Algorithm for real-time comparison of audio streams for broadcast supervision

Mateusz Lorkiewicz, Jakub Stankowski, Krzysztof Klimaszewski
Chair of Multimedia Telecommunications and Microelectronics,
Poznań University of Technology, Poland
jstankowski@multimedia.edu.pl

Abstract - **The paper deals with an efficient method of comparison of audio streams that can be compressed using lossy compression algorithms and, as well, delayed by an unspecified number of samples due to transmission and processing. The algorithm bases on the signal envelope correlation between audio streams . The performance of the algorithm is evaluated using different acceleration methods available in modern desktop computers.**

Keywords **- audio comparison, envelope correlation**

I. INTRODUCTION

Media broadcasting requires continuous monitoring of the quality of the broadcasted content. Any error in transmitted stream is immediately visible (or audible) for all users. The errors in transmission are mainly caused by noise or interference, but they may occur also due to other, less obvious reasons. Mistakes at the stage of stream preparation are an example of those reasons. Therefore, in order to preserve high quality of services, there is a need for real time supervision of the transmitted streams. For example, it is critical to determine if the headend transmits correct signal to the subscribers.

This paper concentrates on algorithm developed for automatic and unattended supervision of audio streams for broadcasting purposes. The most important part of supervision system is an algorithm that is used to determine similarity or dissimilarity of audio streams.

Such an algorithm needs to meet some key requirements that would distinguish it from typical off-line audio samples comparison. The most important requirements are mostly connected with a real-time operation:

- the algorithm has to operate in real time, on live broadcasted signals,
- the algorithm has to operate without excessive buffering, since it is impossible or impractical to store longer part of track and process it offline,
- the error in broadcasted stream has to be detected as soon as possible with the lowest possible delay in order to reduce the time when erroneous signal is transmitted,
- the computational complexity has to be low enough to allow real-time operation, even when a higher number of audio streams has to be analyzed.

Besides previously mentioned difficulties, the broadcast applications are inherently connected with media format conversion, media encoding and transcoding. Therefore, the audio comparison algorithms have to meet additional requirement:

- algorithm must be resilient for signal modifications introduced by transcoding.

In practice, two different types of transcoding must be considered: between different bit rates (homogenous transcoding) or between different compression techniques (heterogenous transcoding). Most of audio compression techniques uses subbands and/or operate in MDCT domain [1] and use psychoacoustic models to remove some imperceptible parts of signal [2].

As a result of such compression, the decoded waveform is completely different than the original one, which introduces additional difficulties as two signals have to be compared. Audio compression with very high compression ratio (very low bitrate) leads to significant distortion of original signal and lack of data in most subbands.

There are many audio comparison algorithms that are commonly used nowadays. The simplest method is to perform the comparison of audio data sample by sample or to calculate the cross-correlation between a reference and a compared sample. The main disadvantage is that even a little difference between given samples due to the delay, transcoding or volume change can make those methods completely unreliable.

Some solutions use “acoustic fingerprints” matching to compare audio data [3]. This comparison algorithm bases on finding frequencies with the highest amplitude on the generated spectrogram. Then a sequence of the found “fingerprints” is compared to a sequence extracted from other audio signal and algorithm decision is made. This method is used in many sound recognition applications like *Shazzam*.

Other approach towards the audio comparison process is to take human sound perception into account. The compared audio data is processed by an algorithm that returns perceptual “signature” of the received data. Such descriptors are compared in pairs and the result of similarity is given on a percentage scale[4]. The *compatronix* sound matching software operates using this solution.

The main disadvantage of such audio comparison algorithms is the required length of audio data. To give reliable decision, the aforementioned algorithms require at least 5 to 10 seconds of data. In broadcasted streams supervision, such a big delay is

unacceptable. What is more, the algorithms are sensitive to the degradation of data quality (caused by processing or transcoding commonly encountered in TV broadcast) lowers the algorithm reliability. Another drawback is that those solutions do not give information about mutual delay of compared audio data, what may be crucial in audio and video synchronization supervising. Moreover, mentioned algorithms have been designed for offline usage and are inappropriate for a real-time comparing scenario.

The research towards real-time comparison of audio streams have been inspired by Telewizja Puls, the third largest commercial TV network in Poland, provider of two nation-wide channels: TV Puls and PULS 2, having a reach of 35.5 million and 29.1 million viewers, respectively (October 2017; 4+).

