LIBXTRACT: A LIGHTWEIGHT LIBRARY FOR AUDIO FEATURE EXTRACTION

Jamie Bullock
UCE Birmingham Conservatoire
Music Technology

ABSTRACT

The libxtract library consists of a collection of over forty functions that can be used for the extraction of low level audio features. In this paper I will describe the development and usage of the library as well as the rationale for its design. Its use in the composition and performance of music involving live electronics also will be discussed. A number of use case scenarios will be presented, including the use of individual features and the use of the library to create a 'feature vector', which may be used in conjunction with a classification algorithm, to extract higher level features.

1. INTRODUCTION

1.1. Audio features

An audio feature is any qualitatively or quantitatively measurable aspect of a sound. If we describe a sound as 'quite loud', we have used our ears to perform an audio feature extraction process. Musical audio features include melodic shape, rhythm, texture and timbre. However, most software-based approaches tend to focus on numerically quantifiable features. For example, the MPEG-7 standard defines seventeen low-level descriptors (LLDs) that can be divided into six categories: 'basic', 'basic spectral', 'signal parameters', 'temporal timbral', 'spectral timbral' and 'spectral basis representations'[1]. The Cuidado project extends the MPEG-7 standard by providing 72 audio features, which it uses for content-based indexing and retrieval[2] .

1.2. libxtract

libxtract is a cross-platform, free and open-source software library that provides a set of feature extraction functions for extracting LLDs. The eventual aim is to provide a superset of the MPEG-7 and Cuidado audio features. The library is written in ANSI C and licensed under the GNU GPL so that it can easily be incorporated into any program that supports linkage to shared libraries.

2. EXISTING TECHNOLOGY

It is beyond the scope of this paper to conduct an exhaustive study of existing technology and compare the results with libxtract. However, a brief survey of the library's context will be given.

One project related to libxtract is the Aubio library [1] by Paul Brossier. Aubio is designed for audio labelling, and includes excellent pitch, beat and onset detectors[5]. libxtract doesn't currently include any onset detection, this makes Aubio complimentary to libxtract with minimal duplication of functionality.

libxtract has much in common with the jAudio project [9], which seeks to provide a system that 'meets the needs of MIR researchers' by providing 'a new framework for feature extraction designed to eliminate the duplication of effort in calculating features from an audio signal'. jAudio provides many useful functions such as the automatic resolution of dependencies between features, an API which makes it easy to add new features, multidimensional feature support, and XML file output. Its implementation in Java makes it cross-platform, and suitable for embedding in other Java applications, such as the promising jMIR software. However, libxtract was written out of a need to perform real-time feature extraction on live instrumental sources, and jAudio is not designed for this task. libxtract, being written in C, also has the advantage that it can be incorporated into programs written in a variety of languages, not just Java. Examples of this are given in section 5.

libxtract also provides functionality that is similar to aspects of the CLAM project [2] . According to Amatriain, CLAM is 'a framework that aims at offering extensible, generic and efficient design and implementation solutions for developing Audio and Music applications as well as for doing more complex research related with the field'[8]. As noted by McKay, 'the [CLAM] system was not intended for extracting features for classification problems', it is a large and complex piece of software with many dependencies, making it a poor choice if only feature extraction functionality is required.

Other similar projects include Marsyas [3] and Maate [4] . The library components of these programs all include feature extraction functionality, but they all provide other functions for tasks such as file and audio i/o, annotation or session handling. libxtract provides only feature extraction functions on the basis that any additional functionality can

[1] `http://aubio.piem.org`
[2] `http://clam.iua.upf.edu`
[3] `http://marsyas.sf.net`
[4] `http://www.cmis.csiro.au/maaate`

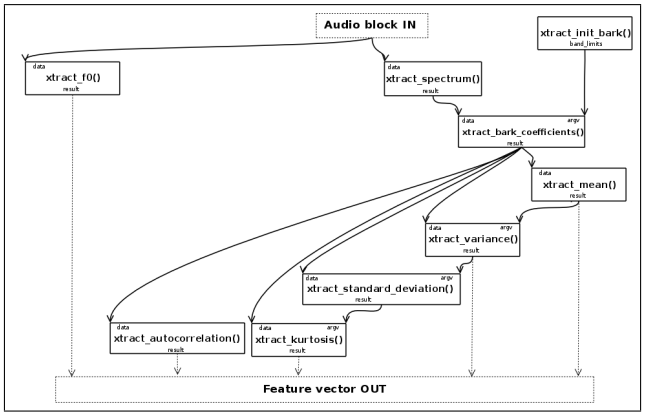

Figure 1. A typical libxtract feature cascade

be provided by the calling application or another library.

