A MATLAB TOOLBOX FOR MUSICAL FEATURE EXTRACTION FROM AUDIO

Olivier Lartillot, Petri Toiviainen

University of Jyväskylä
Finland
lartillo@campus.jyu.fi

ABSTRACT

We present *MIRtoolbox*, an integrated set of functions written in Matlab, dedicated to the extraction of musical features from audio files. The design is based on a modular framework: the different algorithms are decomposed into stages, formalized using a minimal set of elementary mechanisms, and integrating different variants proposed by alternative approaches – including new strategies we have developed –, that users can select and parametrize.

This paper offers an overview of the set of features, related, among others, to timbre, tonality, rhythm or form, that can be extracted with *MIRtoolbox*. Four particular analyses are provided as examples. The toolbox also includes functions for statistical analysis, segmentation and clustering. Particular attention has been paid to the design of a syntax that offers both simplicity of use and transparent adaptiveness to a multiplicity of possible input types. Each feature extraction method can accept as argument an audio file, or any preliminary result from intermediary stages of the chain of operations. Also the same syntax can be used for analyses of single audio files, batches of files, series of audio segments, multichannel signals, etc. For that purpose, the data and methods of the toolbox are organised in an object-oriented architecture.

1. MOTIVATION AND APPROACH

MIRToolbox is a *Matlab* toolbox dedicated to the extraction of musically-related features from audio recordings. It has been designed in particular with the objective of enabling the computation of a large range of features from databases of audio files, that can be applied to statistical analyses.

Few softwares have been proposed in this area. The most important one, Marsyas [1], provides a general architecture for connecting audio, soundfiles, signal processing blocks and machine learning (see section 5 for more details). One particularity of our own approach relies in the use of the *Matlab* computing environment, which offers good visualisation capabilities and gives access to a large variety of other toolboxes. In particular, the *MIRToolbox* makes use of functions available in recommended public-domain toolboxes such as the *Auditory Toolbox* [2], *NetLab* [3], or *SOM-toolbox* [4]. Other toolboxes, such as the *Statistics toolbox* or the *Neural Network toolbox* from MathWorks, can be directly used for further analyses of the features extracted by *MIRToolbox* without having to export the data from one software to another.

Such computational framework, because of its general objectives, could be useful to the research community in Music Information Retrieval (MIR), but also for educational purposes. For that reason, particular attention has been paid concerning the ease of use of the toolbox. In particular, complex analytic processes can be designed using a very simple syntax, whose expressive power comes from the use of an object-oriented paradigm.

The different musical features extracted from the audio files are highly interdependent: in particular, as can be seen in figure 1, some features are based on the same initial computations. In order to improve the computational efficiency, it is important to avoid redundant computations of these common components. Each of these intermediary components, and the final musical features, are therefore considered as *building blocks* that can been freely articulated one with each other. Besides, in keeping with the objective of optimal ease of use of the toolbox, each building block has been conceived in a way that it can adapt to the type of input data. For instance, the computation of the MFCCs can be based on the waveform of the initial audio signal, or on the intermediary representations such as spectrum, or mel-scale spectrum (see Fig. 1). Similarly, autocorrelation is computed for different range of delays depending on the type of input data (audio waveform, envelope, spectrum). This decomposition of all the set of feature extraction algorithms into a common set of building blocks has the advantage of offering a synthetic overview of the different approaches studied in this domain of research.

2. FEATURE EXTRACTION

2.1. Feature overview

Figure 1 shows an overview of the main features implemented in the toolbox. All the different processes start from the audio signal (on the left) and form a chain of operations proceeding to right. The vertical disposition of the processes indicates an increasing order of complexity of the operations, from simplest computation (top) to more detailed auditory modelling (bottom).

Each musical feature is related to one of the musical dimensions traditionally defined in music theory. Boldface characters highlight features related to pitch, to tonality (chromagram, key strength and key Self-Organising Map, or SOM) and to dynamics (Root Mean Square, or RMS, energy). Bold italics indicate features related to rhythm, namely tempo, pulse clarity and fluctuation. Simple italics highlight a large set of features that can be associated to timbre. Among them, all the operators in grey italics can be in fact applied to many others different representations: for instance, statistical moments such as centroid, kurtosis, etc., can be applied to either spectra, envelopes, but also to histograms based on any given feature.

One of the simplest features, zero-crossing rate, is based on a simple description of the audio waveform itself: it counts the number of sign changes of the waveform. Signal energy is computed using root mean square, or RMS [5]. The envelope of the audio signal offers timbral characteristics of isolated sonic event.

