Aurio: Audio Processing, Analysis and Retrieval

Mario Guggenberger
Institute of Information Technology
Alpen-Adria-Universität Klagenfurt
9020 Klagenfurt am Wörthersee, Austria
mg@itec.aau.at

ABSTRACT

Aurio is an open source software library written for audio-based processing, analysis, and retrieval of audio and video recordings. The novelty of this library is the implementation of a number of fingerprinting and time warping algorithms to retrieve, match and synchronize media streams, which no other library currently offers. It is designed with simplicity, performance and versatility in mind, can be easily integrated into .NET applications, and offers a collection of many basic signal processing methods. It can read many file formats, offers multiple export abilities for further processing, and contains various UI widgets for graphical applications. Built upon the Aurio library, *AudioAlign* is an additionally released open source application for the (semi-)automatic synchronization of media recordings.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*reusable libraries*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

Keywords

Audio, features, fingerprinting, dynamic time warping, synchronization, retrieval, processing, open source software

1. INTRODUCTION

The widespread usage of portable and ubiquitous multimedia capable devices that allow easy recording and thus creation of new content has led to an abundance of media recordings that are either kept in the local archives of users or shared across various online platforms. The big question is how all these files can be managed, and while this can be manually done if an archive is small, it gets tedious or even overwhelming when it grows. This is an area where the multimedia retrieval community is actively researching for new helpful tools and methods, and many tools and methods with high impact have been proposed in recent years.

MM'15, October 26–30, 2015, Brisbane, Australia.
© 2015 ACM. ISBN 978-1-4503-3459-4/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2733373.2807408.

For audio content management, the probably most helpful tool is fingerprinting [3], enabling music discovery tools that tell which song is playing on the radio [23], improving music library management through metadata retrieval [22] and duplicate detection, and helping content creators with broadcast monitoring. Researchers have also shown its use for the synchronization of multiple media recordings captured at an event [17,19]. Another older but nonetheless interesting invention is dynamic time warping [20] for the nonlinear alignment of data series, that can be used to synchronize different song interpretations [4] or to follow scores [1].

We think that these two methods fit well together, and that there is a lack of easy to use libraries that implement these methods. We therefore present *Aurio*, an audio library that implements these methods and makes them easily accessible, on top of a collection of basic reusable building blocks that allow for rapid experimentation, prototyping and implementation of additional high-level features and methods.

2. ARCHITECTURE

Aurio[1] is a library written in C# for the .NET Framework, and designed with simplicity, performance, and versatility in mind. Due to the nature of managed .NET code, this library cannot be as effective as purely native code, but critical sections are implemented in unsafe code to minimize the impact and overhead of the managed runtime.

The core module *Aurio* contains classes for file I/O, 32-bit floating point block-based stream processing, audio playback, various data structures, utility functions, FFT, resampling, audio features, and various audio matching algorithms, described in more detail in Section 3. Aurio comes with a few additional modules that are essentially interfaces to 3rd party native libraries that it makes use of. *Aurio.FFmpeg* handles file input and support of various audio and video (container) formats through the well-known FFmpeg[2] library. *Aurio.PFFFT* interfaces with the PFFFT[3] library for FFT calculation, and there is optional support for the FFTW library [10] through *Aurio.FFTW* and the purely managed Exocortex.DSP library[4]. For resampling, Aurio uses the Sox Resampler[5] through *Aurio.Soxr* by default, while libsamplerate[6] is optionally available through

[1]`https://github.com/protyposis/Aurio`
[2]`http://ffmpeg.org/`
[3]`https://bitbucket.org/jpommier/pffft`
[4]`http://exocortex.org/dsp/`
[5]`http://sourceforge.net/p/soxr/`
[6]`http://www.mega-nerd.com/SRC/`

Aurio.LibSampleRate. Reading and writing of wave files, and audio playback is handled through NAudio[7]. The optional module *Aurio.WaveControls* depends on Aurio and provides multiple audio-related UI widgets.

