The Timbre Toolbox

A MATLAB toolbox for extracting Audio Descriptors

User's Manual

Version: R1.0

Authors: S. Kazazis, Ph. Depalle, S. McAdams

Release Notes

The Timbre Toolbox started from the MATLAB version of Ircam Descriptors by G. Peeters with contributions from J. Krimphoff, N. Misdariis, P. Susini, and S. McAdams.

The first modification was done by G. Peeters and accompanies the paper:

> Peeters, G., Giordano, B.L., Susini, P., Misdariis, N. & McAdams, S. (2011). The Timbre Toolbox: Extracting audio descriptors from musical signals. Journal of the Acoustical Society of America, 130, 2902-2916.

The second modification was done by C. Kereliuk at McGill University who made it object oriented.

The third modification was done by N. Esterer at McGill University and accompanies the paper:

> Kazazis, S., Esterer, N., Depalle, P. & McAdams, S. (2017). A performance evaluation of the Timbre Toolbox and the MIRToolBox on calibrated test sounds. In G. Scavone, E. Maestre, C. Kemp & S. Wang (Eds.), Proceedings of the 2017 International Symposium on Musical Acoustics, Montreal, June 18-22, 2017 (pp. 144-147).

`https://github.com/mondaugen/timbretoolbox`

The fourth modification was done by V. Perreault at McGill University.
`https://github.com/VincentPerreault0/timbretoolbox`

The current version (TimbreToolbox-R1.0) was reprogrammed from scratch by S. Kazazis, P. Depalle, and S. McAdams at McGill University.
`https://github.com/MPCL-McGill/TimbreToolbox-R2021a`

Many computations of audio descriptors are different than the ones presented in the Peeters et al. (2011) paper. New descriptors are also added.

Citing this version of the toolbox

If this version of the Timbre Toolbox is used for publications it can be cited as:

Kazazis, S., Depalle, P., and McAdams, S. (2022). "The Timbre Toolbox User's Manual", https://github.com/MPCL-McGill/TimbreToolbox-R2021a (Last viewed yyyy/mm/dd).

Compatibility with MATLAB versions and Dependencies

The Timbre Toolbox version R1.0 requires MATLAB's Signal Processing Toolbox.

This version has been tested on the following MATLAB versions:
R2020a, R2020b, R2021a.

Contents

Installation

Download the toolbox either using git or clicking on the "Download ZIP" link from:
https://github.com/MPCL-McGill/TimbreToolbox-R2021a

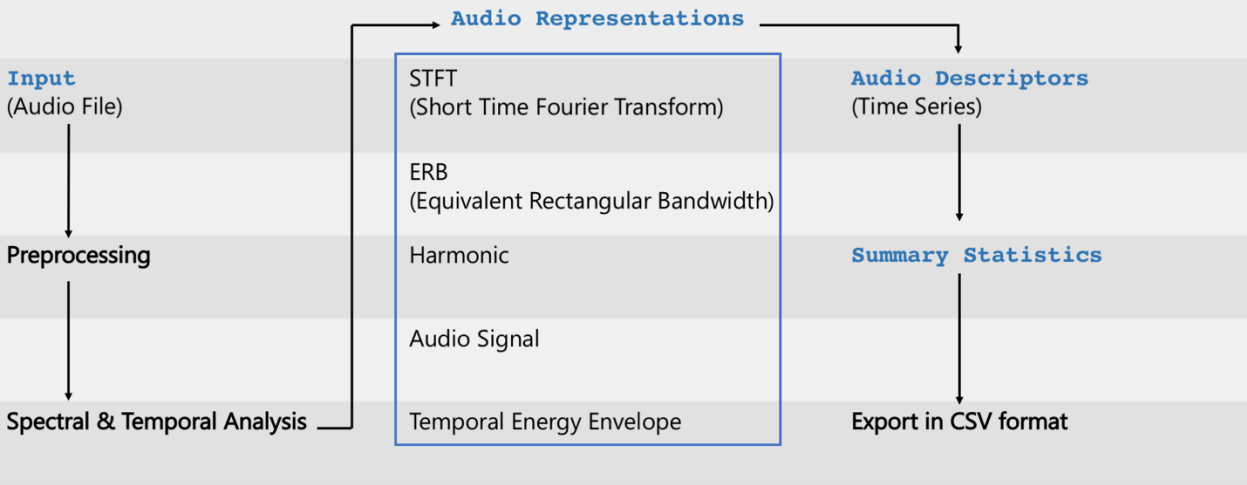
If the file is downloaded through the Code > "Download ZIP" link, decompress the archive to some location (e.g. in Documents > MATLAB) and RENAME the decompressed folder to: **TimbreToolbox-R2021a**.

