




Olaf: a lightweight, portable audio search system

Joren Six ¹

¹ IPEM, Ghent University, Belgium

DOI: [10.21105/joss.05459](https://doi.org/10.21105/joss.05459)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Brian McFee](#)  

Reviewers:

- [@liscio](#)
- [@ebezzam](#)

Submitted: 13 March 2023

Published: 29 June 2023

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

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 in 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.

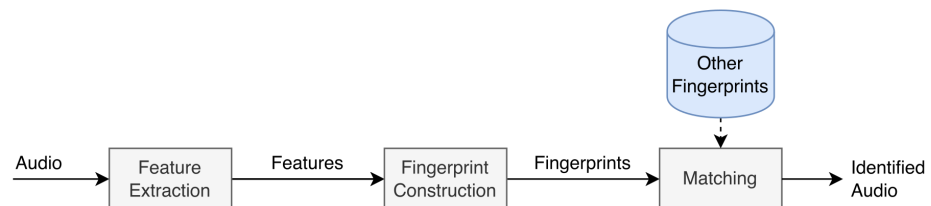


Figure 1: A general acoustic fingerprinting system. Features are extracted from audio and combined into fingerprints. The fingerprints are matched with fingerprints in a reference database. Finally, a match is reported.

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 which 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 250kB memory. The memory usage to run a 20s query on an index containing one hour of audio is less than 512kB¹. 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), Panako² 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. C also provides a lot of exiting footguns, of which I made ample use. 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 script. This script expands lists of incoming audio files, transcodes audio files, checks incoming

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

²Panako 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>

audio, checks for duplicate material, 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

- Cano, P., Batlle, E., Kalker, T., & Haitsma, J. (2005). A review of audio fingerprinting. *The Journal of VLSI Signal Processing*, 41, 271–284. <https://doi.org/10.1007/s11265-005-4151-3>
- Chang, S., Lee, D., Park, J., Lim, H., Lee, K., Ko, K., & Han, Y. (2021). Neural audio fingerprint for high-specific audio retrieval based on contrastive learning. *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3025–3029. <https://doi.org/10.1109/ICASSP39728.2021.9414337>
- Chu, H. (2022). LMDB: A general-purpose programming language and toolchain for maintaining robust, optimal, and reusable software. In *OpenLDAP repository*. OpenLDAP. <https://git.openldap.org/openldap/openldap/tree/mdb.master>
- Cortès, G., Ciurana, A., Molina, E., Miron, M., Meyers, O., Six, J., & Serra, X. (2022). BAF: an audio fingerprinting dataset for broadcast monitoring. *Proceedings of the 23rd International Society for Music Information Retrieval Conference (ISMIR 2022)*, 908–916. <https://doi.org/10.5281/zenodo.7343030>
- Drevo, W. (2020). Dejavu: Audio fingerprinting and recognition algorithm implemented in python. In *GitHub repository*. Github. <https://github.com/worldveil/dejavu>
- Ellis, D. (2014). The 2014 LabROSA audio fingerprint system. *MIREX abstracts of the 15th International Symposium on Music Information Retrieval (ISMIR 2014)*.
- Fenet, S., Richard, G., & Grenier, Y. (2011). A Scalable Audio Fingerprint Method with Robustness to Pitch-Shifting. *Proceedings of the 12th International Symposium on Music Information Retrieval (ISMIR 2011)*, 121–126. <https://doi.org/10.5281/zenodo.1417593>
- Haitsma, J., & Kalker, T. (2002). A highly robust audio fingerprinting system. *Proceedings of the 3th International Symposium on Music Information Retrieval (ISMIR 2002)*. <https://doi.org/10.5281/zenodo.1417973>
- Heesch, D. van. (2023). Doxygen: The de facto standard tool for generating documentation from annotated c++ sources. In *Doxygen website*. Doxygen. <https://www.doxygen.nl/>

