TarsosDSP, a Real-Time Audio Processing Framework in Java

Joren Six[1,2], Olmo Cornelis[2], Marc Leman[1]

[1]*University Ghent, IPEM, Sint-Pietersnieuwstraat 41, 9000, Gent, Belgium*

[2]*University College Ghent, School of Arts, Jozef Kluyskensstraat 2, 9000 Gent, Belgium*

Correspondence should be addressed to Joren Six (`joren.six@ugent.be`)

ABSTRACT

This paper presents TarsosDSP, a framework for real-time audio analysis and processing. Most libraries and frameworks offer either audio analysis and feature extraction or audio synthesis and processing. TarsosDSP is one of a only a few frameworks that offers both analysis, processing and feature extraction in real-time, a unique feature in the Java ecosystem. The framework contains practical audio processing algorithms, it can be extended easily, and has no external dependencies. Each algorithm is implemented as simple as possible thanks to a straightforward processing pipeline. TarsosDSP's features include a resampling algorithm, onset detectors, a number of pitch estimation algorithms, a time stretch algorithm, a pitch shifting algorithm, and an algorithm to calculate the Constant-Q. The framework also allows simple audio synthesis, some audio effects, and several filters. The Open Source framework is a valuable contribution to the MIR-Community and ideal fit for interactive MIR-applications on Android.

1. INTRODUCTION

Frameworks or libraries[1] for audio processing can be divided into two categories.The first category offers audio analysis and feature extraction. The second category offers audio synthesis capabilities. Both types may or may not operate in real-time. Table 1 shows a partial overview of notable audio frameworks. It shows that only a few frameworks offer real-time feature extraction combined with synthesis capabilities. To the best of the authors' knowledge, TarsosDSP is unique in that regard within the Java ecosystem. The combination of real-time feature extraction and synthesis can be of use for music education tools or music video games. Especially for development on the Android platform there is a need for such functionality.

TarsosDSP also fills a need for educational tools for Music Information Retrieval. As identified by Gomez in [14], there is a need for comprehensible, well-documented MIR-frameworks which perform useful tasks on every platforms, without the requirement of a costly software package like Matlab. TarsosDSP serves this educational goal, it has already been used by several master students as a starting point into music information retrieval[5, 32, 28].

[1]The distinction between library and framework is explained in [2]. In short, a framework is an abstract specification of an application whereby analysis and design is reused, conversely when using a (class) library code is reused but a library does not enforce a design.

The framework tries to hit the sweet spot between being capable enough to get real tasks done, and compact enough to serve as a demonstration for beginning MIR-researchers on how audio processing works in practice. TarsosDSP therefore targets both students and more experienced researchers who want to make use of the implemented features.

After this introduction a section about the design decisions made follows, then the main features of TarsosDSP are highlighted. Chapter four is about the availability of the framework. The paper ends with a conclusion and future work.

2. DESIGN DECISIONS

To meet the goals stated in the introduction a couple of design decisions were made.

2.1. Java based

TarsosDSP was written in Java to allow portability from one platform to another. The automatic memory management facilities are a great boon for a system implemented in Java. These features allow a clean implementation

Name	Analysis	Synthesis	Real-Time	Technology
Aubio [7]	True	False	True	C
CLAM [3]	True	True	True	C
CSL [23]	True	True	True	C++
Essentia [6]	True	False	True	C++
Marsyas [29]	True	True	False	C++
SndObj [16]	True	True	True	C++
Sonic Visualizer [11]	True	False	False	C++
STK [25]	False	True	True	C++
Tartini [19]	True	False	True	C++
YAAFE [17]	True	False	False	C++
Beads[21]	False	True	True	Java
JASS[30]	False	True	True	Java
jAudio [18]	True	False	False	Java
Jipes	True	False	False	Java
jMusic[8]	False	True	False	Java
JSyn [10]	False	True	True	Java
Minim[22]	False	True	True	Java
TarsosDSP	True	True	True	Java