Although written for Windows, it should be portable to the Mono framework[8] for use on other OS' like Linux, OS X and possibly Android for which almost all external dependencies are available. The two exceptions are the optional UI widget module based on the Windows Presentation Foundation API and the audio playback functionality using the NAudio library, which are both Windows-only. The library supports x86 and x64 architectures and automatically loads the appropriate native dependencies.

3. FEATURES

The stream-based audio processing engine in Aurio works similar to conventional I/O stream implementations in Java and .NET. All available streams implement a common interface and can be chained together to achieve certain goals, through which complex processing can be achieved very easily and logically. A chain of streams must start with a source where audio data is initially read from, continue with intermediate processing or analysis streams, and optionally end in a sink stream where the final data is written to. Currently supported sources are wave files through NAudio, various file formats through FFmpeg, raw memory, and a sine generator. Intermediate streams are available for concatenation of multiple streams, cropping of start and end positions, (double) buffering, data monitoring (e.g. for online calculation of spectrograms and correlations), mixing multiple input streams into one output stream, mono down- and up-conversion, time offsetting, phase inversion, resampling, time warping for nonlinear time stretching, visualization with seamless zooming, and volume clipping, metering, balancing and leveling. All higher level features and algorithms in Aurio build upon this interface and require a stream as input, and all streams and features check the format of its input stream and reject it with an exception and a meaningful message in case it is not supported.

Conversion from the time into the frequency domain can be achieved through one of the FFT implementations, for which multiple utility functions for magnitude calculation, normalization and scale conversion are available. The library also provides all commonly used window functions, and a class for automatic sequential windowing of sample streams. Based on the windowing are implementations for calculating spectrograms with the short-time Fourier transform (STFT), calculating chromagrams [2], and the Continuous Frequency Activation feature [21] for music detection.

The included data structures are commonly used buffers, moving average calculation methods, a undirected graph implementation with minimal spanning tree and connected component calculation, and various memory efficient matrix representations including the diagonal matrix and two kinds of sparse matrices.

3.1 Fingerprinting and Matching

The key features of Aurio that distinguish the library from other audio processing and retrieval libraries are the fingerprinting and matching methods.

[7]`http://naudio.codeplex.com/`
[8]`http://www.mono-project.com/`

Fingerprinting systems all share the same basic idea of transforming a stream of audio samples from known tracks into a series of hashes that are stored in a hash table and later used for lookup operations with hashes from unknown tracks [3]. The differences lie in the way the hashes are calculated and matched, but even there, many ideas are shared. Four fingerprinting methods are currently implemented. The first one is a well-known and widely used method usually referred to as "Philips fingerprinting", developed by Haitsma and Kalker at Philips Research [15]. Hashes are calculated by simply detecting onsets in the spectrogram and mapping them to a 32-bit integer bitmap, which proved to be working very well with noisy data. The second is probably the most famous algorithm by Wang because of its use in the popular smartphone app Shazam, therefore usually referred to as the "Shazam algorithm" [23]. Compared to the first method, Wang uses an intelligent approach to achieve a reduced number of more sparsely distributed hashes with greater entropy by selecting only the most prominent peaks that most likely survive high noise ("constellation map"), and forming pairs between each peak and the peaks in a defined target zone. The hashes are a concatenation of the two frequency bin indices and the temporal distance of a pair's peaks, resulting in 24-bit data stored in a 32-bit integer. Tests have shown even better resilience to noise than the Philips approach. A more recent method is Echoprint by Ellis et al. [8], built for The Echo Nest corporation that has been acquired by Spotify. It differentiates itself from Wang's method by first whitening the stream, then splitting the spectrum into 8 sub-bands, and detecting peaks and forming peak pairs separately in each band. Hashes are formed from the distances between two consecutive pairs' peaks and their band index, truncated to 20 bits but stored in 32-bit integers. Their distribution is even sparser with an average distance of one second between peaks. Its original application is music identification with over-the-air support, but compared to the previous two methods it does not work very well with high noise. The fourth method is named Chromaprint [18] and is based on ideas from [2, 16]. Chromaprint computes a chromagram, smooths and normalizes it over time, and then applies 16 classifiers frame by frame consisting of a filtering and quantization step, whose results are concatenated to a 32-bit hash. This method has been developed for audio file identification and is used by the MusicBrainz project [22] for metadata tagging. It is not designed for noise and over-the-air applications.