FFT-based spectrum can be computed along the frequency domain or along Mel-bands, with linear or decibel energy scale, and

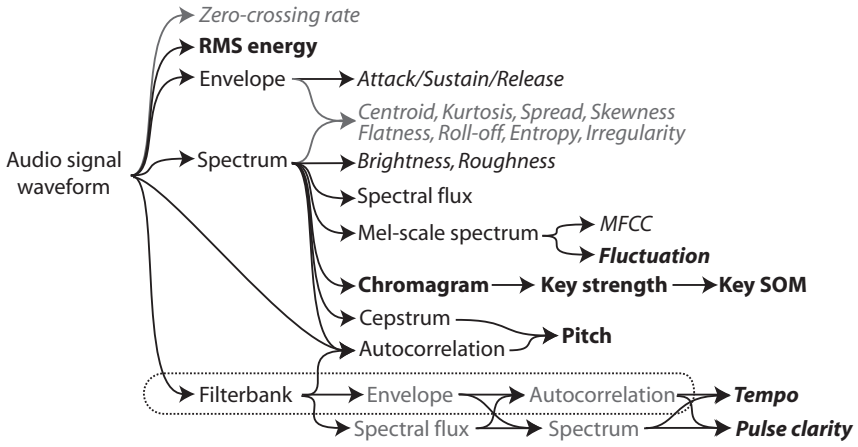

Figure 1: *Overview of the musical features that can be extracted with* MIRToolbox.

applying various windowing methods. The results can be multiplied with diverse resonance curves in order to highlight different aspects such as metrical pulsation (when computing the FFT of envelopes) or fluctuation [6].

Many features can be derived from the FFT:

- Basic statistics of the spectrum gives some timbral characteristics (such as spectral centroid, roll-off [5], brightness, flatness, etc.).
- The temporal derivative of spectrum gives the spectral flux.
- An estimation of roughness, or sensory dissonance, can be assessed by adding the beating provoked by each couple of energy peaks in the spectrum [7].
- A conversion of the spectrum in a Mel-scale can lead to the computation of Mel-Frequency Cepstral Coefficients (MFCC) (cf. example 2.2), and of fluctuation [6].
- Tonality can also be estimated (cf. example 2.3).

The computation of the autocorrelation can use diverses normalization strategies, and integrates the improvement proposed by Boersma [8] in order to compensate the side-effects due to the windowing. Resonance curve are also available here. Autocorrelation can be "generalized" through a compression of the spectral representation [9].

The estimation of pitch is usually based on spectrum, autocorrelation, or cepstrum, or a mixture of these strategies [10].

A distinct approach consists of designing a complete chain of processes based on the modelling of auditory perception of sound and music [2] (circled in Figure 1). This approach can be used in particular for the computation of rhythmic pulsation (cf. example 2.4).

2.2. Example: Timbre analysis

One common way of describing timbre is based on MFCCs [11, 2]. Figure 2 shows the diagram of operations. First, the audio sequence is loaded (1), decomposed into successive frames (2), which are then converted into the spectral domain, using the mirspectrum function (3). The spectra are converted from the frequency domain to the Mel-scale domain: the frequencies are rearranged into 40 frequency bands called Mel-bands[1]. The envelope of the Mel-scale spectrum is described with the MFCCs, which are obtained by applying the Discrete Cosine Transform to the Mel-scale spectrum. Usually only a restricted number of them (for instance the 13 first ones) are selected (5).

```
a = miraudio('audiofile.wav')      (1)
f = mirframe(a)                    (2)
s = mirspectrum(f)                 (3)
m = mirspectrum(s,'Mel')           (4)
c = mirmfcc(s,'Rank',1:13)         (5)
```

The computation can be carried in a window sliding through the audio signal (this corresponded to the code line 1), resulting in a series of MFCC vectors, one for each successive frame, that can be represented column-wise in a matrix. Figure 2 shows an example of such matrix. The MFCCs do not convey very intuitive meaning per se, but are generally applied to distance computation between frames, and therefore to segmentation tasks (cf. paragrapn 2.5).

The whole process can be executed in one single line by calling directly the mirmfcc function with the audio input as argument:

```
mirmfcc(f,'Rank',1:13)             (6)
```

2.3. Example: Tonality analysis

The spectrum is converted from the frequency domain to the pitch domain by applying a log-frequency transformation. The distribution of the energy along the pitches is called the *chromagram*. The chromagram is then wrapped, by fusing the pitches belonging to same pitch classes. The wrapped chromagram shows therefore a distribution of the energy with respect to the twelve possible pitch classes [12].

Krumhansl and Schmuckler [13] proposed a method for estimating the tonality of a musical piece (or an extract thereof)

[1] The Mel-scale conversion is available as an option of the mirspectrum function (4). Note how it is possible to recall a function using one of its previous output as input (here, s), in order to perform some additional optional operations.