Table 1: A table with notable audio frameworks. Only a few frameworks offer real-time feature extraction and audio synthesis capabilities. According to the research by the authors, in the Java ecosystem only TarsosDSP offers this capability.

of audio processing algorithms. The clutter introduced by memory management instructions, and platform dependent `ifdef`'s typically found in C++ implementations are avoided. The Dalvik Java runtime enables to run TarsosDSP's algorithms unmodified on the Android platform. Java or C++ libraries are often hard to use due to external dependencies. TarsosDSP has no external dependencies, except for the standard Java Runtime. Java does have a serious drawback, it struggles to offer a low-latency audio pipeline. If real-time low-latency is needed, the environment in which TarsosDSP operates needs to be optimized, e.g. by following the instructions found in [1].

2.2. Processing pipeline

The processing pipeline is kept as simple as possible. Currently, only single channel audio is allowed, which helps to makes the processing chain extremely straightforward[2]. A schematic representation can be found in Figure 1. The source of the audio is a file, a microphone, or an optionally empty stream. The `AudioDispatcher` chops incoming audio in blocks of a requested number of samples, with a defined overlap. Subsequently the blocks of audio are scaled to a `float` in the range `[-1,1]`. The wrapped blocks are encapsulated in an `AudioEvent` object which contains a pointer to the audio, the start time in seconds, and has some auxiliary methods, e.g. to calculate the energy of the audio block. The `AudioDispatcher` sends the `AudioEvent` through a series of `AudioProcessor` objects, which execute an operation on audio. The core of the algorithms are contained in these `AudioProcessor` objects. They can e.g. estimate pitch or detect onsets in a block of audio. Note that the size of a block of audio can change during the processing flow. This is the case when a block of audio is stretched in time. For more examples of available `AudioProcessor` operations see section 3. Figure 2.2 shows a processing pipeline. It shows how the dispatcher chops up audio and how the `AudioProcessor` objects are linked. Also interesting to note is line 8, where an anonymous inner class is declared to handle pitch estimation results. The example covers filtering, analysis, effects and playback. The last statement on line 23 boot-

[2]Actually multichannel audio is accepted as well, but it is automatically downmixed to one channel before it is send through the processing pipeline

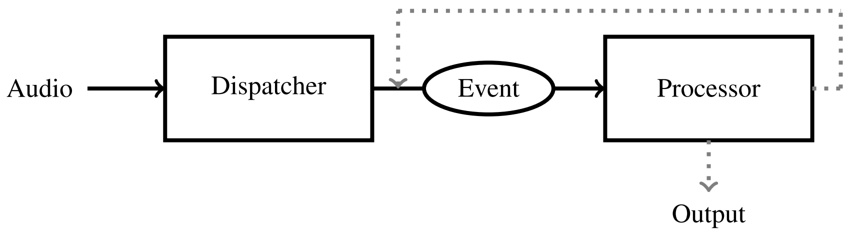


Fig. 1: A schematic representation of the TarsosDSP processing pipeline. The incoming audio (left) is divided into blocks which are encapsulated in Event objects by the Dispatcher. The event objects flow through one or more Processor blocks, which may or may not alter the audio and can generate output (e.g. pitch estimations). Dotted lines represent optional flows.

```java
//Get an audio stream from the microphone, chop it in blocks of 1024 samples, no overlap (0 samples)
AudioDispatcher d = AudioDispatcher.fromDefaultMicrophone(1024, 0);
float sr = 44100;//The sample rate
//High pass filter, let everything pass above 110Hz
AudioProcessor highPass = new HighPass(110,sr);
d.addAudioProcessor(highPass);
//Pitch detection, print estimated pitches on standard out
PitchDetectionHandler printPitch = new PitchDetectionHandler() {
        @Override
        public void handlePitch(PitchDetectionResult pitchDetectionResult,AudioEvent audioEvent) {
                System.out.println(pitchDetectionResult.getPitch());
        }
};
PitchEstimationAlgorithm algo = PitchEstimationAlgorithm.YIN; //use YIN
AudioProcessor pitchEstimator = new PitchProcessor(algo, sr,1024,printPitch);
d.addAudioProcessor(pitchEstimator);
//Add an audio effect (delay)
d.addAudioProcessor(new DelayEffect(0.5,0.3,sr));
//Mix some noise with the audio (synthesis)
d.addAudioProcessor(new NoiseGenerator(0.3));
//Play the audio on the loudspeakers
d.addAudioProcessor(new AudioPlayer(new AudioFormat(sr, 16, 1, true,true)));
d.run();//starts the dispatching process
```