II. PROPOSED ALGORITHM

The proposed algorithm is based on a simple approach with cross-correlation calculated between the reference and the compared audio chunk. Despite sharing common idea with already mentioned straightforward approach, the algorithm has been designed to take all broadcast specific requirements into account.

In order to improve reliability, the comparison between signals is performed based on signal envelope (as opposed to raw sample values). Using signal envelope instead of raw samples gives immunity to signal degradation due to the fact that shape of the signal envelope is usually well preserved by the most common processing and encoding operations.

A. Algorithm description

In order to improve reliability, the comparison between signals is performed based on real envelope of each signal (as opposed to raw sample values). Using signal envelope instead of raw samples gives independence from signal degradation due to the fact that signal envelop is free from sudden signal changes.

The first step of proposed algorithm is the derivation of signal envelope. To avoid Hilbert transformation calculation, real signal envelope may be estimated by low pass filtering of rectified signal and subtracting the mean value of filtered data. In order to preserve reasonable computational complexity, we designed a 32-tap FIR low-pass filter with normalized cutoff frequency equal to 0.25 (Figure 1.). In this step two envelopes for the reference and the compared chunk are calculated.

$$E_R(n) = envelope\big(S_R(n)\big) \tag{1.1}$$

$$E_C(n) = envelope\big(S_C(n)\big) \tag{1.2}$$

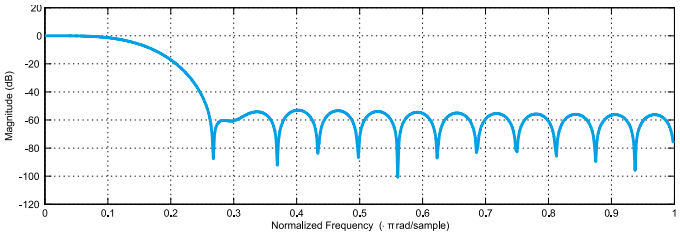

Figure 1. Amplitude characteristic of low-pass FIR filter used during envelope calculation process

Next step is to use previously calculated envelopes to calculate cross-correlation $CC(n)$ between the reference and the compared signal envelopes.

$$CC(n) = xcorr(E_R, E_C) \tag{2}$$

In the next step the correlogram $CC(n)$ resulting from the previous step enables a decision on audio similarity. In order to decide whether given audio signals are the same, the algorithm calculates the correlation peak descriptor, described in details in the next point. Based on its value, the decision is made. To make algorithm decision more reliable, the power of signal is computed prior to any further calculations. If the received power is too low, further calculations are not performed (decision about similarity cannot be determined).

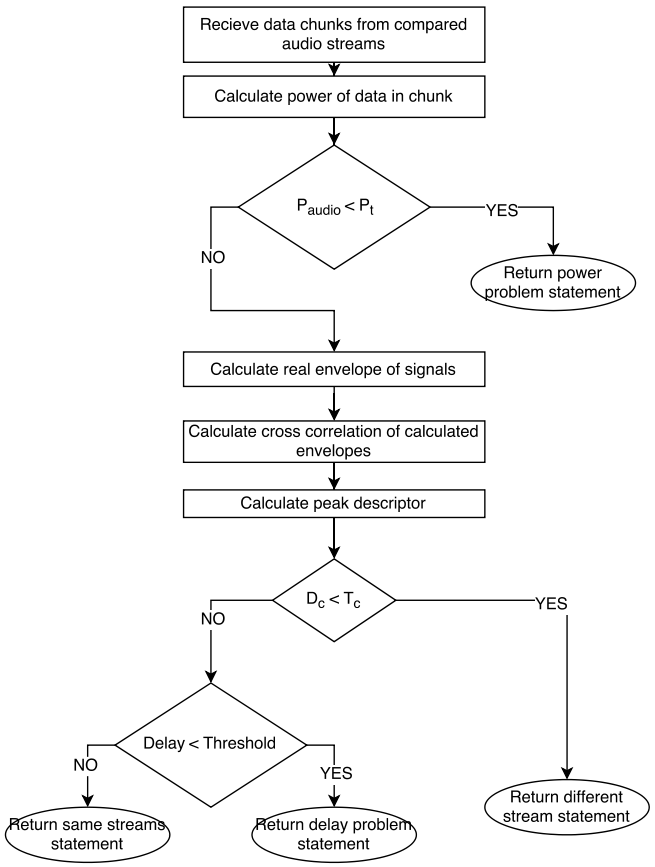

Figure 2. Algorithm block diagram.

