)

Searches for a peak (minimum) that is steep enough in the vector of Euclidean distances created by match() function. The return value is the index of the minimum in the ED array, or 0 if no steep peak found.

For example: ***edpeak(e_wnm)***

clear_hashtable

Creates and resets (clears) the hash table used for the fingerprint searching.

[N, T] = add_tracks (N, ID)

Adds the wav files given by names in the N cell-array to the fingerprint searching system’s reference database. Performs the whole fingerprinting process and stores the computed hashes in the hash table. The 2nd parameter can be used to explicitly give identifiers to the files in the hash table, or it can be omitted. Can also be called with a waveform vector, see help. The return values are the number of added tracks and the time of the running.

For example:

tks{1}='tracks/lvnp.wav';

tks{2}='lvnp_GSM.wav';

clear_hashtable;

add_tracks(tks);

[R, L] = match_query (D, SR, IX)

Performs fingerprint based matching. Creates the fingerprints for the waveform given in D vector and searches for matches in the hash table. SR is the sampling frequency. Return values: The R variable contains the matches, L contains the matched landmarks for the reference file with IX index.

Each row of matrix R correspond the landmark matches of a reference sound sample. The columns denote the following, in order:

- Index of reference sound sample
- Number of matching landmarks
- Offset of the matching test sample, in 32 ms time units

A detailed result can be requested by giving the IX parameter. In this each row of matrix L corresponds to one of the matching landmarks of the reference sample with index IX. The columns denote the following, in order:

- Time offset of the first element of the peak pair constituting the matching landmark in the reference sound sample, in 32 ms units
- Frequency of the first element of the peak pair in 15.625 Hz units (This unit comes as the frequency step of the 8000 Hz, 512 points FFT.)
- Frequency of the second element of the peak pair in 15.625 Hz units
- Temporal distance of the 2 elements of the peak pair in 32 ms units
- The offset of the test sound sample compared to the reference sound sample which resulted the match, in 32 ms units

If we aren't interested in the details of the landmark matches, we can call this command with less parameters.

For example: ***match_query(y_part_GSM,16000)***

[DM, SRO, TK, T] = illustrate_match (DQ, SR, FL, IX)

Displays the landmarks of the test and reference sound sample on a spectrogram, highlights the matching ones. Also runs match_query before creating the figure.

It draws 2 spectrograms as shown on Figure 4.1. The spectrogram on the top depicts the test sample, the spectrogram on the bottom shows the matching part of the reference sample. The offset of the bottom spectrogram corresponds to the match, and the displayed time interval is the same as on the top spectrogram. Two circles connected by a line represent the landmark (pair of peaks on the spectrogram). Often these lines are connected, creating a longer because 1 peak can be an element of several pairs. The matching landmarks are green, the other landmarks are red.

Input parameters:

- DQ: waveform of the test sound sample
- SR: sampling frequency
- FL: cell-array of the names of the files in the database
- IX: index of the match to be displayed

Output variables:

- DM: part of the reference sound sample that matches with the test sample
- SRO: sampling frequency
- TK: identifier of the matching reference sound sample
- T: time offset of the match in the reference sound sample, in 32 ms units

For example:

illustrate_match(y_part_mic,16000,tks);

colormap(1-gray)