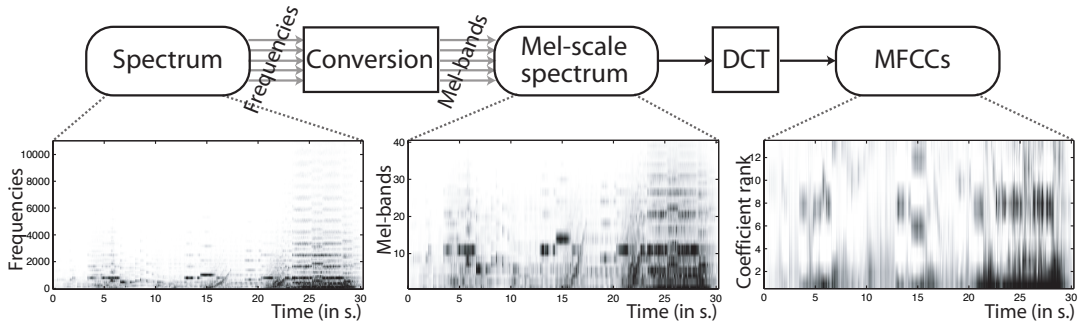

Figure 2: *Successive steps for the computation of MFCCs, illustrated with the analysis of an audio excerpt decomposed into frames.*

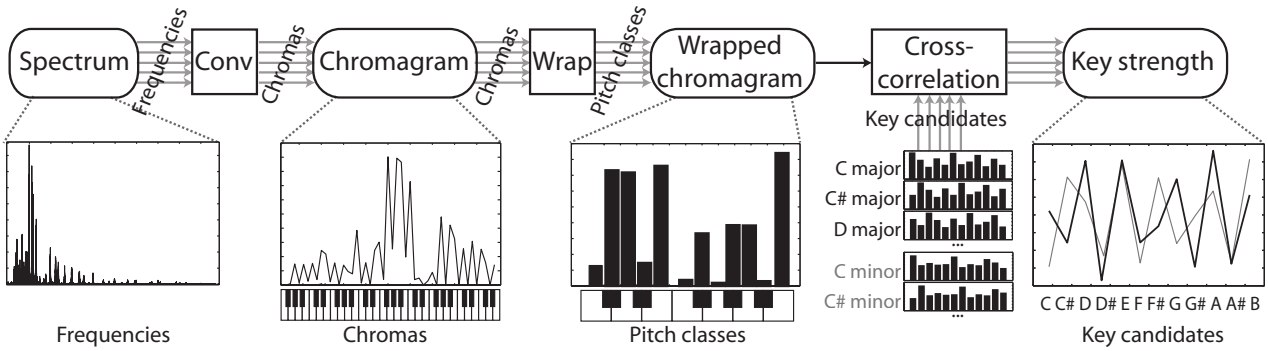

Figure 3: *Successive steps for the calculation of chromagram and estimation of key strengths, illustrated with the analysis of an audio excerpt, this time not decomposed into frames.*

by computing the cross-correlation of its pitch class distribution with the distribution associated to each possible tonality. These distribution have been established though listening experiments [14]. The most prevalent tonality is considered to be the tonality candidate with highest correlation, or *key strength*. This method was originally designed for the analysis of symbolic representations of music but has been extended to audio analysis through an adaptation of the pitch class distribution to the chromagram representation [12]. Figure 3 displays the successive steps of this approach. For instance the following command estimates the three most probable key candidates for each frame.

```
mirkey(f,'Total',3)
```
(7)

A richer representation of the tonality estimation can be drawn with the help of a self-organizing map (SOM), trained by the 24 tonal profiles [15]. The configuration of the trained SOM reveals key relations that correspond to music theoretical notions. The estimation of the tonality of the musical piece under study is carried by projecting its wrapped chromagram onto the SOM. Figure 4 shows the resulting activity pattern in the SOM.

2.4. Example: Rhythm analysis

One common way of estimating the rhythmic pulsation, described in figure 6, is based on auditory modelling [5]. The audio signal is first decomposed into auditory channels using a bank of filters. Diverse types of filterbanks are proposed and the number of channels can be changed, such as 20 for instance (8). The envelope of each

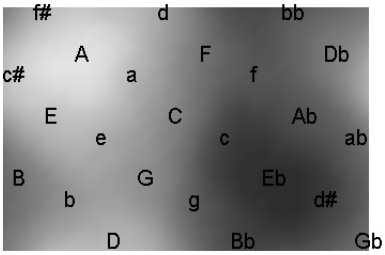

Figure 4: *Activity pattern of a self-organizing map representing the tonal configuration of the first two seconds of Mozart* Sonata in A major, *K 331. High activity is represented by bright nuances.*

channel is extracted (9)[2]. As pulsation is generally related to increase of energy only, the envelopes are differentiated, half-wave rectified, before being finally summed together again (10). This gives a precise description of the variation of energy produced by each note event from the different auditory channels.

After this onset detection, the periodicity is estimated through autocorrelation (12)[3]. However, if the tempo varies throughout the piece, an autocorrelation of the whole sequence will not show clear periodicities. In such cases it is better to compute the autocorrela-

[2]Note how the analysis of multi-channel signal (such as fb) follows exactly the same kind of syntax than for mono-channel signal.