The described method is flexible, as it allows processing of audio data fragments with variable number of samples. Data chunks length may vary, depending on the requirements. Obviously a low number of samples in chunk results in the inevitable decrease of the algorithm reliability.

B. Signal similarity or dissimilarity detection

The comparison decision is taken based on signal envelope cross correlation correlogram $CC(n)$. If signals contain the same (or similar) perceivable content, resultant correlogram will have characteristic shape described as spiky peak – sharply rising to extreme and suddenly falling. Huge signal similarity will result

in single dominant peak in the whole correlogram. In case of different audio streams, the shape of cross correlation result, as well as the number of peaks, will vary and will not have a single dominant peak (Figure 3. .

In order to decide if audio signals are the same, the algorithm must check if a single peak has appeared on correlogram and calculate its descriptor.

For a correlogram with a single peak, the maximum of correlogram will be much higher than for most of the samples. The mean value of the correlogram will be relatively small. To ensure a single narrow peak, the algorithm also calculates the standard deviation to check dispersion of values which will be small in case of the dominant peak existence in the correlogram. Due to the fact that correlogram values may be both positive and negative, it is possible that the mean value will be zero or very low. In order to avoid that, all calculation use correlogram's absolute values.

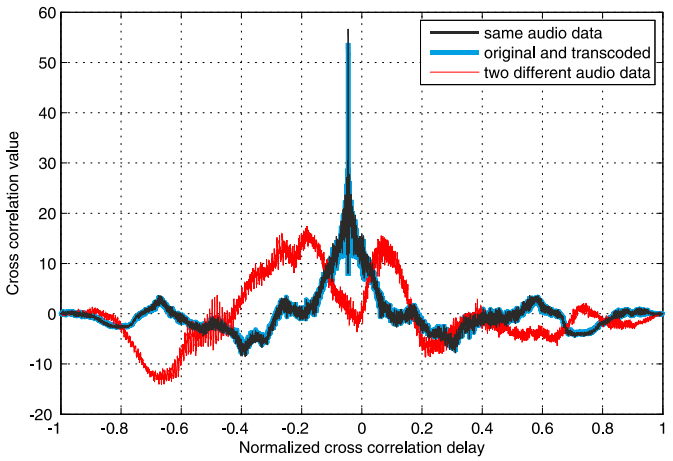

Figure 3. Exemplary envelope correlograms for the same (black), same but compressed (blue) and diferent (red) signals.

In case of a peak in cross correlation result, we expect high maximum value CC_{max}, small mean value CC_{μ} and small standard deviation CC_{σ}. Based on that observation, we can create a correlation peak descriptor D_c given by the following formula:

$$D_c = \frac{CC_{max}}{CC_{\mu}+CC_{\sigma}} \tag{3}$$

After calculating descriptor value, the algorithm compares it with a defined threshold T_c. If the value is higher than the threshold, the compared audio data is recognized as the same.

C. Delay calculation

If the audio chunks are found to convey the same content, it is also desirable to find the delay between the audio streams. Due to the fact that algorithm uses cross correlation in order to measure similarity, the correlogram $CC(n)$ could be used to estimate sample delay between reference and compared chunk. If algorithm recognizes the given data to be the same, it is also possible to calculate the delay between those audio chunks. Audio streams delay may be calculated by finding the position of maximum value of cross correlation result. This position corresponds to the delay represented in audio sample periods. In order to receive the mutual delay in seconds, the division by data sample frequency is needed.

D. Power thresholding

In the broadcasted audio stream, the appearance of silent moments in a content is common, for example during long pauses in speech. In those moments, an audio signal is mostly noise. Any transcoding would almost certainly modify this signal significantly. For processed silence periods with noise, the correlogram is usually flat, with no dominant peaks. The algorithm described above may erroneously inform in such a case that the two streams with silence periods are different. In order to avoid such situations, the algorithm calculates the power of the given signals, prior to any other action. If the power of any of the compared signals is lower than a given threshold, the silence moment is detected and information about this situation is returned, determining that no reliable comparison can be made. Additionally, the information about audio streams power is very useful in broadcast supervising, for example to inspect power difference between audio stream channels and to detect the missing audio content in the encoded audio stream.