- Herre, J., Hellmuth, O., & Cremer, M. (2002). Scalable robust audio fingerprinting using MPEG-7 content description. *Multimedia Signal Processing, 2002 IEEE Workshop on*, 165–168. <https://doi.org/10.1109/MMSP.2002.1203273>
- Howard, S. (2020). C algorithms: A collection of common computer science algorithms. In *GitHub repository*. GitHub. <https://github.com/fraggle/c-algorithms>
- ISO. (2011). *ISO/IEC 9899:1999 Information technology — Programming languages — C*. International Organization for Standardization.
- Kelley, A., & contributors. (2023). Zig: A general-purpose programming language and toolchain for maintaining robust, optimal, and reusable software. In *GitHub repository*. GitHub. <https://github.com/ziglang/zig>
- Lalinský, L., & contributors. (2023). Chromaprint: An audio fingerprint library developed for the AcoustID project. In *GitHub repository*. Github. <https://github.com/acoustid/chromaprint>
- López-García, A., Martínez-Rodríguez, B., & Liern, V. (2022). A proposal to compare the similarity between musical products. One more step for automated plagiarism detection? In M. Montiel, O. A. Agustín-Aquino, F. Gómez, J. Kastine, E. Lluís-Puebla, & B. Milam (Eds.), *Mathematics and computation in music* (pp. 192–204). Springer International Publishing. https://doi.org/10.1007/978-3-031-07015-0_16
- Pommier, J. (2022). PFFFT: A pretty fast FFT. In *BitBucket repository*. BitBucket. <https://bitbucket.org/jpommier/pffft/src/master/>
- Six, J. (2020). OLAF: Overly lightweight acoustic fingerprinting. *Extended abstracts for the Late-Breaking Demo Session of the 21st International Society for Music Information Conference (ISMIR 2020)*.
- Six, J. (2022). Panako: A scalable audio search system. *Journal of Open Source Software*, 7(78). <https://doi.org/10.21105/joss.04554>
- Six, J., Bressan, F., & Leman, M. (2018). Applications of duplicate detection in music archives: From metadata comparison to storage optimisation. In G. Serra & C. Tasso (Eds.), *Digital libraries and multimedia archives* (pp. 101–113). Springer International Publishing. https://doi.org/10.1007/978-3-319-73165-0_10
- Six, J., Bressan, F., & Renders, K. (2023). Duplicate detection for digital audio archive management: Two case studies. In A. Biswas, E. Wennekes, A. Wiczorkowska, & R. H. Laskar (Eds.), *Advances in speech and music technology: Computational aspects and applications* (pp. 311–329). Springer International Publishing. https://doi.org/10.1007/978-3-031-18444-4_16
- Six, J., & Leman, M. (2014). Panako - A scalable acoustic fingerprinting system handling time-scale and pitch modification. *Proceedings of the 15th ISMIR Conference (ISMIR 2014)*, 1–6. <https://doi.org/10.5281/zenodo.1416190>
- Six, J., & Leman, M. (2015). Synchronizing multimodal recordings using audio-to-audio alignment: An application of acoustic fingerprinting to facilitate music interaction research. *Journal on Multimodal User Interfaces*, 9, 223–229. <https://doi.org/10.1007/s12193-015-0196-1>
- Six, J., & Leman, M. (2017). A framework to provide fine-grained time-dependent context for active listening experiences. *Audio Engineering Society Conference: 2017 AES International Conference on Semantic Audio*.
- Sonnleitner, R., & Widmer, G. (2014). Quad-based audio fingerprinting robust to time and frequency scaling. *Proceedings of the 17th International Conference on Digital Audio Effects (DAFx-14)*.

- Wang, Avery L. (2003). An industrial-strength audio search algorithm. *Proceedings of the 4th International Symposium on Music Information Retrieval (ISMIR 2003)*, 7–13. <https://doi.org/10.5281/zenodo.1416340>
- Wang, A. Li-chun, & Culbert, D. (2003). Robust and invariant audio pattern matching (Patent No. US7627477 B). In *Patent US7627477 B* (US7627477 B).
- Weck, B., & Serra, X. (2023). Data leakage in cross-modal retrieval training: A case study. *arXiv e-Prints*, arXiv–2302. <https://doi.org/10.48550/arXiv.2302.12258>