Fig. 2: A TarsosDSP processing PipeLine. Here, pitch estimation on filtered audio from a microphone sample session is done in real-time. A delay audio effect is added and some noise is added to the audio before it is played back. The example covers filtering, analysis, audio effects, synthesis and playback.

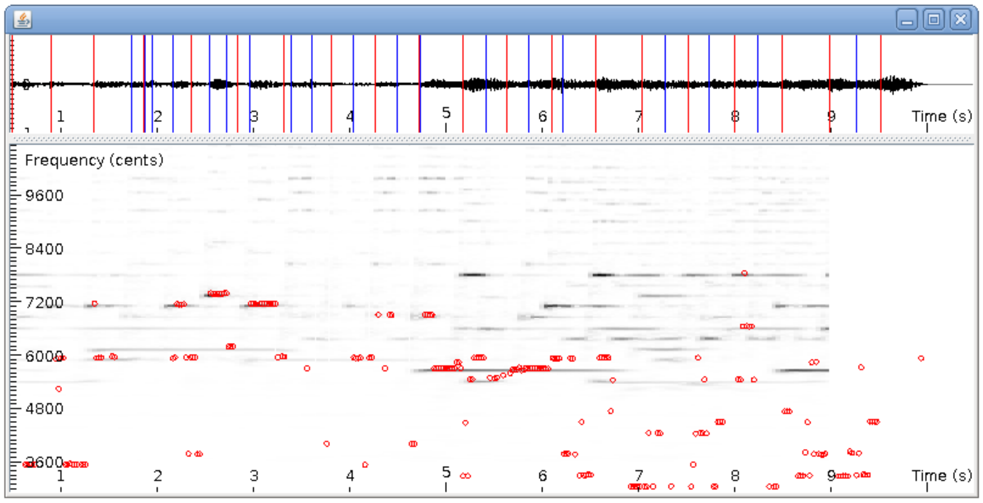


Fig. 3: A visualization of some of the features that can be extracted using TarsosDSP: a waveform (top panel, black), onsets (top panel, blue), beats (top panel, red), a Constant-Q spectrogram (bottom panel, gray), and pitch estimations (bottom panel, red). The source code for the visualisation is part of the TarsosDSP distribution as well.

straps the whole process.

2.3. Optimizations

TarsosDSP serves an educational goal, therefore the implementations of the algorithms are kept as pure as possible, and no obfuscating optimizations are made. Readability of the source code is put before its execution speed, if algorithms are not quick enough users are invited to optimize the Java code themselves, or look for alternatives, perhaps in another programming language like C++. This is a rather unique feature of the TarsosDSP framework, other libraries take a different approach. jAudio [18] and YAAFE[17] for example reuse calculations for feature extraction, this makes algorithms more efficient, but also harder to grasp. Other libraries still, like SoundTouch[3], carry a burden by being highly optimized - with assembler code - and by having a large history. These things tend to contribute to less readable code, especially for people new in the field.

3. IMPLEMENTED FEATURES

In this chapter the main implemented features are highlighted. Next to the list below, there are boiler-plate features e.g. to adjust gain, write a wav-file, detect silence, following envelope, playback audio. Figure 3 shows a visualization of several features computed with TarsosDSP.