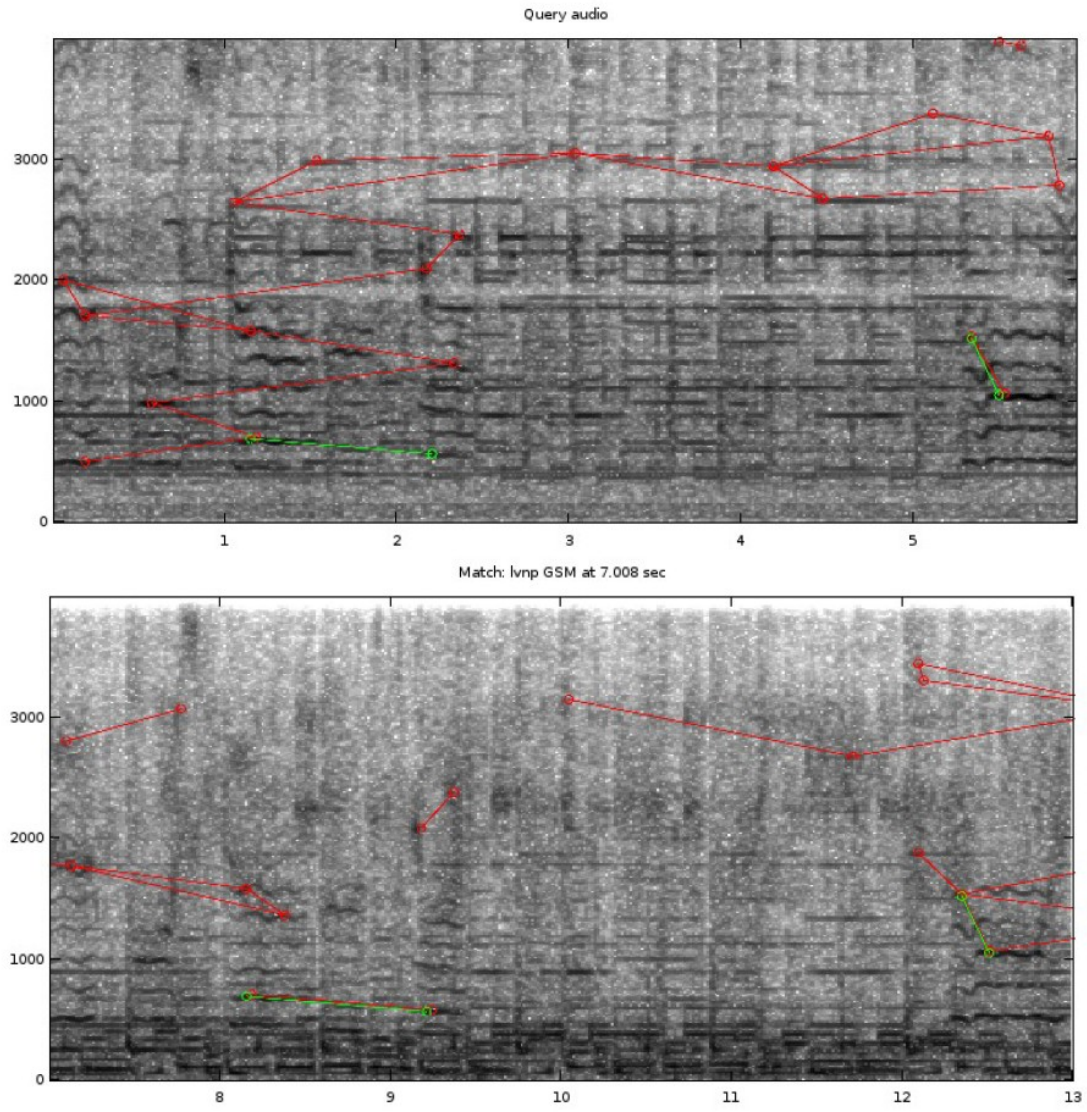

Figure 5.1. Illustration of a match on a spectrogram

X = csvread (FILENAME)

Loads a CSV (comma-separated-values) text file into a matrix.

For example:

ct=csvread ('cross_test_2000.csv');

#{ and #}

These are not functions but markers for commenting multiple lines. It can be convenient during the lab exercises to enclose the already solved parts of the exercise between these markers. So we keep the code, but avoid running all the code again.

Note: These markers should be written exclusively in a line, no could should be written before or after #{ and #} in the same line.

For example:

#{

 # Exercise 1 a.

x=y+1;

 # etc.

#}

 # Exercise 1 b.

x=y+2;

disp (X)

Displays the value of its parameter. The parameter can be a variable or a literal text between double quotes. Useful for labelling the data when displaying the results.

For example:

t=toc;

disp("Running time");

disp(t);

5.2 SOX scripts

SOX is a free command line tool for audio processing. It has quite an extensive set of functions, it's "the Swiss Army knife of audio manipulation", as said by its creators. You should use it in the laboratory exercises to apply various modifications, distortions on the test audio samples.

It is not a necessary to fully understand sox to complete the lab tasks. There are some small scripts in the ***sox_scripts*** directory that can perform the audio processing you will need. Here follows the description of these scripts:

gain.sh

Parameters:
1. name of the input wav file
2. gain in dB
3. name of the output wav file

Amplifies the input by the gain factor given in dB, i.e. multiples the input waveform with the (linear value of the) gain. We can cause overdrive distortion or clipping with a gain value so high that the result of the multiplication doesn't fit in the fixed point number representation.