[3]For the sake of clarity, several options in the following functions have been omitted.

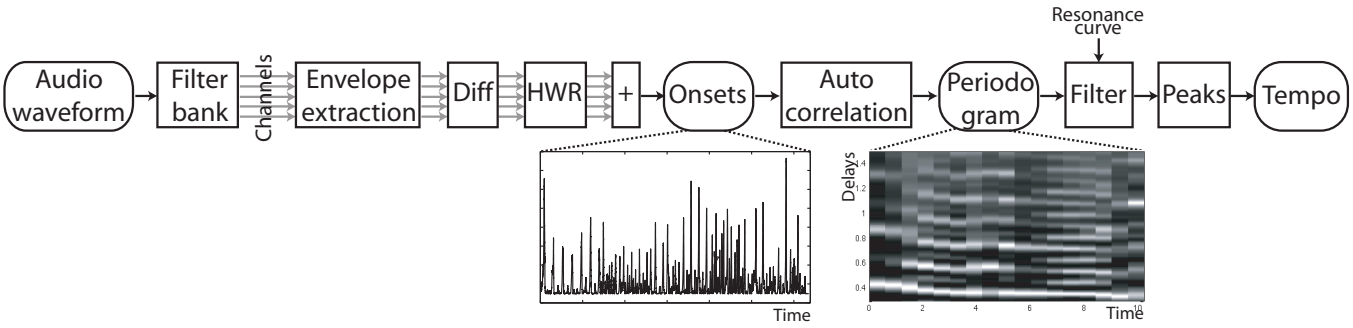

Figure 5: *Successive steps for the estimation of tempo illustrated with the analysis of an audio excerpt. In the periodogram, high autocorrelation values are represented by bright nuances.*

tion for a frame decomposition (11)[4]. This yields a periodogram that highlights the different periodicities, as shown in figure 6. In order to focus on the periodicities that are more perceptible, the periodogram is filtered using a resonance curve [16] (12), after which the best tempos are estimated through peak picking (13), and the results are converted into beat per minutes (14). Due to the difficulty of choosing among the possible multiples of the tempo, several candidates (three for instance) may be selected for each frame, and a histogram of all the candidates for all the frames, called periodicity histogram, can be drawn (15).

```
fb = mirfilterbank(a,20)                    (8)
e = mirenvelope(fb,'Diff','Halfwave')       (9)
s = mirsum(e)                               (10)
fr = mirframe(s,3,.1)                       (11)
ac = mirautocor(fr,'Resonance')             (12)
p = mirpeaks(ac,'Total',1,'NoEnd')          (13)
t = mirtempo(p)                             (14)
h = mirhisto(t)                             (15)
```

The whole process can be executed in one single line by calling directly the mirtempo function with the audio input as argument:

```
mirtempo(a,'Frame')                         (16)
```

In this case, the different options available throughout the process can directly be specified as argument of the tempo function. For instance, a computation of a frame-based tempo estimation, with a selection of the 3 best tempo candidates in each frame, a range of admissible tempi between 60 and 120 beats per minute, an estimation strategy based on a mixture of spectrum and autocorrelation applied on the spectral flux will be executed with the syntax:

```
mirtempo(a,'Frame','Total',3,
         'Min',60,'Max',120,'Spectrum',
         'Autocor','SpectralFlux')          (17)
```

[4]The mirframe function can accept both audio signal and envelope as argument. Here, the frame size is 3 seconds and the hop factor .1.

2.5. Segmentation

More elaborate tools have also been implemented that can carry out higher-level analyses and transformations. In particular, audio files can be automatically segmented into a series of homogeneous sections, through the estimation of temporal discontinuities along diverse alternative features such as timbre in particular [17]. First the audio signal is decomposed into frames (18) and one chosen feature, such as MFCC (19), is computed along these frames. The feature-based distances between all possible frame pairs are stored in a similarity matrix (20). Convolution along the main diagonal of the similarity matrix using a Gaussian checkerboard kernel yields a novelty curve that indicates the temporal locations of significant textural changes (21). Peak detection applied to the novelty curve returns the temporal position of feature discontinuities (22) that can be used for the actual segmentation of the audio sequence (23)[5].

```
fr = mirframe(a)                            (18)
fe = mirmfcc(fr)                            (19)
sm = mirsimatrix(fe)                        (20)
nv = mirnovelty(sm)                         (21)
ps = mirpeaks(nv)                           (22)
sg = mirsegment(a,ps)                       (23)
                                            (24)
```

The whole segmentation process can be executed in one single line by calling directly the mirsegment function with the audio input as argument:

```
mirsegment(a,'Novelty')                     (25)
```

By default, the novelty curve is based on MFCC, but other features can be selected as well using an additional option:

```
mirsegment(a,'Novelty','Spectrum')          (26)
```

A second similarity matrix can be computed, in order to show the distance – according to the same feature than the one used for

[5]The first argument of the mirsegment function is the audio file that needs to be segmented. It is possible for instance to compute the novelty curve using a downsampled version of `a` (18) and to perform the actual segmentation using the original audio file.