- TarsosDSP was originally conceived as a library for pitch estimation, therefore it contains several pitch estimators: YIN [12], MPM [20], AMDF[24][4], and an estimator based on dynamic wavelets[15]. There are two YIN implementations, one remains within the comforts of the time domain, the other calculates convolution in the frequency domain[5].

- Two onset detectors are provided. One described in [4], and the one used by the BeatRoot system[13].

- The WSOLA time stretch algorithm[31], which allows to alter the speed of an audio stream without altering the pitch is included. On moderate time stretch factors - 80%-120% of the original speed - only limited audible artifacts are noticeable.

- A resampling algorithm based on [27] and the related open source resample software package[6].

[3]http://www.surina.net/soundtouch/ SoundTouch, by Olli Parviainen, is an open-source audio processing library.

[4]Partial implementation provided by Eder Souza

[5]The YIN FFT implementation was kindly contributed by Matthias Mauch

[6]Resample 1.8.1 can be found on the digital audio resampling home page https://ccrma.stanford.edu/ jos/resample/, maintained by Julius O. Smith

- A pitch shifting algorithm, which allows to change the pitch of audio without affecting speed, is formed by chaining the time-stretch algorithm with the resample algorithm.
- As examples of audio effects, TarsosDSP contains a delay and flanger effect. Both are implemented as minimalistic as possible.
- Several IIR-filters are included. A single pass and four stage low pass filter, a high pass filter, and a band pass filter.
- TarsosDSP also allows audio synthesis and includes generators for sine waves and noise. Also included is a Low Frequency Oscillator (LFO) to control the amplitude of the resulting audio.
- A spectrum can be calculated with the inevitable FFT or using the provided implementation of the Constant-Q[9] transform.

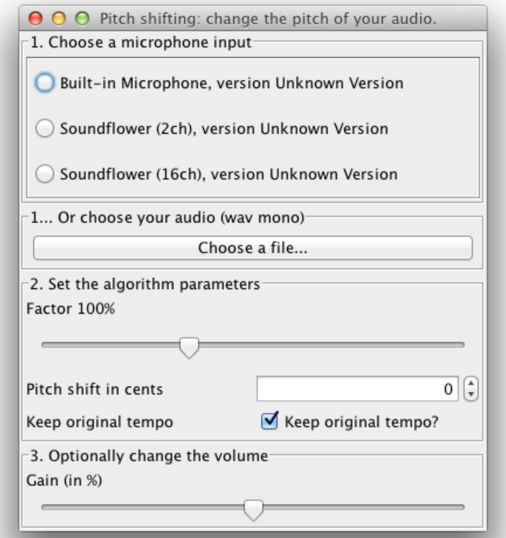
3.1. Example Applications

To show the capabilities of the framework, seventeen examples are built. Most examples are small programmes with a simple user interface, showcasing one algorithm. They don't only show which functionality is present in the framework, but also how to use those in other applications. There are example applications for time stretching, pitch shifting, pitch estimation, onset detection,. . . Figure 4 shows an example application, featuring the pitch shifting algorithm.

TarsosDSP is used by Tarsos[26] a software tool to analyze and experiment with pitch organization in non-western music. It is an end-user application with a graphical user interface that leverages a lot of TarsosDSP's features. It can be seen as a showcase for the framework.

4. AVAILABILITY AND LICENSE

The source code is available under the GPL license terms at GitHub: https://github.com/JorenSix/TarsosDSP. Contributions are more than welcome. TarsosDSP releases, the manual, and documentation can be found at the release directory http://0110.be/releases/TarsosDSP/. Nightly builds can be found there as well. Other downloads, documentation on the example applications and background information is available on http://0110.be. Providing the source code under the GPL license makes sure that derivative works also need to provide the source code, which enables reproducibility.