For example: ***sox_scripts/gain.sh tracks/lvnp.wav 20 lvnp_overdrive.wav***

highpass.sh

Parameters:
1. name of the input wav file
2. cutoff frequency
3. name of the output wav file

Applies a highpass filter on the input signal with the given cutoff frequency.

For example: ***sox_scripts/highpass.sh tracks /lvnp.wav 4000 lvnp_highpass.wav***

gsm_encoder.sh

Parameters:
1. name of the input wav file
2. name of the output wav file

Performs GSM encoding on the input signal. It means compression with data loss and causes deterioration of the sound quality. Then it transforms back into the original format (16 kHz, 16 bit, PCM), so that the format of the output file will be the same as the input file.

For example: ***sox_scripts/gsm_encoder.sh tracks/lvnp.wav lvnp_gsm.wav***

mix.sh

Parameters:
1. name of the input wav file
2. name of the wav file to mix with
3. name of the output wav file

Mixes 2 waveforms together, that is summarizes them sample by sample. The length of the output will be the same as the length of the file given in the first parameter. So when adding noise, it should be put in the second parameter.

Note: We can adjust the volume of the noise by calling gain.sh first.

For example: ***sox_scripts/mix.sh tracks/lvnp.wav noises/SIGNAL019-20kHZ-2min_16k.wav lvnp_speech.wav***

mp3_encoder

Parameters:
1. name of the input wav file
2. name of the output wav file

Performs MP3 encoding on the input signal. It means compression with data loss and causes deterioration of the sound quality. The parameters of the encoder are set to a very high compression level, for the sake of producing significantly bad sound quality. After converting to MP3, the script transforms the sound back into the original format (16 kHz, 16 bit, PCM), so that the format of the output file will be the same as the input file.

For example: ***sox_scripts/mp3_encoder tracks/lvnp.wav lvnp_mp3.wav***

speed.sh

Parameters
1. name of the input wav file
2. speed parameter
3. name of the output wav file

Modifies the speed of the input wav file with the given rate. This means stretching or shrinking the waveform, so the pitch will change as well as the tempo. If the speed parameter is greater than 1, the result gets faster, if less than 1, the result gets slower.

For example: ***sox_scripts/speed.shtracks/lvnp.wav 1.3 lvnp_high.wav***

tempo.sh

Parameters
1. name of the input wav file
2. tempo parameter
3. name of the output wav file

Modifies the tempo of the input wav file with the given rate. It pefromrs a processing so that only the tempo will change, the pitch remains unaltered. If the tempo parameter is greater than 1, the gets faster, if less than 1, the result gets slower.

For example: ***sox_scripts/tempo.sh tracks/lvnp.wav 1.3 lvnp_fast.wav***

whitenoise.sh

Parameters
1. name of the input wav file
2. amplitude of the noise
3. name of the output wav file

Generates white noise with the given amplitude and adds it to the input waveform.

For example: ***sox_scripts/whitenoise.sh trcks/lvnp.wav 0.3 lvnp_white.wav***

trim.sh

Parameters:
1. name of the input wav file
2. start of cut
3. length of the cut
4. name of the output wav file

Cuts a part from the input wav file.

For example: ***sox_scripts/trim.sh tracks/lvnp.wav 7 6 lvnp_part.wav***

6 Bibliography

[1] Wang, Avery, et al., „An Industrial Strength Audio Search Algorithm”, In: ISMIR. 2003. p. 7-13 (shazam)

[2] P. Cano, E.Batlle, T. Kalker, J. Haitsma, „A Review of Audio Fingerprinting”, Journal of VLSI Signal Processing 41, 2005, pp. 271–284

[3] H. B. Kekre, Nikita Bhandari, Nisha Nair, Purnima Padmanabhan, Shravya Bhandari, „A Review of Audio Fingerprinting and Comparison of Algorithms” International Journal of Computer Applications, Volume 70 - Number 13, pp. 24-30

[4] Pang-Ning Tan, Michael Steinbach, Vipin Kumar „Bevezetés az adatbányászatba”, TAMOP 4.2.5 Book Database, http://www.tankonyvtar.hu/en/tartalom/tamop425/0046_adatbanyaszat/ch05s07.html