All methods are exactly documented, except for Wang's which is only described conceptually but has an unofficial open source library available [6]. For the Philips method, all crucial parameters are specified in the original paper [15], Echoprint and Chromaprint even offer official open source libraries [18, 25]. Although there are implementations available, Aurio reimplements all methods from scratch to allow for stream-based processing of arbitrarily long input streams with constant memory usage. It also provides the ability to parameterize most variable settings through profiles, and each method comes with at least a default profile. The Echoprint and Chromaprint default profiles are compatible with the official libraries and produce the same hash output.

For retrieval, Aurio provides an accompanying hash storage for each fingerprinting method, backed by a hash table or SQLite[9]. Generated hashes can be written to the storage, which can then be queried for matches. Because Aurio is optimized for media synchronization, the hash stores do not just return a result telling if one track matches another, but they return exact matching positions of hashes within tracks. This essentially means that Aurio can be used to determine which temporal position within one track matches a temporal position within another track.

The accuracy of a match strongly depends on the granularity of the hashes, e.g. the Philips method returns much more accurate matches than Echoprint using the default settings. Aurio implements various methods for calculating the cross-correlation between stream segments, which can be used to improve the accuracy up to the sample level [13]. It also implements the common dynamic time warping method [20], and the improved on-line time warping [5] method. Given a single matching point, both can be used to calculate a series of matching points over the whole runtime of stream pairs, which effectively provides means for nonlinear synchronization, and the detection of time drift [12], cuts and drop-outs.

3.2 UI Widgets

To support building applications with graphical user interfaces, Aurio.WaveControls offers a range of widgets commonly found in audio and video analysis and editing tools. The *spectrogram* widget renders spectrograms and other 2D -grams, e.g. chromagrams, with configurable color gradients. The *graph* widget renders audio waves, windows, spectra and other kinds of value series. The *wave view* widget is the most elaborate control that can render arbitrarily long audio tracks with scrolling and continuous zooming between the whole track and single samples, and offers bitmap- and vector-based rendering modes. Additional widgets include a time scale, correlation meter, VU meter, and volume control slider.

3.3 Demo Applications

Included with the library are multiple tools, test applications and unit tests that serve as examples on how the library can be used. Among the interesting demos is a benchmark of the fingerprinting methods, which runs a file through all algorithms and prints the required computation time. It is useful for optimizing and monitoring the impact of changes in the algorithms and their many parameters. Another is a multi-track media player where multiple files can be mixed together and played back through a simple GUI. A demo application for the Philips fingerprinting algorithms allows the graphical inspection of the hash table and calculated fingerprints. For the Shazam fingerprinting, there is a demo that displays the constellation map on a spectrogram.

4. AUDIOALIGN

AudioAlign[10] is a Windows application shown in Figure 1, jointly developed with and built upon the Aurio library. Its main purpose is the (semi-)automatic synchronization of audio and video recordings. Other use-cases include the evaluation of fingerprinting algorithms and their configuration parameters, comparison of file sources, analysis of differences