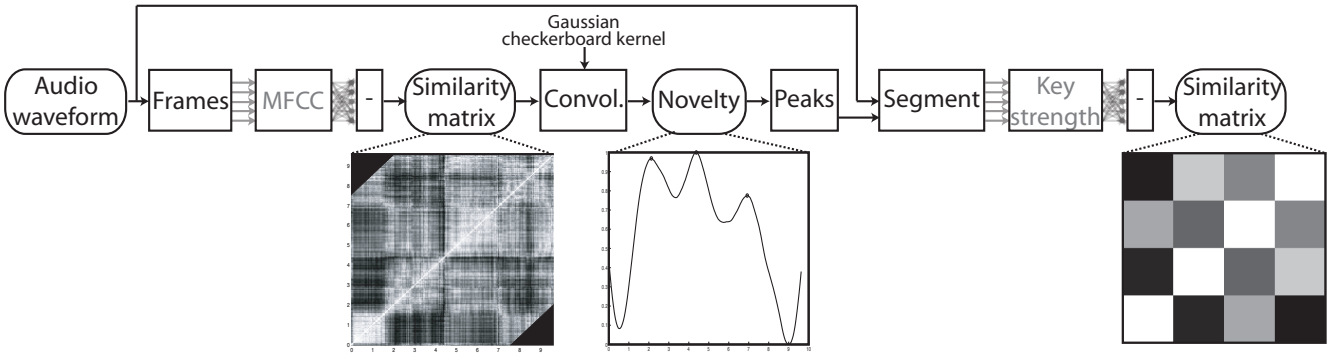

Figure 6: *Successive steps for the segmentation of an audio sequence based on timbral novelty. In the similarity matrix, high similarity values are represented by bright nuances.*

the segmentation – between all possible segment pairs (28).[6]

```
fesg = mirmfcc(sg)                (27)
smsg = mirsimatrix(fesg)          (28)
```

2.6. Data analysis

The toolbox includes diverse tools for data analysis, such as a peak extractor, and functions that compute histograms, entropy, zero-crossing rates, irregularity or various statistical moments (centroid, spread, skewness, kurtosis, flatness) on data of various types, such as spectrum, envelope or histogram.

The `mirpeaks` functions can accept any data returned by any other function of the MIRtoolbox and can adapt to the different kind of data of any number of dimensions. In the graphical representation of the results, the peaks are automatically located on the corresponding curves (for 1D data) or bit-map images (for 2D data).

The `mirpeaks` functions offers alternative possible heuristics. It is possible to define a global threshold that peaks must exceed for them to be selected. We have designed a new strategy of peak selection, based on a notion of *contrast*, discarding peaks that are not sufficiently contrastive (based on a certain threshold) with the neighbouring peaks. This adaptive filtering strategy hence adapts to the local particularities of the curves. Its articulation with other more conventional thresholding strategies leads to an efficient peak picking module that can be applied throughout the *MIRtoolbox*.

Supervised classification of musical samples can also be performed, using techniques such as K-Nearest Neighbours or Gaussian Mixture Model. One possible application is the classification of audio recordings into musical genres.

3. DESIGN OF THE TOOLBOX

3.1. Data encapsulation

All the data returned by the functions in the toolbox are encapsulated into types objects. The default display method associated to all these objects is a graphical display of the corresponding curves.

[6]Note how the computation of a feature along the successive segments of an audio sequence (27) follows exactly the same kind of syntax that for the computation of a feature along successive frames (19).

In this way, when the display of the values of a given analysis is requested, what is printed is not a listing of long vectors or matrices, but rather a correctly formatted graphical representation.

The actual data matrices associated to those data can be obtained by calling a method called `mirgetdata`, which constructs the simplest possible data structure associated to the data (cf. paragraph 4.1).

3.2. Frame analysis

Frame-based analyses (i.e., based on the use of a sliding window) can be specified using two alternative methods. The first method is based on the use of the `mirframe` function, which decomposes an audio signal into successive frames. Optional arguments can specify the frame size (in seconds, by default), and the hop factor (between 0 and 1, by default). For instance, in the following code (line 29), the frames have a size of 50 milliseconds and are half overlapped. The results of that function could then be directly sent as input of any other function of the toolbox (30):

```
f = mirframe(a,.05,.5)            (29)
mirtempo(f)                       (30)
```

Yet this first method does not work correctly for instance when dealing with tempo estimation as described in section 2.4. Following this first method, as shown in figure 7, the frame decomposition is the first step performed in the chain of processes. As a result, the input of the filterbank decomposition is a series of short frames, which induces two main difficulties. Firstly, in order to avoid the presence of undesirable transitory state at the beginning of each filtered frame, the initial state of each filter would need to be tuned depending on the state of the filter at one particular instant of the previous frame (depending of the overlapping factor). Secondly, the demultiplication of the redundancies of the frame decomposition (if the frames are overlapped) throughout the multiple channels of the filterbank would require the use of consequent memory space. The technical difficulties and waste of memory induced by this first method can be immediately overcome if the frame decomposition is performed after the filterbank decomposition and recomposition, as shown in figure 8.