3. LIBRARY DESIGN

The central idea behind libxtract is that the feature extraction functions should be modularised so they can be combined arbitrarily. Central to this approach is the idea of a cascaded extraction hierarchy. A simple example of this is shown in Figure 1. This approach serves a dual purpose: it avoids the duplication of 'subfeatures', making computation more efficient, and if the calling application allows, it enables a certain degree of experimentation. For example the user can easily create novel features by making unconventional extraction hierarchies.

libxtract seeks to provide a simple API for developers. This is achieved by using an array of function pointers as the primary means of calling extraction functions. A consequence of this is that all feature extraction functions have the same prototype for their arguments. The array of function pointers can be indexed using an enumeration of descriptively-named constants. A typical libxtract call in the DSP loop of an application will look like this:

```
xtract[XTRACT_FUNCTION_NAME]
(input_vector, blocksize, argv,
 output_vector);
```

This design makes libxtract particularly suitable for use in modular patching environments such as Pure Data and Max/MSP, because it alleviates the need for the program making use of libxtract to provide mappings between symbolic 'xtractor' names and callback function names.

libxtract divides features into scalar features, which give the result as a single value, vector features, which give the result as an array, and delta features, which have some temporal element in their calculation process. To make the process of incorporating the wide variety of features (with their slightly varying argument requirements) easier, each extraction function has its own function descriptor. The purpose of the function descriptor is to provide useful 'self documentation' about the feature extraction function in question. This enables a calling application to easily determine the expected format of the data pointed to by `*data` and `*argv`, and what the 'donor' functions for any sub-features (passed in as part of the argument vector) would be.

Feature name	Description
Mean	Mean of a vector
Kurtosis	Kurtosis of a vector
Spectral Mean	Mean of a spectrum
Spectral Kurtosis	Kurtosis of a spectrum
Spectral Centroid	Centroid of a spectrum
Irregularity (2 types)	Irregularity of a spectrum
Tristimulus (3 types)	Tristimulus of a spectrum
Smoothness	Smoothness of a spectrum
Spread	Spread of a spectrum
Zero Crossing Rate	Zero crossing rate of a vector
Loudness	Loudness of a signal
Inharmonicity	Inharmonicity of a spectrum
F0	Fundamental frequency of a signal
Autocorrelation	Autocorrelation vector of a signal
Bark Coefficients	Bark coefficients from a spectrum
Peak Spectrum	Spectral peaks from a spectrum
MFCC	MFCC from a spectrum

Table 1. Some of the features provided by the library

The library has been written on the assumption that a contiguous block of data will be written to the input array of the feature extraction functions. Some functions assume the data represents a block of time-domain audio data, others use a special spectral data format, and others make no assumption about what the data represents. Some of the functions may therefore be suitable for analysing non-audio data.

Certain feature extraction functions require that the FFT of an audio block be taken. The inclusion of FFT processing is provided as a compile time option because it entails a dependency on the FFTW library. Signal windowing and zero padding are provided as helper functions.

4. LIST OF FEATURES

It is beyond the scope of this paper to list all the features provided by the library, but some of the most useful ones are listed in table 1. If the feature is 'of a spectrum' it denotes that the input data will follow the format of libxtract's spectral data types.

5. PROGRAMS USING THE LIBRARY

Despite the fact that libxtract is a relatively recent library, it has already been incorporated into a number of useful programs.

5.1. Vamp libxtract plugin

The Vamp analysis plugin API was devised by Chris Cannam for the Sonic Visualiser [5] software. Sonic Visualiser is an application for visualising features extracted from audio files, it was developed at Queen Mary University of London, and has applications in musicology, signal-processing research and performance practice[4]. Sonic Visualiser acts as a Vamp plugin host, with vamp plugins supplying analysis data to it.

libxtract is used to provide analysis functions for Sonic Visualiser using the vamp-libxtract-plugin. The vamp-libxtract-plugin acts as a wrapper for the libxtract library, making nearly the entire set of libxtract features available to any vamp host. This is done by providing only the minimum set of feature combinations, the implication of this being that the facility to experiment with different cascaded features is lost.