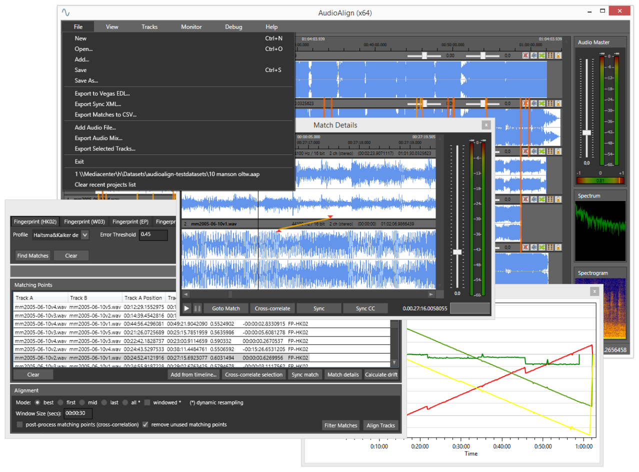

Figure 1: AudioAlign main window, file menu options, matching window, alignment graph, and match detail window.

in distinct live performances of artists, automatic synchronization of multiple audio tracks of different languages or formats, the synchronization of different cover interpretations of a musical piece, and dubbing of vocal recordings. It provides a simple multitrack timeline editor similar to common digital audio editors and video cutting tools, but does not offer direct audio editing abilities. Instead, it serves as a front end to Aurio's fingerprinting and matching algorithms, uses almost all of the core library's functionality and all the UI widgets, and is additionally designed to speed up processing by parallelizing tasks over multiple CPU cores.

To synchronize multiple recordings of an event, they can be dragged into the timeline or added through the menu. Because Aurio supports many file formats, it can also load videos downloaded from online video platforms like YouTube. Once the timeline is populated, tracks can be shifted across the scrollable and zoomable timeline which displays the audio waveforms. This way, tracks can be synchronized manually by specifying synchronization points through the *Match & Align* window. Playback can be started from any position within the timeline, and controls attached to each track allow basic balancing and mixing of parallel tracks to inspect their synchronization by listening. An instance of the application with its timeline and all its settings makes up a project that can be saved and loaded through the file menu. When a desirable synchronization has been achieved, a mixdown of the whole timeline or of separate selected tracks can be rendered into audio files. The timeline can also be exported to the Sony Vegas Pro video editor[11], where presynchronized video clips can easily be further processed, e.g. to create a multicamera cut. Of course, the real value is not the manual timeline adjustment but the available algorithms. A populated timeline can be automatically synchronized from the Match & Align window by first executing a fingerprinting algorithm, after which the discovered matching points are displayed on the timeline, and then carrying out an alignment step where the matching points are filtered by adjustable criteria and the tracks are temporally adjusted on the timeline. Relations between tracks and their

[9] `http://sqlite.org/`
[10] `https://github.com/protyposis/AudioAlign`
[11] `http://www.sonycreativesoftware.com/vegaspro`

matching points can be inspected in the *Alignment Graph* window that is also shown in Figure 1. Additional and more fine-grained matching points can be obtained through the dynamic time warping and cross-correlation options.

AudioAlign has already been presented as a demo [11] and has been used to create a synchronization ground truth for the Jiku Mobile Video Dataset [13]. Its sources are made available together with Aurio to serve as a reference application for the Aurio library.

5. LICENSE & PATENTS

Both Aurio and AudioAlign are available as open source software under the GNU Affero General Public License Version 3 [9]. Aurio contains methods protected by patents that can legally only be used under certain conditions. In Europe, these conditions are private use and research. For other uses, all users are encouraged to clarify the legal situation before using these methods. This particularly applies to the Philips and Shazam fingerprinting methods which are protected in many regions of the world [14, 24], and Echoprint which is protected in the US [7].

6. REFERENCES

[1] A. Arzt. Score following with dynamic time warping. Master's thesis, Vienna University of Technology, Vienna, Austria, 2008.

[2] M. Bartsch and G. Wakefield. Audio thumbnailing of popular music using chroma-based representations. *Multimedia, IEEE Transactions on*, 7(1):96–104, Feb 2005.

[3] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of audio fingerprinting. *Journal of VLSI signal processing systems for signal, image and video technology*, 41(3):271–284, 2005.