This second method, more successful in this context, cannot be managed using the previous syntax, as the input of the `mirtempo` function should not be frame-decomposed yet. The other alternative syntax consists in proposing the frame decomposition option

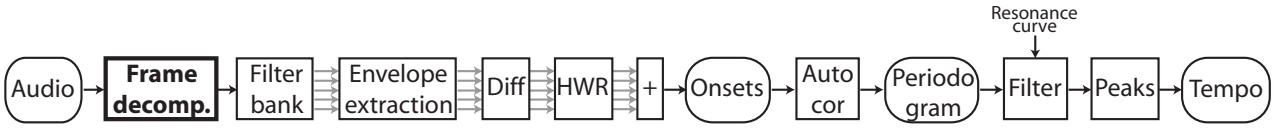

Figure 7: *First possible location of the frame decomposition step (in bold) within the chain of processes defining the tempo estimation method.*

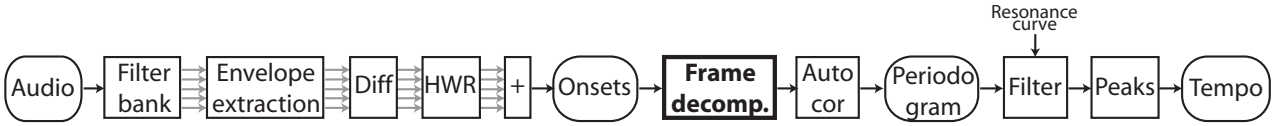

Figure 8: *Second – and more suitable – possible location of the frame decomposition step (in bold), once the onset detection curve has been computed.*

as a possible argument (`'Frame'`) of the `mirtempo` function (31). This corresponds to what was presented in section 2.4 (code lines 16 and 17).

```
mirtempo(a,'Frame',.05,.5)          (31)
```

The frame decomposition option is available as a possible argument to most of the functions of the toolbox. Each function can then specify the exact position of the frame decomposition within its chain of operations. Besides, if not specified, the default parameters of the frame decomposition – i.e., frame size and hop factor – can be adapted to each specific function. Hence, from a user's point of view, the execution and chaining of the different operators of the MIRtoolbox follow the same syntax, be there frame decomposition or not, apart from the additional use of either the command `mirframe` or the option `'Frame'` for frame decomposition. Of course, from a developer's point of view, this requires that each feature extraction algorithm should adapt to frame-decomposed input. More precisely, as will be explained in section 4.1, input can be either a single vector or a matrix, where columns represent the successive frames. Conveniently enough, in the Matlab environment, the generalization of vector-based algorithms to matrix-based versions is generally effortless.

3.3. Adaptive syntax

As explained previously, the diverse functions of the toolbox can accept alternative input:

- The name of a particular audio file (either in wav or au format) can be directly specified as input:

```
mirspectrum('myfile')               (32)
```

- The audio file can be first loaded using the `miraudio` function, which can perform diverse operations such as resampling, automated trimming of the silence at the beginning and/or at the end of the sequence, extraction of a given subsequence, centering, normalization with respect to RMS energy, etc.

```
a = mirtempo('myfile','Sampling',11025,
             'Trim','Extract',2,3,
             'Center','Normal')     (33)
mirspectrum(a)                      (34)
```

- Batch analyses of audio files can be carried out by simply replacing the name of the audio file by the keyword `'Folder'`.

```
mirspectrum('Folder')               (35)
```

- Any vector `v` computed in Matlab can be converted into a waveform using, once again, the `miraudio` function, by specifying a specific sampling rate.

```
a = miraudio{v,44100)               (36)
mirspectrum(a)                      (37)
```

- Any feature extraction can be based on the result of a previous computation. For instance, the autocorrelation of a spectrum curve can be computed as follows:

```
s = mirspectrum(a)                  (38)
as = mirautocor(s)                  (39)
```

- Product of curves [10] can be performed easily:

```
mirautocor(a)*mirautocor(s)         (40)
```

In this particular example, the waveform autocorrelation `mirautocor(a)` is automatically converted to frequency domain in order to be combined with the spectrum autocorrelation `mirautocor(s)`.