Add the folder to MATLAB's path, e.g., by using the MATLAB browser:

1. select the folder
2. right-click on it
3. select Add to Path > Selected Folders and Subfolders

Overview

The following diagram illustrates the basic workflow of the toolbox.



The input audio file is first preprocessed (e.g., stereo to mono conversion).

The audio file is further analyzed to estimate the temporal and spectral parameters that are required for each audio representation (shown inside the blue square).

The audio descriptors available for each representation are then extracted, and their time-varying values are summarized according to the summary statistics.

The results (time-series and summary statistics) can also be exported in a .csv format for further processing outside MATLAB's programming environment.

All the above steps are described in detail in the respective sections.

Command Window

The main function called to compute descriptors is `TTdescriptors` which accepts the following six arguments (args):

```
>> TTdescriptors(soundsDirectory, resultsDirectory, Files, ...
Reps, Descs, Stats);[1]
```

See also the Timbre Toolbox's `Documentation` folder for examples.

Basic Input Arguments

arg1: Directory of the Audio Files (input)

This is the folder's *path* in which the sounds to be analyzed are located, e.g.:

```
soundsDirectory = ...
'/Users/SomeUser/Documents/MATLAB/TimbreToolbox-
R2021a/Documentation/SoundsToAnalyze';
```

In the above statement, the folder's name which contains the audio files is ***`SoundsToAnalyze`*** and is located inside the ***`Documentation`*** folder of the Timbre Toolbox. However, it can be located elsewhere, outside the Timbre Toolbox.

In case the folder does not exist (or, if the path is misspelled in the above statement) the program will terminate with the following *error* message:

```
The folder in:
/Users/SomeUser/Documents/MATLAB/TimbreToolbox-R2021a/doc/SoundsToAnalyze
does not exist.
```

**** It is advisable to always keep a copy of your audio files elsewhere. ****

[1] The three dots are necessary to continue the statement in the next line.

arg2: Directory of the Results (output)

This is the folder's *path* in which all the analysis results will be saved, e.g.:

```
resultsDirectory = ...
'/Users/SomeUser/Documents/MATLAB/TimbreToolbox-R2021a/doc/Results';
```

In the above statement, the name of the folder that will contain the analysis results is `Results` and is located inside the `doc` folder of the Timbre Toolbox. However, it can be located elsewhere, outside the Timbre Toolbox.

> If the folder does not exist (or, if the path is misspelled in the above statement) the program will terminate with the following *error* message:

```
The folder in:
/Users/SomeUser/Documents/MATLAB/TimbreToolbox-
R2021a/Documentation/Results
does not exist.
```

**** Before running a new analysis, it is advisable to store the* `Results` *folder elsewhere and create a new one. Any previous results will be overwritten by the new analysis. ****

arg3: Audio Files

The following three options are provided for specifying which audio files to analyze.

- Analyze *all* audio files by using the keyword `ALL` (*character array*):

  ```
  >> Files = 'ALL';
  ```

  Invalid files or audio files with unsupported audio file formats will be excluded from the analysis without a *warning* message.

- Analyze a *single* audio file by specifying its name and file extension (*character array*):

  ```
  >> Files = 'SomeAudioFile.wav';
  ```

- Analyze a *list* of audio files specified inside a *cell array*[2]:

  ```
  >> Files = {'SomeFile_1.wav', 'SomeFile_2.wav'};
  ```

[2] In MATLAB, a cell array is constructed with the use of curly brackets.