III. Implementation

In order to evaluate the performance of the proposed algorithm, a test implementation has been prepared. As a first step, a software framework has been developed. The description of the framework is not in the scope of this paper. It is sufficient to mention here that the framework was designed as a multi-format software data receiver with the following functionalities:

- reading audio track from a file,
- reading "Transport Stream over IP" streams,
- receiving and demultiplexing of transmitted streams,
- audio track decoding,
- synchronization of audio frames,
- performing audio stream comparison using the described algorithm,
- managing all execution threads and used buffers.

The proposed algorithm has been implemented as highly optimized, multithreaded C++ code. All time consuming operations related to audio samples processing, like:

- sample format conversion (from native 16 bit signed integer to normalized single precision floating point),
- calculation of envelope (low-pass filtering, absolute value calculation),
- calculation of cross-correlation,
- calculation of average value from selected samples,
- calculation of standard deviation,
- searching for maximum value,

have been carefully optimized and implemented in three different versions:

- portable implementation using plain C++ programming language,
- x86 exclusive, highly optimized implementation using SIMD instructions from older 128 bit SSE extensions (SSE, SSE2, SSE3 and SSSE3 extension sets), allowing to process 4 samples at once,
- x86 exclusive, highly optimized implementation using 256 bit SIMD instructions from modern AVX extensions (AVX and FMA extension sets), allowing to process 8 samples at once.

The preferred version could be selected on compile time according to instruction sets available on target computer.

The calculation of cross-correlation has been implemented in Discrete Fourier Transform (DFT) domain. The calculation of forward and inverse transform if performed by Fast Fourier Transform (FFT) algorithm[5]. The publicly available FFT implementation called FFTW ("Fastest Fourier Transform in the West") [6][7] has been used. In order to speed up calculations, special version of FFT and iFFT routines optimized for real signals has been selected. Moreover, calculation of cross-correlation in DCT domain requires additional calculations like: multiplication by complex conjugative and *fftshift*-like coefficient/samples rearrangement. Both operations have been implemented in similar manner as sample processing related operations.

IV. EXPERIMANTAL EVALUATION

Experimental evaluation has been performed by testing the implemented algorithm using a wide set of different audio streams. The tests were performed using different digital TV streams with audio. The evaluation has been divided into two steps: first concentrated on the algorithm performance and reliability, the second step was performed to measure the computational complexity of the implementation.

A. *Algorithm reliability*

To check algorithm reliability, the set of television audio streams had been prepared. Recorded audio streams duration ranged from 4 up to 8 minutes. Prepared data set had different audio characteristic, for example football match, soap opera, movie etc. Every audio stream was transcoded using various coding formats (MPEG-1 Audio Layer 3[8], Ogg Vorbis[9] and Advanced Audio Coding (AAC)[10]) and bitrates (96, 128, 192 and 320 kbit/s). A single test consisted of the following steps:

- choosing randomly one original audio stream,
- choosing randomly if a second stream content should be same or different,
- randomly choosing a second stream to compare: transcoded version of a first stream in case of same content or not transcoded stream with different content,
- randomly choose the time offset of the stream. Second stream is additionally delayed by 2205 samples,
- verification of algorithm response.

During experiment, the silence detection (power thresholding) has not been activated. The chunk size was set to 48384 samples. This size was dictated by an integer number of transport stream packets that contain around 1 second of the reconstructed audio (42*1152 samples).

Algorithm decision was checked for various descriptor thresholds. On (Figure 2.) the percentage of correct algorithm decisions for 50 000 tests is presented, as a function of the decision threshold value. The decision threshold is used to compare it to the descriptor value, as described in chapter II B.

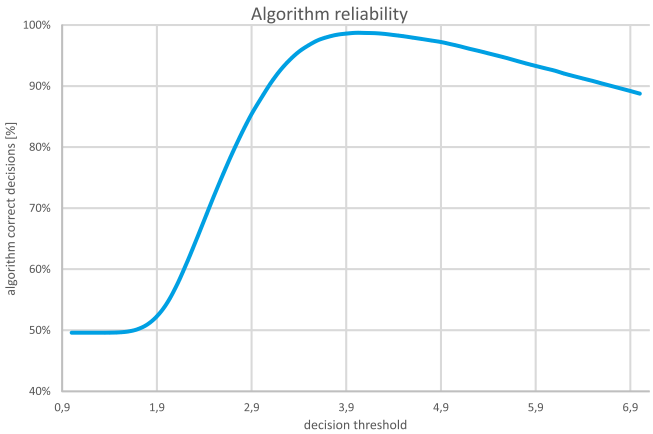

Figure 4. Algorithm correct decision percentage according to decision threshold (for 50 000 tests, 1 second chunk).