4. IMPLEMENTATION DETAILS

4.1. Data representation

All data returned by the toolbox is represented using the same general framework:

- The one-dimensional analysis of a given frame or of a whole signal is stored in a column vector, which corresponds to the first dimension in Matlab convention.
- The multiple columns corresponding to successive frame analyses are arranged row-wise (along the second dimension in Matlab convention), forming a matrix. Respectively, any two-dimensional data (such as a self-organizing map) is stored in a same matrix using the first two Matlab dimensions.
- The multiple matrices corresponding to multiple channels, when applicable (9), are arranged along the third dimension in Matlab convention, forming a 3D-matrix.
- The fourth Matlab dimension is sometimes used for more complex data. For instance, the keystrength function returns two sets of data – one for major keys, one for minor keys - that are arranged following the fourth dimension.
- These matrices (one to four-dimensional) are computed for each successive segments of a segmented audio file, when applicable (27), and stored in a Matlab cell array.
- The multiple cell arrays corresponding to the analyses of the multiple audio files of a batch of audio files are stored in another cell array.

Figure 9 shows the overall structure.

This complex data structure, although enabling to grasp all the potentiality offered by the toolbox, is rarely used in its plain capacity. Therefore, a particular mechanism has been designed in order to automatically simplify the structure, when calling the $mirgetdata$ function that return the numerical data associated to a given feature analysis.

4.2. Object-oriented architecture

The organization of the data and functions of the mirtoolbox is founded on an object-oriented architecture. The superclass from which all the data and methods are based is called `mirdata`. It contains all the information commonly used by all data.A hierarchy of classes is contructed from the *mirdata* hyperclass. The *miraudio* and *mirenvelope* classes inherit from the *mirtemporal* class, which contains particular data and methods adapted to waveforms of diverse sampling rates. For instance, the *mirplay* method plays back the audio signal. When applied to an envelope, *mirplay* actually produces a white noise featuring the same envelope.

A large number of features actually returns a single scalar value per analysed frame. They are all members of the *mirscalar* class, which features all the necessary methods for their processing, such as their graphical display in particular. The non-scalar features, on the contrary, are organized into a set of different specialised classes (*mirautocor*, *mirspectrum*, *mirhisto*, *mirmfcc*, etc.).

4.3. Memory optimization

The flexibility of the syntax requires a complex data representation that can handle alternative configurations (frame and/or channels decompositions, segmentation, batch analysis). This data structure could in theory become very extensive in terms of memory usage, especially if entire folders of audio files are loaded into the memory in one go. We have designed new methods allowing a better management of memory without deterioration of the syntactical simplicity and power. Audio files are loaded one after the other in the memory, and if necessary, long audio files are also divided into a series of successive blocks of frames that are loaded one after the other. We plan to further optimise the computational efficiency of the toolbox by proposing the possibility of distributing the computational loads among a network of computers, with the help of the *Distributed Computing* Toolbox and Engine proposed by Matlab.

4.4. Software Development Kit

The different feature extraction algorithms will be progressively refined and new features will be added in future versions of *MIRtoolbox*. Users are encouraged to write their own functions, using the building blocks offered by the current version. A set of meta-functions have been designed that enable the writing of additional algorithms using very simple function templates. As the meta-functions take care of all the complex management of the data structure and methods, the development of new algorithms can concentrate simply on the purely mathematical and DSP considerations. This may result in a computational environment where large-scale MIR systems could be developed, articulated one with each other, and compared.

5. *MIRTOOLBOX* COMPARISON TO *MARSYAS*

Marsyas is a framework written in C++ and Java for prototyping and experimentation with computer audition applications [1]. It provides a general architecture for connecting audio, soundfiles, signal processing blocks and machine learning. The architecture is based on dataflow programming, where computation is expressed as a network of processing nodes/components connected by a number of communication channels/arcs. Users can build their own dataflow network using a scripting language at run-time. Marsyas provides a framework for building applications rather than a set of applications [1] [7] Marsyas executables operate either on individual soundfiles or collections which are simple text files that contain lists of soundfiles. In general collection files should contain soundfiles with the same sampling rate as Marsyas doesn't perform automatic sampling conversion (except between 44100Hz and 22050Hz). The results of feature extraction processes are stored in Marsyas as text files that can be used later in the Weka machine learning environment. In parallel, Marsyas integrates some basic machine learning components.

Also MIRtoolbox offers the possibility of articulating process one after the other in order to construct complex computation, using a simple and adaptive syntax. Contrary to Marsyas though, MIRtoolbox does not offer real-time capabilities. On the other hand, its object-based architecture (paragraph 4.2) enables a significant simplification of the syntax. MIRtoolbox can also analyse folders of audio files, and can deal with folder of varying sampling rates without having to perform any conversion. The data computed by the MIRtoolbox can be further processed directly in the

[7] The main features currently proposed are spectral moments, flux, and rolloff, pitch and harmonicity estimation, MFCC and LPC, zero-crossing and RMS.