Warnings and Errors

> If the specified file exists inside the `soundsDirectory` but is of an unsupported audio format, e.g.:

```
Files = 'SomeAudioFile.m';
```

The following *warning* message will be issued:

```
Warning:  'SomeAudioFile.m' will not be analyzed.
(Unknown or unsupported audio file format)
```

> If the specified file does not exist inside the `soundsDirectory` but is of a supported audio file format e.g.:

```
Files = 'SomeAudioFile.wav';
```

The following warning message will be issued:

```
Warning:  The audio file 'SomeAudioFile.wav' was not found.
```

> If no valid files are found inside the `soundsDirectory` the program will terminate with the following *error* message:

```
No valid audio files to analyze in:
/Users/SomeUser/Documents/MATLAB/TimbreToolbox-
R2021a/doc/SoundsToAnalyze
```

arg4: Audio Representations

The same options (as in arg3: *character* and *cell* arrays) are provided for specifying which signal representation to use, in order to compute the relevant audio descriptors. The signal representations are specified using the following *keywords* (case-sensitive):

`ALL`	Computes all signal representations.
`TEErep`	The descriptors are computed using the Temporal Energy Envelope of the audio signal.
`PowSTFTrep`	The descriptors are computed using the *power* spectrum of the short time Fourier transform (STFT).
`MagSTFTrep`	The descriptors are computed using the *magnitude* spectrum of the STFT.
`HARMrep`	The descriptors are computed on harmonic and inharmonic partials.
`ERBrep`	Similar to the `powSTFTrep` except that the frequency spectrum of the audio signal is partitioned into a number of adjacent equivalent rectangular bandwidth (ERB) filters.

`ASrep`	The descriptors are computed using the (temporal) waveform.

To get the full list of audio representations, type in the command window:

```
>> AudioRepresentations
```

Examples

```
>> Reps = 'ALL';
>> Reps = 'HARMrep';
>> Reps = {'powSTFTrep', 'ERBrep', TEErep'};
```

Warnings and Errors

- If the statement contains invalid representations, e.g.,
  `Reps = 'STFTre';`
  The following *warning* will be issued:
  `Warning: 'STFTre' is an invalid representation and will be ignored.`

- If all specified representations are invalid, the Timbre Toolbox will terminate with the following *error*:
  `No valid representations to compute.`

See the section "Signal Representations" for more details on the representations.

arg5: Audio Descriptors

The same options (*character* and *cell* arrays) are provided for specifying which audio descriptors to compute. All keywords are case-sensitive.

- To get the full list of audio descriptors related to each representation, type in the command window:

`AudioSignalDescriptors`	Descriptors related to the `ASrep`.
`HarmonicDescriptors`	Descriptors related to the `HARMrep`.

`SpectralDescriptors`	Descriptors related to the `MagSTFTrep`, `PowSTFTrep`, `ERBrep`.
`TemporalEnergyEnvelopeDescriptors`	Descriptors related to the `TEErep`.

The descriptors specified here should belong to any of the representations given in arg4, otherwise a warning will be issued (see Warnings and Errors, below).

Examples

The *keyword* `ALL` computes all descriptors, which are available for a given representation:

```
>> Descs = 'ALL';
```

The *keyword* `spectralCentroid` computes the spectral centroid:

```
>> Descs = 'spectralCentroid';
```

The *keywords* `spectralSpread` and `spectralSkewness` compute the spectral spread and spectral skewness respectively:

```
>> Descs = {'spectralSpread', 'spectralSkewness'};
```

Warnings and Errors

- If the statement contains invalid descriptors (e.g., '`specCent`', the following *warning* will be issued:
  `Warning: Unknown descriptor: specCent`

- If all descriptors are invalid, or don't belong to the specified representation, the following *error* will be issued:
  `No descriptors to compute for the specified representations.`

arg6: Summary Statistics

The same options (*character* and *cell* arrays) are provided for specifying which summary statistics to compute on the time-series of descriptor values. All *keywords* are case-sensitive. The following summary statistics are available:

`ALL`	Computes all statistics.
`Median`	Computes the median.
`Mean`	Computes the mean.
`IQR`	Computes the interquartile range.
`Std`	Computes the standard deviation.
`Min`	Computes the minimum.
`Max`	Computes the maximum.

