Panako: a scalable audio search system

Joren Six

1 IPEM, Ghent University, Belgium

Summary

Panako solves the problem of finding short audio fragments in large digital audio archives. The content based audio search algorithm implemented in Panako is able to identify a short audio query in a large database of thousands of hours of audio using an acoustic fingerprinting technique.

The dataflow in Panako 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.

There are three algorithms implemented in Panako. The first is based on pitch class histograms Six & Cornelis (2012). The other two are based on peaks in a spectral representation: a baseline algorithm (Six, 2020; Avery L. Wang, 2003) and the main Panako algorithm (Six & Leman, 2014; Six, Joren, 2021; A. Li-chun Wang & Culbert, 2003).

The main algorithm in Panako finds matches between a short query fragment and the audio in the database even if the query is time-stretched, pitch-shifted, slowed down or sped up while maintaining other desirable features such as scalability, robustness and reliability. This is important since changes in replay speed do occur commonly, they are either introduced by accident during an analog to digital conversion or are introduced deliberately.

Accidental audio speed changes are often the result of varying or uncalibrated recording or playback speeds of analogue physical media such as wax cylinders, wire recordings, magnetic tapes or gramophone records. To identify duplicates in a digitized archive, a music search algorithm should be robust against replay speed. Deliberate speed manipulations are sometimes introduced during radio broadcasts: occasionally songs are played a bit faster to fit into a time-slot. During a DJ-set speed changes are almost always present. To correctly identify audio in these cases as well, a music search algorithm must be robust against pitch shifting, time stretching and speed changes.

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. Panako solves this problem by providing an acoustic fingerprinting system which can be used by researchers for DJ-set analysis, digital music archive management and audio-to-audio alignment.

In DJ-set analysis the aim is to automatically identify music in sets and how that music was modified and mixed (Kim et al., 2020; Schwarz & Fourer, 2018; Sonnleitner et al., 2016). With its robustness for time-stretch/pitch-shift and speed-up Panako is ideally suited to gather large scale insights in DJ performance practice.

Six et al. (2018) describes 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 Panako 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.

Alternative systems with available implementations are by Chang et al. (2021) and audfprint by Ellis (2014). Both systems however lack robustness to significant speed changes of more than 5%. Note that there are two implementations of the audfprint system: there is a MatLab\(^1\) and Python\(^2\) version. The matlab version has support for small amounts time scaling. The details on how Panako handles time-scaling are described in two papers (Six & Leman, 2014; Six, Joren, 2021).

Design

Simplicity and maintainability are two keywords in the design of Panako. The code aims to be as readable and simple as possible. The second version of Panako is a complete rewrite to ensure this simplicity while still keeping query and computational performance in check.

Relying on conservative platforms with a long history of backwards compatibility should allow Panako to stand the test of time. Panako is developed in Java and targets the long term support release Java SE 11. Panako also relies on software in C and C++. Java, C and C++ have been around for decades and it is reasonable to assume that these platforms will be supported for decades to come. Boring technology enables longevity.

Next to Java 11, Panako depends on three libraries: a DSP library, a key-value store and a spectral transform library. The first is a pure Java DSP library called TarsosDSP\(^3\) (Six et al., 2014). LMDB\(^4\) is used as a high performance key-value store. It is a C library and accessible through lmdbjava. The third and final dependency is JGaborator\(^5\): a wrapper around the Gaborator\(^6\) library which implements a constant-Q non-stationary Gabor transform in C++11 (Velasco, Gino Angelo and Holighaus, Nicki and Dörfler, Monika and Grill, Thomas, 2011). The last two have native compiled parts and need to be ported to new or exotic platforms if the need arises. The transition to aarch64 (Apple M1), for example consisted of a straightforward compilation step and repackaging of this native library. Panako can be

\(^{1}\)https://www.ee.columbia.edu/~dpwe/resources/matlab/audfprint/
\(^{2}\)https://github.com/dpwe/audfprint
\(^{3}\)https://github.com/JorenSix/TarsosDSP
\(^{4}\)https://www.symas.com/lmdb
\(^{5}\)https://github.com/JorenSix/JGaborator
\(^{6}\)https://gaborator.com
containerized and the Docker file supports both ARM and x86 platforms and always compiles these native dependencies.

The code of Panako is hosted in a publicly available GitHub repository. Internal documentation follows the JavaDoc standards. Two papers give the rationale behind the algorithms (Six & Leman, 2014; Six, Joren, 2021). Panako can be installed using a Gradle wrapper script which automatically downloads the Gradle build system if it is not present on the system. The Gradle wrapper compiles and installs Panako.

Acknowledgements

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

References