[4] S. Dixon. An on-line time warping algorithm for tracking musical performances. In *Proceedings of the 19th international joint conference on Artificial intelligence*, IJCAI'05, pages 1727–1728, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[5] S. Dixon. An on-line time warping algorithm for tracking musical performances. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1727–1728, 2005.

[6] D. Ellis. Robust landmark-based audio fingerprinting. Web resource: `http://labrosa.ee.columbia.edu /matlab/fingerprint/` , 2009. Accessed 2015-05-20.

[7] D. Ellis and B. Whitman. Musical fingerprinting based on onset intervals. US Patent US20130139673, June 6 2013. Priority date Dec. 2 2011.

[8] D. Ellis, B. Whitman, and A. Porter. Echoprint - an open music identification service. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, Miami, Florida, 2011.

[9] Free Software Foundation, Inc. GNU Affero General Public License Version 3 (AGPLv3). Web resource: `http://www.gnu.org/licenses/agpl-3.0.txt`, Nov. 19 2007. Accessed 2015-05-20.

[10] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[11] M. Guggenberger, M. Lux, and L. Böszörmenyi. Audioalign - synchronization of A/V-streams based on audio data. In *Multimedia (ISM), 2012 IEEE International Symposium on*, pages 382–383, 2012.

[12] M. Guggenberger, M. Lux, and L. Böszörmenyi. An analysis of time drift in hand-held recording devices. In *MultiMedia Modeling*, volume 8935 of *Lecture Notes in Computer Science*, pages 203–213. Springer International Publishing, 2015.

[13] M. Guggenberger, M. Lux, and L. Böszörmenyi. A synchronization ground truth for the jiku mobile video dataset. In *MultiMedia Modeling*, volume 8936 of *Lecture Notes in Computer Science*, pages 87–98. Springer International Publishing, 2015.

[14] J. Haitsma, A. Kalker, C. Baggen, and J. Oostveen. Generating and matching hashes of multimedia content. WIPO publication WO/2002/065782, Aug. 22 2002. Priority date Feb. 2 2001.

[15] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR)*, Paris, France, 2002.

[16] D. Jang, C. Yoo, S. Lee, S. Kim, and T. Kalker. Pairwise boosted audio fingerprint. *Information Forensics and Security, IEEE Transactions on*, 4(4):995–1004, Dec 2009.

[17] L. Kennedy and M. Naaman. Less talk, more rock: Automated organization of community-contributed collections of concert videos. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 311–320, New York, NY, USA, 2009. ACM.

[18] L. Lalinský. AcoustID Chromaprint. Web resource: `https://acoustid.org/chromaprint`, 2011. Accessed: 2015-05-20.

[19] A. Llagostera Casanovas and A. Cavallaro. Audio-visual events for multi-camera synchronization. *Multimedia Tools and Applications*, pages 1–24, 2014.

[20] M. Müller. Dynamic time warping. In *Information Retrieval for Music and Motion*, pages 69–84. Springer Berlin Heidelberg, 2007.

[21] K. Seyerlehner, G. Widmer, T. Pohle, and M. Schedl. Automatic music detection in television productions. In *Proceedings of the Int. Conf. on Digital Audio Effects (DAFx-2007)*, 2007.

[22] A. Swartz. Musicbrainz: A semantic web service. *Intelligent Systems, IEEE*, 17(1):76–77, Jan 2002.

[23] A. L.-C. Wang. An industrial strength audio search algorithm. In *Proceedings of the Fourth International Conference on Music Information Retrieval (ISMIR 2003)*, pages 7–13, 2003.

[24] A. L.-C. Wang and J. O. Smith III. Method for search in an audio database. WIPO publication WO/2002/011123, Feb. 7 2002. Priority date Aug. 31 2000.

[25] B. Whitman, A. Porter, D. Ellis, et al. Echoprint Codegen. Web resource: `http://echoprint.me/codegen`, 2011. Accessed 2015-05-20.