To get the list of available summary stats type in the command window:

```
>> SummaryStatistics
```

Examples

```
>> Stats = 'ALL';
>> Stats = {'Median', 'IQR'};
```

Warnings and Errors

- If the summary statistic is invalid (e.g., 'Mi '), the following *error* will be issued:
  `Invalid summary statistic: 'Mi'.`

Using Fewer Arguments

It is also possible to call `TTdescriptors` with fewer arguments:

```
>> TTdescriptors(soundsDirectory, resultsDirectory)

>> TTdescriptors(soundsDirectory, resultsDirectory, Files)

>> TTdescriptors(soundsDirectory, resultsDirectory, Files, Reps)

>> TTdescriptors(soundsDirectory, resultsDirectory, Files, ...
Reps, Descs)
```

The default settings for the missing arguments are:

```
Files  ALL
Reps   PowSTFTrep
Descs  ALL
Stats  Median
```

Advanced Settings

The default settings of the representations can be changed using a *struct* with specific fields.

The general format is:

```
arg4.representation.parameter = value
```

arg4:	is the 4th argument of the function `TTdescriptors`.
representation:	which audio representation to compute
parameter:	an editable parameter
value:	parameter's value

> To get the full list of editable parameters related to each representation, type in the command window:

`ASrepSettings`	Available struct fields for the `ASrep`
`ERBrepSettings`	Available struct fields for the `ERBrep`

`HARMrepSettings`	Available struct fields for the `HARMrep`
`STFTrepSettings`	Available struct fields for the `PowSTFTrep` and `MagSTFTrep`
`TEErepSettings`	Available struct fields for the `TEErep`

See the section 'Signal Representations' for an explanation of the editable parameters.

Parameters that are not specified by the user will be set to their default values.

Examples

```
>> Reps.HARMrep.winLength_s = 0.1; % Window length in seconds
```
Computes the harmonic representation using a window length of 100 ms.

```
>> Reps.PowSTFTrep.winType = 'hamming';
>> Reps.PowSTFTrep.winLength = 2048; % Window length in samples
```
Computes the power STFT representation using a Hamming window of 2048 samples.

Once the fields of the struct are specified, the function `TTdescriptors` will compute the descriptors with the user-specified settings:

```
>> TTdescriptors(soundsDirectory, resultsDirectory, Files, ...
Reps, Descs, Stats);[3]
```

See the MATLAB examples in the documentation folder for the complete list of editable parameters.

Errors

Audio representations cannot be specified both as a *character* (or *cell* array) and as a *struct*. In this case, MATLAB will issue an *error*.

For example, the following assignment will result in an *error*:
```
>> Reps = 'HARMrep';
>> Reps.PowSTFTrep.winType = 'hamming';
```

[3] The three dots are necessary to continue the statement in the next line.

- If any of the parameters (fields) is invalid (e.g., '`winL'`), the following *error* will be issued:
  ```
  Invalid field name: 'winL'.
  ```

- Parameters of a particular representation cannot be specified both in seconds and in samples. For example, the following statement will result in an *error*:
  ```
  >> Reps.PowSTFTrep.winLength   = 2048;
  >> Reps.PowSTFTrep.hopSize_s   = 0.1;

  'winLength' and 'hopsize_s' (default) units do not agree.
  ```

- Same parameters of a particular representation cannot be specified both in seconds and in samples. In that case, an *error* will be issued:

  ```
  'winLength' cannot be specified both in samples and in seconds.
  ```

Similar *errors* will be issued for any invalid settings of the parameters. If the audio files are of different sample rates it is advisable to specify the parameters in seconds (default setting).

Graphical User Interface

The Timbre Toolbox can also be run through a GUI. The results will be stored inside the folder:
`Documentation > Results`

There are two different versions available, implemented as functions.

- The function `TT_GUIdefaults.m` opens the GUI and the audio files will be analyzed with the default settings of each audio representation. This function is located inside the folder `TT_GUIdefaults`. To open the GUI type in the command line `TT_GUIdefaults` or open the function and click "Run" in the editor.

- The function `TT_GUIsettings.m` opens the GUI but the settings of each audio representation can be customized. This function is located inside the folder `TT_GUIsettings`. To open the GUI type in the command line `TT_GUIsettings` or open the function and click "Run" in the editor.