1. Choose a microphone input
[✓ Built-in Microphone, version Unknown Version]
Soundflower (2ch), version Unknown Version
Soundflower (16ch), version Unknown Version

1... Or choose your audio (wav mono)

2. Set the algorithm parameters
Factor 100%
Pitch shift in cents: 0
Keep original tempo: [✓ Keep original tempo?]

3. Optionally change the volume
Gain (in %)

Fig. 4: A TarsosDSP example application. Most algorithms implemented in TarsosDSP have a demo application with a user interface. Here, the capabilities of a pitch shifting algorithm are shown.

5. CONCLUSION

In this paper TarsosDSP was presented. An Open Source Java library for real time audio processing without external dependencies. It allows real-time pitch and onset extraction, a unique feature in the Java ecosystem. It also contains algorithms for time stretching, pitch shifting, filtering, resampling, effects, and synthesis. TarsosDSP serves an educational goal, therefore algorithms are implemented as simple and self-contained as possible using a straightforward pipeline. The library can be used on the Android platform, as a back-end for Java applications or stand alone, by using one of the provided example applications. After two years of active development it has become a valuable addition to the MIR-community.

6. REFERENCES

[1] Real-Time, Low Latency Audio Processing in Java. In *Proceedings of the International Computer Music Conference (ICMC 2007)*, pages 99–102, 2007.

[2] X. Amatriain. *An Object-Oriented Metamodel for Digital Signal Processing with a focus on Audio and Music*. PhD thesis, 2005.

[3] X. Amatriain, P. Arumí, and M. Ramírez. CLAM, Yet Another Library for Audio and Music Processing? In *Proceedings of 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2002)*, 2002.

[4] Dan Barry, Derry Fitzgerald, Eugene Coyle, and Bob Lawlor. Drum Source Separation using Percussive Feature Detection and Spectral Modulation. In *Proceedings of the Irish Signals and Systems Conference (ISSC 2005)*, 2005.

[5] Santiago David Davila Benavides. Racioco de Agentes Musicais Composi Algorica, Vida artificiale Interatividade em Sistemas Multiagentes Musicais. Master's thesis, Instituto de Matemca e Estatica - Universidade de Saulo, 2012.

[6] D. Bogdanov, Nicolas Wack, Emilia Gómez, Sankalp Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, and Xavier Serra. ESSENTIA: an Audio Analysis Library for Music Information Retrieval. In *Proceedings of the 14th International Symposium on Music Information Retrieval (ISMIR 2013)*, pages 493–498, Curitiba, Brazil, 04/11/2013 2013.

[7] Paul Brossier. *Automatic Annotation of Musical Audio for Interactive Applications*. PhD thesis, Queen Mary University of London, UK, August 2006.

[8] Andrew R. Brown and Andrew C. Sorensen. Introducing jMusic. In Andrew R. Brown and Richard Wilding, editors, *Australasian Computer Music Conference*, pages 68–76, Queensland University of Technology, Brisbane, 2000. ACMA.

[9] Judith Brown and Miller S. Puckette. An efficient algorithm for the calculation of a constant q transform. *Journal of the Acoustical Society of America*, 92(5):2698–2701, November 1992.

[10] P. Burk. JSyn - A Real-time Synthesis API for Java. In *Proceedings of the 1998 International Computer Music Conference (ICMC 1998)*. Computer Music Associaciation, 1998.

[11] C Cannam, C Landone, M Sandler, and J.P Bello. The Sonic Visualiser: A Visualisation Platform for Semantic Descriptors from Musical Signals. In *Proceedings of the 7th International Symposium on Music Information Retrieval (ISMIR 2006)*, Victoria, Canada, 2006.

[12] Alain de Cheveigné and Kawahara Hideki. YIN, a Fundamental Frequency Estimator for Speech and Music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930, 2002.

[13] Simon Dixon. Automatic Extraction of Tempo and Beat From Expressive Performances. *Journal of New Music Research (JNMR)*, 30(1):39–58, 2001.