5.2. PD and Max/MSP externals

The libxtract library comes with a Pure Data (PD) external, which acts as a wrapper to the library's API. This is an ideal use case because it enables feature extraction functions to be cascaded arbitrarily, and for non-validated data to be passed in as arguments through the [xtract~] object's right inlet. The disadvantage of this approach is that it requires the user to learn how the library works, and to understand in a limited way what each function does.

A Max/MSP external is available, which provides functionality that is analogous to the PD external.

5.3. SC3 ugen

There also exists a Supercollider libxtract wrapper by Dan Stowell. This is implemented as a number of SuperCollider UGens, which are object-oriented multiply instantiable DSP units [6]. The primary focus of Stowell's libxtract work is a libxtract-based MFCC UGen, but several other libxtract-based Ugens are under development.

6. EXTRACTING HIGHER LEVEL FEATURES

The primary focus of the libxtract library is the extraction of relatively low-level features. However, one the main reasons for its development was that it could serve as a basis for extracting more semantically meaningful features. These could include psychoacoustic features such as roughness, sharpness and loudness[10], some of which are included in the library; instrument classification outputs[3]; or arbitrary user-defined descriptors such as 'Danceability'[7].

It is possible to extract these 'high level' features by using libxtract to extract a feature vector, which can be constructed using the results from a range of extraction functions. This vector can then be submitted to a mapping that entails a further reduction in dimensionality of the data. Possible algorithms for the dimension reduction task include Neural Networks, k-NN and Multidimensional Gauss[11]. Figure 2 shows a dimension reduction implementation in Pure Data using the [xtract~] libxtract wrapper, and the [ann_mlp] Fast Artificial Neural Network [7] (FANN) library wrapper by Davide Morelli [8].

An extended version of this system has recently been used in one of the author's own compositions for Piano and Live electronics. For this particular piece, a PD patch was created to detect whether a specific chord was being played, and to add a 'resonance' effect to the Piano accordingly. For the detection aspect of the patch, a selection of audio features, represented as floating point values, are 'packed' into a list using the PD [pack] object. This data is used to train the neural network (a multi-layer perceptron), by successively presenting it with input lists, followed by the corresponding expected output. Once the network has been trained (giving the minimum possible error), it can operate in 'run' mode, whereby it should give appropriate output when presented with new data that shows similarity to the training data. With a close-mic'd studio recording in a dry acoustic, an average detection accuracy of 92% was achieved. This dropped to around 70% in a concert environment. An exploration of these results is beyond the scope of this paper.

Another possible use case for the library is as a source for continuous mapping to an output feature space. With a continuous mapping, the classifier gives as output, a location on a low-dimensional map rather than giving a discrete classification 'decision'. This has been implemented in one of the author's recent works for Flute and live electronics, whereby the flautist can control the classifier's output by modifying their instrumental timbre. Semantic descriptors were used to tag specific map locations, and proximity to these locations was measured and used as a source of realtime control data. The system was used to measure the 'breathiness' and 'shrillness' of the flute timbre. Further work could involve the recognition of timbral gestures in this resultant data stream.

7. EFFICIENCY AND REALTIME USE

The library in its current form makes no guarantee about how long it will take for a function to execute. For any given blocksize, there is no defined behaviour determining what will happen if the function does not return in the duration of the block.

Most of the extraction functions have an algorithmic efficiency of O(n) or better, meaning that computation time is usually proportional to the audio blocksize used. However, because of the way in which the library has been designed (flexibility of feature combination has been given priority), certain features end up being computed comparatively inefficiently. For example if only the Kurtosis feature was required in the system shown in figure 1, the functions xtract_mean(), xtract_variance(),

[5] http://www.sonicvisualiser.org
[6] http://mcld.co.uk/supercollider
[7] http://leenissen.dk/fann/
[8] http://www.davidemorelli.it