B. *Computational complexity*

The computational complexity of algorithm implementation has been measured indirectly by calculating the computation time. The experiments have been performed in the following conditions. The system was set to compare two audio tracks with parameters typical for broadcasting purposes. Each audio track contained 2 channels (stereo) with 48kHz sampling rate and 16 bits per sample resolution. Experiments have been performed for a set of different comparison chunk lengths (1, 2, and 4 seconds). Moreover, each type of implementation (C++, SSE and AVX) was examined. Tests were performed on typical desktop computer with modern quad core CPU operating at ~4GHz and capable of executing AVX instructions (Intel® Skylake microarchitecture). Test result are summarized in TABLE I.

TABLE I. IMPLEMENTATION PERFORMANCE

Implementation	**Calculation time [milliseconds] 48kHz, 16bit, 2 channels (stereo)**		
	1 second chunk	***2 seconds chunk***	***4 seconds chunk***
AVX	3.247	7.051	14.190
SSE	3.849	7.701	15.235
C++	6.788	13.747	27.728

The computation time is strongly correlated with chunk length. It's worth noticing that transform (FFT and iFFT) calculation is significant constituent in computation time, therefore algorithm computation time resembles the $O(n \cdot \log(n))$ complexity relation known for FFT.

The usage of vector instructions allows for significant reduction in computation time, i.e. AVC implementation is approximately two times faster than plain C++ code.

The computational complexity evaluation clearly shows that algorithm has reasonably low computational complexity and allows for real-time operation. Moreover, the low computational complexity allows for simultaneous comparison of several dozen audio pairs on examined computer.

V. CONCLUSIONS

The presented algorithm proves to be an efficient way of performing the automated supervision of the audio data streams, especially useful for situations where the same signal is recoded or transcoded and inserted into different output streams. In such cases, mistakes in configuration of the process are likely to happen, and a way of quick automatic verification is a good method of ensuring the proper assignment of the streams.

The algorithm, especially in its optimized version, is able to process several different audio streams at the same time in real time, thus it is able to monitor the assignment of audio streams for a whole set of different outputs. Even as much as 100 different streams can be compared at once (depending on the settings of the chunk length and available hardware capabilities) using a standard desktop personal computer.

Also the problem of noise influence and silence periods in the signal are handled correctly by the algorithm. Therefore, it is possible to detect the situations of missing audio. The number of situations when, due to transcoding of silence periods, the audio streams could be categorized as different ones, can be significantly reduced.

ACKNOWLEDGMENT

The presented work has been partially funded by the Polish Ministry of Science and Higher Education for the status activity consisting of research and development and associated tasks supporting development of young scientists and doctoral students in 2018 in Chair of Multimedia Telecommunications and Microelectronics.

REFERENCES

[1] H. S. Malvar, "Lapped Transforms for Efficient Transform/Subband Coding", IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. 38, no. 6, pp. 969–978 (Equation 22), June 1990

[2] E. Zwicker and H. Fastl, “Psychoacoustics: Facts and Models”, Springer-Verlag, Berlin Heidelberg 1990.

[3] A. Wang et al., “An industrial strength audio search algorithm.,” ISMIR, Washington, DC, 2003, pp. 7–13.

[4] S. V. Rice and S. M. Bailey, "General-Purpose Real-Time Monitoring of Machine Sounds," in Essential Technologies for Successful Prognostics: Proceedings of the 59th Meeting of the Society for Machinery Failure Prevention Technology, Virginia Beach, VA, 2005

[5] W. T. Cochran *et al*., "What is the fast Fourier transform?," in *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664-1674, Oct. 1967.

[6] M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," Proceedings of the IEEE 93 (2), 216–231 (2005). Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation.

[7] M. Frigo, "A Fast Fourier Transform Compiler," Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation, Atlanta, Georgia, May 1999.

[8] "ISO/IEC 13818-3:1995 – Information technology — Generic coding of moving pictures and associated audio information — Part 3: Audio". ISO. 1995.

[9] Xiph.Org Foundation, "Vorbis I specification". Xiph.Org Foundation, February 27, 2015

[10] "ISO/IEC 13818-7:1997, Information technology -- Generic coding of moving pictures and associated audio information -- Part 7: Advanced Audio Coding (AAC)