These functions can also be run from the scripts located inside the main folder of the toolbox: `Run_GUIdefaults.m` and `Run_GUIsettings.m`.

See also the examples provided inside the `Documentation` folder.

Default Settings (`TT_GUIdefaults.m`)

The main panel consists of the following items.

`Audio Files`: Select which audio files to analyze.

`Results Folder`: Select the folder in which the analysis results will be stored.

`Audio Representations`: Select which audio representations to compute.

`Audio Descriptors`: Select which audio descriptors to compute.

- The option ALL (default) corresponds to all relevant descriptors to the selected audio representation.
- The option `Select…` allows one to select a subset of descriptors that relate to the selected audio representation. Descriptors that cannot be computed from the selected representations will be unavailable.

`Summary Statistics`: Select which summary statistics to compute.

There is also an option to export the summary statistics to .csv format by ticking:
`Export Summary Statistics to .csv format`

`Analyze`: Runs the analysis.
Any errors and warnings will be displayed in the `Command Window`.

`Quit`: Exits the GUI.

Advanced Settings (`TT_GUIsettings.m`)

This GUI is essentially the same as the one described above. The only difference is that the **`Audio Representations`** item opens a new window where the default settings of each representation can be changed.

Click **`Done`** in order to compute this representation with the specified settings. If you click again on **`Audio Representations`**, the representation will be ticked (selected).

Pressing **`Cancel`** will exit the settings, untick previously selected representations, and will not compute the representation.

Signal Representations

Before computing any audio representation, the audio signal is pre-processed in the following ways:

1. If the audio file is stereo, only the channel with the greatest energy (RMS value) will be analyzed.
2. The DC component is removed, i.e., a mean value of the whole signal that is different from zero.
3. The audio file is normalized to unit-amplitude.

These pre-processing steps can be overridden by commenting the respective lines in `preprocess_audioFile.m`, which is located inside the `Functions` folder.

Temporal Energy Envelope (TEErep)

This representation uses the power amplitude envelope of the audio signal, as well as the raw waveform (in power dBFS). It should only be used for single audio events (e.g., for analyzing a single sound and not a sequence of sounds).

To calculate the energy envelope, the waveform is first segmented using a window length of 200 ms and a hop size of 100 ms. The maxima across successive windows are then interpolated using piecewise cubic interpolation (`pchip`).

The following parameters can be changed through the GUI or the command window:

Parameter	Default (in dBFS)	Description
`noiseTHD`	-inf	Noise threshold (thd). Values below this thd will be discarded from the computation of *attack time* and *temporal centroid*.
`attackTHD`	0	Threshold up to which *attack time* will be computed.

`decreaseSlopeTHD`	-12	Threshold up to which *decrease slope* will be computed.
`effectiveDurationTHD`	-20	Threshold above which *effective duration* will be computed.

All thresholds should be specified in dBFS. The default settings can also be found in the GUI or by typing:
`>>TEErepSettings`

Note that this representation does not compute time-varying descriptors.

For this particular representation, it is important that the sounds be as clean as possible with very little or no background noise. If background noise is present, then the `noiseTHD` should be adjusted accordingly.

Note also that a silent segment before the sound's onset will lead to incorrect values of *attack time*. This can be avoided by slightly increasing the `noiseTHD` (e.g., to –100 dBFS).

In a similar way, "clicks" due to recording artifacts or any other processing may also lead to incorrect values of attack time. In some cases (i.e., if the "click" is stronger than the energy of the audio signal) this can be avoided by slightly lowering the `attackTHD` (e.g., to –1 dBFS)

By default, `effectiveDurationTHD` is also used to compute the energy modulation. See the subsection Energy Modulation for more details.

STFT (MagSTFTrep, PowSTFTrep)

This representation relies on the Short Time Fourier Transform (STFT) of the audio signal. In `PowSTFTrep` the magnitudes of the STFT bins are squared. The following parameters can be changed through the GUI or the command window:

Parameter	Default	Description
`winType`	`hann`	Window type.
`winLength_s`	0.04645 s.	Window length in seconds.
`hopSize_s`	0.04645/4 s.	Hop-size in seconds.
`winLength`	–	Window length in samples.
`hopSize`	–	Hop-size in samples.