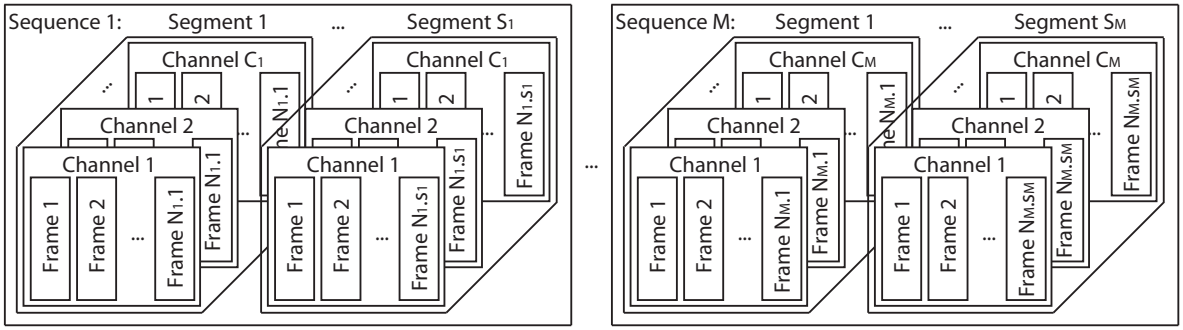

Figure 9: *Structure of the data representation used for each feature results.*

Matlab environment with the help of other toolboxes, or can be exported into text files.

6. AVAILABILITY OF THE *MIRTOOLBOX*

Following our first Matlab toolbox, called *MIDItoolbox* [18], dedicated to the analysis of symbolic representations of music, the *MIRtoolbox* is offered for free to the research community. It can be downloaded from the following URL:

`http://www.cc.jyu.fi/~lartillo/mirtoolbox`

7. ACKNOWLEDGMENTS

This work has been supported by the European Commission (NEST project "Tuning the Brain for Music", code 028570). The development of the toolbox has benefitted from productive collaborations with the other partners of the project, in particular Tuomas Eerola, Jose Fornari, Marco Fabiani, and students of our department.

8. REFERENCES

[1] G. Tzanetakis and P. Cook, "Marsyas: A framework for audio analysis," *Organized Sound*, vol. 4, no. 3, 2000.

[2] M. Slaney, "Auditory toolbox version 2," Tech. Rep., Interval Research Corporation, 1998-010, 1998.

[3] I. Nabney, *Springer Advances In Pattern Recognition Series*, chapter NETLAB: Algorithms for pattern recognition, 2002.

[4] J. Vesanto, "Proceedings of the matlab dsp conference," in *Self-Organizing Map in Matlab: the SOM Toolbox*, 1999, pp. 35–40.

[5] G. Tzanetakis and P. Cook, "Multifeature audio segmentation for browsing and annotation," in *Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 1999.

[6] A. Rauber E. Pampalk and D. Merkl, "Content-based organization and visualization of music archives," in *Proceedings of the 10th ACM International Conference on Multimedia*, 2002, pp. 570–579.

[7] E. Terhardt, "On the perception of periodic sound fluctuations (roughness)," *Acustica*, vol. 30, no. 4, pp. 201–213, 1974.

[8] P. Boersma, "Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound," *IFA Proceedings*, vol. 17, pp. 97–110, 1993.

[9] T. Tolonen and M. Karjalainen, "A computationally efficient multipitch analysis model," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 6, pp. 708–716, 2000.

[10] G. Peeters, "Music pitch representation by periodicity measures based on combined temporal and spectral representations," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006.

[11] L. Rabiner and B. H. Juangl, *Fundamentals of Speech Recognition*, Prentice-Hall, 1993.

[12] E. Gomez, "Tonal description of polyphonic audio for music content processing," *INFORMS Journal on Computing*, vol. 18, no. 3, pp. 294–304, 2006.

[13] C. Krumhansl, *Cognitive Foundations of Musical Pitch*, Oxford University Press, 1990.

[14] C. Krumhansl and E. J. Kessler, "Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys," *Psychological Review*, vol. 89, pp. 334–368, 1982.

[15] P. Toiviainen and C. Krumhansl, "Measuring and modeling real-time responses to music: The dynamics of tonality induction," *Perception*, vol. 32, no. 6, pp. 741–766, 2003.

[16] P. Toiviainen and J.S. Snyder, "Tapping to bach: Resonance-based modeling of pulse," *Music Perception*, vol. 21, no. 1, pp. 43–80, 2003.

[17] J. Foote and M. Cooper, "Media segmentation using self-similarity decomposition," in *Proceedings of SPIE Storage and Retrieval for Multimedia Databases*, 2003, number 5021, pp. 167–175.

[18] T. Eerola and P. Toiviainen, "MIR in Matlab: The Midi Toolbox," in *Proceedings of 5th International Conference on Music Information Retrieval*, 2004, pp. 22–27.

[19] P. N. Juslin, "Emotional communication in music performance: A functionalist perspective and some data," *Music Perception*, vol. 14, pp. 383–418, 1997.

[20] K. R. Scherer and J. S. Oshinsky, "Cue utilization in emotion attribution from auditory stimuli," *Motivation and Emotion*, vol. 1, no. 4, pp. 331–346, 1977.