[14] Emilia G. Teaching MIR: Educational Resources Related To Music Information Retrieval. In *Proceedings of the 13th International Symposium on Music Information Retrieval (ISMIR 2012)*. International Society for Music Information Retrieval, 2012.

[15] Eric Larson and Ross Maddox. Real-Time Time-Domain Pitch Tracking Using Wavelets. 2005.

[16] Victor Lazzarini. Sound Processing with the SndObj Library: An Overview. In *Proceedings of the 4th International Conference on Digital Audio Effects (DAFX 2001)*, pages 6–8, 2001.

[17] Benoathieu, Slim Essid, Thomas Fillon, Jacques Prado, and Gaichard. YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software. In *Proceedings of the 11th International Symposium on Music Information Retrieval (ISMIR 2010)*, pages 441–446. International Society for Music Information Retrieval, 2010.

[18] D. McEnnis, C. McKay, and I. Fujinaga. jAudio: A Feature Extraction Library. In *Proceedings of the 6th International Symposium on Music Information Retrieval (ISMIR 2005)*, 2005.

[19] Philip McLeod. *Fast, Accurate Pitch Detection Tools for Music Analysis*. PhD thesis, University of Otago. Department of Computer Science, 2009.

[20] Phillip McLeod and Geoff Wyvill. A Smarter Way to Find Pitch. In *Proceedings of the International Computer Music Conference (ICMC 2005)*, 2005.

[21] E.X. Merz. *Sonifying Processing: The Beads Tutorial*. CreateSpace, 2011.

[22] John Anderson Mills III, Damien Di Fede, and Nicolas Brix. Music programming in minim. In *Proceedings of the New Interfaces for Musical Expression++ Conference (NIME++)*, Sydney, Australia, 2010.

[23] Stephen Travis Pope and Chandrasekhar Ramakrishnan. The Create Signal Library (Sizzle): Design, Issues and Applications. In *Proceedings of the 2003 International Computer Music Conference (ICMC 2003)*, 2003.

[24] M. J. Ross, H. L. Shaffer, A. Cohen, R. Freudberg, and H. J. Manley. Average Magnitude Difference Function Pitch Extractor. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 22(5):353–362, October 1974.

[25] Gary P. Scavone and Perry R. Cook. RTMidi, RTAudio, and a Synthesis Toolkit (STK) UPdate. In *Proceedings of the International Computer Music Conference (ICMC 2005)*, 2005.

[26] Joren Six, Olmo Cornelis, and Marc Leman. Tarsos, a Modular Platform for Precise Pitch Analysis of Western and Non-Western Music. *Journal of New Music Research*, 42(2):113–129, 2013.

[27] Julius O. Smith and Phil Gosset. A Flexible Sampling-Rate Conversion Method. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP 1984)*, volume 2, 1984.

[28] Thomas Stubbe. Geautomatiseerde vorm- en structuuranalyse van muzikale audio. Master's thesis, Universiteit Gent, 2013.

[29] George Tzanetakis and Perry Cook. MARSYAS: a Framework for Audio Analysis. *Organized Sound*, 4(3):169–175, December 1999.

[30] K. van den Doel and D. K. Pai. JASS: A Java Audio Synthesis System for Programmers. In J. Hiipakka, N. Zacharov, and T. Takala, editors, *Proceedings of the 7th International Conference on Auditory Display (ICAD 2001)*, pages 150–154, 2001.

[31] Werner Verhelst and Marc Roelands. An Overlap-Add Technique Based on Waveform Similarity (WSOLA) for High Quality Time-Scale Modification of Speech. In *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP 1993)*, pages 554–557, 1993.

[32] Michael Wager. Entwicklung eines Systems zur automatischen Notentranskription von monophonischem Audiomaterial. Master's thesis, Hochschule Ausburg, 2011.