Note that window length and hop-size must be specified either in samples or in seconds (default) otherwise an *error* will be issued.

The default settings and available window types can also be found in the GUI or by typing:
`>>STFTrepSettings`

Harmonic (HARMrep)

This representation should ideally be used only on *pitched* and *monophonic* sounds. It relies on partial tracking (sinusoidal analysis) for estimating harmonic and inharmonic components that are required for calculating harmonic descriptors.

For the computations of:

```
harmonicEnergy
harmonicOddToEvenRatio
harmonicSpectralDeviation
tristimulusValues
```

only one partial per ideal harmonic is taken into account whose frequency depends on the value of *inharmonicity tolerance* (described below) and maximum amplitude: if there are many detuned partials that fall within the range of *inharmonicity tolerance* (per harmonic number), only the partial with the greatest amplitude will be considered in the computation of these descriptors.

Prior to partial tracking, the frequencies and amplitudes of spectral peaks are estimated using quadratic interpolation. By default, spectral peaks below –90 dBFS are not included in the sinusoidal analysis. This setting can be changed inside the function `quadmaxloc.m` which is located inside:
`Classes > Representations > @cHARMrep > private`

The partial tracking relies on the algorithms of:

Lagrange, M., Marchand S., Raspaud, M., and Rault, JB. (2003). "Enhanced partial tracking using linear prediction," in *Proc. Digital Audio Effects (DAFx) Conf.* (London, UK).

McAulay, R.J., and Quatieri, T.F. (1986). "Speech analysis/synthesis based on a sinusoidal representation," IEEE ICAASP, vol. 34(4), 744-754.

Pitch estimation is based on the algorithm of:

Camacho, A., and Harris, J.G. (2008). "A sawtooth waveform inspired pitch estimator for speech and music," J. Acoust. Soc. Am. 124, 1638-1652.

The following parameters can be changed through the GUI or the command window:

Parameter	Default	Description
`winType`	`blackman`	Window type.
`winLength_s`	0.04645 s.	Window length in seconds.
`hopSize_s`	0.04645/4 s.	Hop-size in seconds.
`winLength`	–	Window length in samples.
`hopSize`	–	Hop-size in samples.
`magnitudeThreshold`	–30 dBFS	Spectral peaks below this threshold from the maximum peak of each analysis window will not be included in partial tracking.
`minPartialsDuration`	0.05 s.	Minimum partial's duration in seconds. Partials below this duration will be excluded from the computation of descriptors.
`pitchRange`	[30, 5000] Hz	Pitch range. The pitch estimation is restricted within this interval. If the sound has a pitch below or above this range, the upper or lower values should be increased or decreased. Narrower pitch ranges also speed up the computations.
`inharmonicityTolerance`	0.5	Partials' maximum allowable deviation from ideal harmonics. For example, a value of 0.5 indicates that the n^{th} partial may deviate as much as 50% above or below its ideal n^{th} harmonic location. Use lower values to restrict the analysis only to harmonic or slightly inharmonic sounds (e.g., 0.05).

Note that window length and hop-size must be specified either in samples or in seconds (default) otherwise an *error* will be issued.

The default settings can also be found in the GUI or by typing:
`>> HARMrepSettings`

- Although the partial tracking algorithm can be used with polyphonic sounds, the *pitch* estimation will not be reliable.

- It is important that the pitches of all sounds be inside the `pitchRange`. If not, the pitch estimation will be erroneous (e.g., may return sub-harmonics of the true pitch). Note also that increasing the pitch range will also increase the computation time.

- If no partials are found using the current settings of HARMrep, a *warning* will be issued, and MATLAB will terminate the analysis with an *error*.
  `Warning: No partials found using the current settings of HARMrep.`

- Note that the formal definitions of *Odd-to-Even ratio* (`harmonicOddToEvenRatio`) and *Tristimulus values* (`tristimulusValues`) refer to strictly harmonic or slightly inharmonic partials (e.g., up to 5% deviation from ideal harmonics: `inharmonicityTolerance` = 0.05). Setting `inharmonicityTolerance` to its maximum value, the computations of those descriptors may include inharmonic components. If that is not intended, the value of `inharmonicityTolerance` should be lowered.

