Olaf: a lightweight, portable audio search system

Joren Six

1 IPEM, Ghent University, Belgium

Summary

Olaf stands for **Overly Lightweight Acoustic Fingerprinting** and solves the problem of finding short audio fragments in large digital audio archives. The content-based audio search algorithm implemented in Olaf can identify a short audio query in a large database of thousands of hours of audio using an acoustic fingerprinting technique.

The dataflow in Olaf closely resembles the flow depicted in Figure 1. Audio is transformed to features, which are grouped into recognizable fingerprints. The fingerprints are compared with a database of reference fingerprints. If a match is found, it is reported or, in case of a true negative, the system reports that the audio is not present in the database. The properties of the acoustic fingerprinting system mainly depend on the selection of features, the information captured by the fingerprints, and the performance of the matching step.

The fingerprints of Olaf are based on peaks in a spectral representation of audio, an audio feature which has been proven to be a good candidate for audio matching (Six, 2020, 2022; Six & Leman, 2014; Avery L. Wang, 2003; A. Li-chun Wang & Culbert, 2003). Olaf combines either two or three spectral peaks into a fingerprint. Two-peak fingerprints allow matching shorter queries and improves matching noisy queries. The limited information in two peaks becomes a problem if the reference dataset becomes larger and false positive matches – fingerprint hash collisions – become more common. Three-peak fingerprints contain more bits of information, which makes false positives much less common. Matches become more reliable but shorter, or more distorted queries, might be missed. Three-peak fingerprints work well for longer or ‘cleaner’ queries in combination with larger reference datasets.

Statement of need

Audio search algorithms have been described for decades (Cano et al., 2005; Fenet et al., 2011; Haitsma & Kalker, 2002; Herre et al., 2002; Sonnleitner & Widmer, 2014; Avery L. Wang, 2003) but have not been accessible for researchers due to the lack of proper, scalable, freely-available implementations. Olaf solves this problem by providing an acoustic fingerprinting system that
can be used by researchers for digital music archive management, audio-to-audio alignment on embedded devices, or other uses.

Six et al. (2018) and Six et al. (2023) describe the applications of acoustic fingerprints for digital music archive management. These range from meta-data quality verification – through the identification of duplicates – to merging archives with potential duplicate material. A less straightforward application of Olaf is audio-to-audio alignment and synchronization (Six & Leman, 2015, 2017). In that case the matching fingerprints are used to align, e.g., multiple video recordings of the same event, by aligning the audio attached to each video.

On a more meta-level Olaf also facilitates Music Information Retrieval (MIR) and acoustic fingerprinting research. Audio duplicate detection can be used to clean up and evaluate machine learning datasets (Weck & Serra, 2023). Olaf can also serve as a baseline for specific acoustic fingerprinting cases such as broadcast audio monitoring (Cortès et al., 2022).

The portability and low memory usage of Olaf allow it to run on microcontrollers such as the ESP32, RP2040, or similar chips with at least 250 kB memory. The memory usage to run a 20-second query on an index containing one hour of audio is less than 512 kB\(^1\). The embedded variant takes considerably less memory since it does not need the key-value store and has a smaller index. This unique feature facilitates innovative IoT music recognition and music synchronization applications. Olaf also runs in the browser. A compilation emits a WebAssembly binary, which, together with the Web Audio API, enables browser based acoustic fingerprinting applications.

Alternative systems with available implementations are by Chang et al. (2021), Panako\(^2\) by Six (2022), audfprint by Ellis (2014), PeakFP by Cortès et al. (2022), ChromaPrint by Lalinský & contributors (2023), SpectroMap by López-García et al. (2022) and Dejavu by Drevo (2020). All have a different focus and trade-offs but none offer the portability to target browsers or have the low memory usage to target microcontrollers.

**Design**

Simplicity and maintainability are two keywords in the design of Olaf. The code aims to be as readable and simple as possible. The code uses an object-oriented inspired approach to organize the ANSI C11 (ISO, 2011) code. Opaque structs are used to store and encapsulate state information. Polymorphism is implemented by having a header file defining an interface which is then implemented in different ways in source files. The choice of implementation is done at compile time. For example, the database used by Olaf can be a key-value store or an in-memory database. To provide this functionality, the interface olaf.db.h has two implementations which provide either the key-value store or the in-memory database. The modularity of Olaf makes it relatively straightforward to stack Olaf’s building blocks for use on embedded devices, browsers and traditional computers.

C has a long history, which should allow Olaf to stand the test of time. Arguably C is the most portable programming language and has been around for decades and will be available for decades to come. Boring technology enables longevity. Developing bug-free code in the C programming language is notoriously challenging. However, many bugs have been found by running Olaf on different platforms / contexts - Windows, browsers, embedded devices, musl - and, thanks to continuous integration. Each time Olaf is updated a battery of functional tests are executed automatically.

For traditional computers, file handling and transcoding are governed by a companion Ruby

\(^1\)A script to measure memory use can be found here: https://github.com/JorenSix/Olaf/blob/master/eval/olaf_memory_use.rb

\(^2\)Panako and Olaf implement similar algorithms, one in Java and the other in C. A direct comparison shows that Olaf is more than twice as fast as Panako: https://github.com/JorenSix/Olaf/tree/master/eval
This script expands lists of incoming audio files, transcodes audio files, checks incoming audio, checks for duplicate material, and validates arguments and input. The Ruby script essentially makes Olaf an easy-to-use CLI application and keeps the C parts of Olaf simple. The C core it is not concerned with, e.g., transcoding. Crucially, the C core trusts input and does not do much input validation and does not provide many guardrails. Since the interface is the Ruby script, this seems warranted.

Olaf depends on two C libraries: a key-value store and an FFT library. LMDB (Chu, 2022) serves as a high performance key-value store. PFFFT (Pommier, 2022) is used to speed up FFT calculations. Additionally a hash table and a dequeue data structure are included from c-algorithms (Howard, 2020). Internal documentation follows the DoxyGen (Heesch, 2023) standards. Two papers give the rationale behind the algorithms (Six & Leman, 2014; Avery L. Wang, 2003). Olaf can be compiled and installed using the make tool or with Zig (Kelley & contributors, 2023) cross-compiler.

To try Olaf yourself or adapt Olaf for your needs, the code and documentation of Olaf is hosted in a publicly-available GitHub repository.

Acknowledgements

Development of Olaf is partially funded by the Ghent University BOF Project PaPiOM.

References