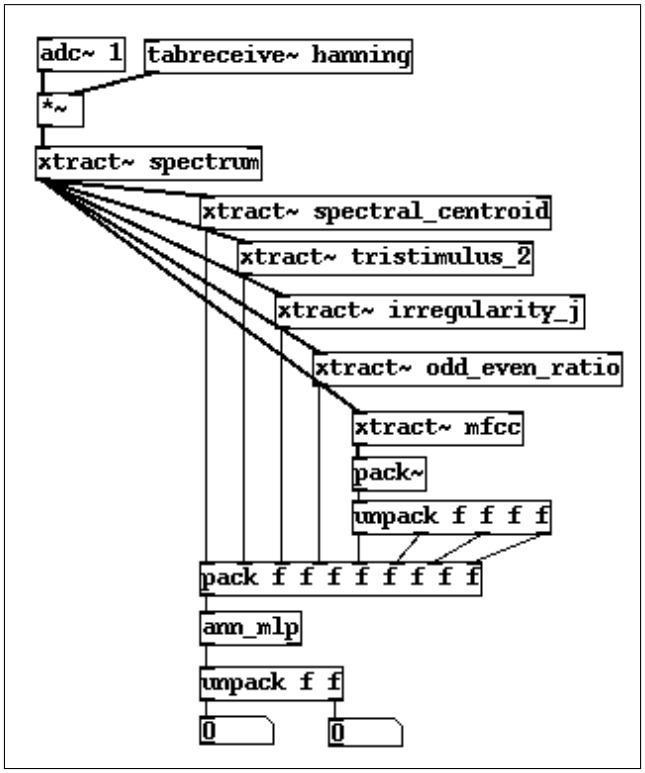

Figure 2. Audio feature vector construction and dimension reduction using libxtract and FANN bindings in Pure Data

xtract_standard_deviation() and xtract_kurtosis() must all execute N iterations over their input (where N is the size of the input array). The efficiency of xtract_kurtosis() could be improved if the outputs from all the intermediate features were not exposed to the user or developer.

Tests show that all the features shown in table 1 can be computed simultaneously with a blocksize of 512 samples, and 20 Mel filter bands with a load of 20-22% on a dual Intel 2.1GHz Macbook Pro Laptop running GNU/Linux with a 2.6 series kernel. This increases to 70% for a block size of 8192, but removing xtract_f0() reduces this figure to 50%.

8. CONCLUSIONS

In this paper I have described a new library that can be used for low level audio feature extraction. It is capable of being used inside a realtime application, and serves as a useful tool for experimentation with audio features. Use of the library inside a variety of applications has been discussed, along with a description of its role in extracting higher level, more abstract features. It can be concluded that the library is a versatile tool for low-level feature extraction, with a simple and convenient API.

9. REFERENCES

[1] Lindsay, T., Burnett, I., Quackenbush, S., Jackson, M. "Fundamentals of Audio Descriptors", *Introduction to MPEG-7 Multimedia Content Description Interface*, West Sussex, England, 2003.

[2] Peeters, G. *A large set of audio features for sound description (similarity and classification) in the CUIDADO project,*. IRCAM, Paris, 2003.

[3] Fujinaga, I., and MacMillan, K. "Realtime recognition of orchestral instruments." *Proceedings of the Interface Computer Music Conference* 2000.

[4] Cannam, C., Landone, C., Sandler, M., and Bello, J., P. "The Sonic Visualiser: A Visualisaton Platform for Semantic Descriptors from Musical Signals." *Proceedings of the 7th International Conference on Music Information Retrieval* Victoria, Canada, 2006.

[5] Brossier, P., M. "Automatic Annotation of Musical Audio for Interactive Applications" *PhD Thesis* Centre for Digital Music, Queen Mary, University of London, UK, 2006.

[6] Lerch, A. "FEAPI: A Low Level Feature Extraction Plugin API" *Proceedings of the 8th International Conference on Digital Audio Effects (DAFx)* Madrid, Spain, 2005

[7] Amatriain, X., Massaguer, J., Garcia, D., and Mosguera, I. "Features for Audio and Music Classification" *Proceedings of the 6th International Conference on Music Information Retrieval* London, UK, 2005

[8] Amatriain, X. "An Object-Oriented Metamodel for Digital Signal Processing with a focus on Audio and Music" *PhD Thesis* Music Technology Group of the Institut Universitari de l'Audiovisual at the Universitat Pompeu Fabra, Barcelona, Spain, 2004

[9] McEnnis, D., McKay, C., Fujinaga, I., Depalle, P. "jAudio: A Feature Extraction Library" *Proceedings of the 6th International Conference on Music Information Retrieval* London, UK, 2005

[10] Moore, B. C. J., Glasberg, B. R., and Baer, T. "A model for the prediction of thresholds, loudness and partial loudness" *J. Audio Eng. Soc., vol. 45, pp. 224-240* New York, USA, 1997

[11] Herrera-Boyer, P., Peeters, G., Dubnov, S. "Automatic Classification of Musical Instrument Sounds" *Journal of New Music Research, Volume 32, Issue 1, pages 3 - 21* London, UK, 2003