- The same applies for the computations related to *harmonic energy* (`harmonicEnergy`) and *harmonic spectral deviation* (`harmonicSpectralDeviation`): if `inharmonicityTolerance` is high, then the computations of these descriptors may also include inharmonic components.

- If the sustained part of the sound is relatively long, the number of tracked partials can be reduced to the most stable ones by increasing `minPartialsDuration`.

Equivalent Rectangular Bandwidth (ERBrep)

This representation relies on the Equivalent Rectangular Bandwidth (ERB) proposed by:

Moore, B. C. J., and Glasberg, B. R. (1983). "Suggested formulae for calculating auditory-filter bandwidths and excitation patterns," J. Acoust. Soc. Am. 74, 750–753.

The frequency spectrum is partitioned into adjacent ERB filters through a gammatone filterbank:

Patterson, R., Robinson, K., Holdsworth, J., McKeown, D., Zhang, C., and Allerhand, M. (1992). "Complex sounds and auditory images," Aud. Physiol. Percept. 83, 429–446.

The following parameters can be changed through the GUI or the command window:

Parameter	**Default**	**Description**
`winLength_s`	0.02 s.	Window length in seconds.
`hopSize_s`	0.01 s.	Hop-size in seconds.
`winLength`	–	Window length in samples.
`hopSize`	–	Hop-size in samples.
`fLow`	50 Hz.	Lowest center frequency of the gammatone filterbank
`nChannels`	64	Number of channels (or filters) ERB-spaced between `fLow` and Nyquist frequency.

Note that window length and hop-size must be specified either in samples or in seconds (default) otherwise an *error* will be issued.

The default settings can also be found in the GUI or by typing:

```
>> ERBrepSettings
```

Audio Signal (ASrep)

This representation relies on the "raw" audio signal.
The following parameters can be changed through the GUI or the command window:

Parameter	Default	Description
`winLength_s`	0.02 s.	Window length in seconds.
`hopSize_s`	0.01 s.	Hop-size in seconds.
`winLength`	–	Window length in samples.
`hopSize`	–	Hop-size in samples.

Note that window length and hop-size must be specified either in samples or in seconds (default) otherwise an *error* will be issued.

The default settings can also be found in the GUI or by typing:

```
>> ASrepSettings
```

Audio Descriptors

This section provides the methods and equations used to calculate the descriptors. Similar explanations along with references for some audio descriptors can be found in:

Peeters, G., Giordano, B.L., Susini, P., Misdariis, N. & McAdams, S. (2011). The Timbre Toolbox: Extracting audio descriptors from musical signals. Journal of the Acoustical Society of America, 130, 2902–2916.

Spectral Descriptors

The spectral descriptors can be computed from the following audio representations: `MagSTFrep`, `PowSTFTrep`, `HARMrep`, and `ERBrep`.

Therefore, in the following, the index *k* may refer to FFT bin number, partial, or to a channel of the ERB filterbank. Similarly, *f* and *a* may refer to the frequency (in Hz) and amplitude, respectively, of the FFT bins, partials, or channels of the ERB filterbank.

p_k is the normalized form of a_k and is defined as:

$$p_k = \frac{a_k}{\sum_{k=1}^{K} a_k} \quad ,$$

where *K* is the total number of FFT bins[4], partials, or channels of the ERB filterbank.

Spectral Centroid: `spectralCentroid.m`

The spectral centroid represents the spectral center of gravity. It is defined as the frequency weighted average of the (power) spectrum:

$$m_1 = \sum_{k=1}^{K} f_k \cdot p_k$$

[4] K refers to the number of bins of the one-sided spectrum.

Spectral centroid is related to auditory brightness. Low centroid values indicate a dark sound and high values a bright sound. However, it also increases in the presence of noise, and it tends to fluctuate to a great extent during the transient regions of sound events. In many cases it is also correlated with fundamental frequency and pitch.

Spectral Spread: `spectralSpread.m`

Spectral spread (also known as instantaneous bandwidth and spectral standard deviation) is defined as:

$$m_2 = \left(\sum_{k=1}^{K} (f_k - m_1)^2 \cdot p_k \right)^{1/2}$$

It measures the standard deviation of the spectrum around the spectral centroid. High values indicate a rich spectrum.

Spectral Skewness: `spectralSkewness.m`

Spectral skewness is defined as the third statistical moment of the spectrum:

$$m_3 = \left(\sum_{k=1}^{K} (f_k - m_1)^3 \cdot p_k \right) \Big/ m_2^3$$

It is a measure of the asymmetry of the spectrum around the spectral centroid. Zero skewness indicates a symmetric distribution around the spectral centroid. Negative skewness indicates more energy at lower frequencies whereas positive skewness indicates more energy at higher frequencies. Most instrument sounds (rich in overtones or harmonics) exhibit positive skewness. However, mixtures of different sound sources may exhibit nearly symmetric distributions.

Spectral Kurtosis: `spectralKurtosis.m`

Spectral kurtosis is related to the fourth statistical moment of the spectrum and measures the departure from Gaussianity of the spectrum around the spectral centroid. It is defined as:

$$m_4 = \left(\sum_{k=1}^{K} (f_k - m_1)^4 \cdot p_k \right) \Big/ m_2^4$$

Values close to 3 indicate a Gaussian (mesokurtic) distribution which for natural sounds relates to the presence of stationary Gaussian noise. Values above 3 indicate a peakier distribution (leptokurtic) and values below 3 a flatter distribution (platykurtic).

Spectral Slope: `spectralSlope.m`

Spectral slope estimates the slope of the spectral envelope over frequency. Spectral slope is defined as:

$$\text{slope} = \frac{\sum_{k=1}^{K} (f_k - \mu_f)(a_k - \mu_a)}{\sum_{k=1}^{K} (f_k - \mu_f)^2}$$

This descriptor is an approximation of the spectral shape computed by a linear regression over the spectral magnitudes. Most instrument sounds exhibit negative spectral slopes because the energy of the upper harmonics decreases by harmonic number. The fundamental waveforms used in analog synthesis also have different slopes. For instance, the (negative) slope of a triangular waveform is twice as steep compared to that of a square waveform and is often described as “warmer” than the square waveform. A spectral slope close to zero indicates a flat spectrum (equal energy at all frequencies) and is related to “noisy” regions. In many cases, spectral slope is correlated with spectral centroid.

Spectral Decrease: `spectralDecrease.m`

Similar to spectral slope, spectral decrease is a measure of steepness of the decrease of the spectrum. It averages the set of slopes between frequencies f_k and f_1 and therefore emphasizes the slopes of the lowest frequencies:

$$\text{decrease} = \frac{\sum_{k=2}^{K} \frac{1}{k-1} \cdot (|a_k - a_1|)}{\sum_{k=2}^{K} a_k}$$

Spectral Roll-Off: `spectralRollOff.m`

Spectral roll-off is defined as the frequency below which a percentage of the signal's energy is contained:

$$\sum_{f=0}^{f_c} a_f^2 = 0.95 \sum_{f=0}^{sr/2} a_f^2$$

By default, this percentage is set to 95%. However, this threshold can easily be changed inside the function. For harmonic sounds spectral roll-off is related to the harmonic-to-noise cut off frequency. Low values indicate the absence of energy at high frequencies. Although it is zero for perfectly silent (zero amplitude) segments, it can be quite large in the presence of low-level noise during silent segments and pauses (e.g., noise due to AD/DA conversion, or background noise picked up by the microphone).

Spectral Flatness: `spectralFlatness.m`

Spectral flatness is a measure of noisiness of the spectrum. It is defined as:

$$\text{flatness} = \frac{\left(\prod_{k=1}^{K} a_k\right)^{1/K}}{\frac{1}{K}\sum_{k=1}^{K} a_k}$$

High values indicate a flat (and therefore possibly noisy) spectrum which often sounds "harsh".

Spectral Crest: `spectralCrest.m`

In contrast to spectral flatness, spectral crest is a measure of the "peakiness" of the spectrum or its tonalness. It is defined as:

$$\text{crest} = \frac{\max_K a_k}{\frac{1}{K}\sum_{k=1}^{K} a_k}$$

Low values indicate a flat spectrum whereas high values indicate a peaky spectrum consisting of strong sinusoidal components (which relate to its tonal quality).

Spectral Flux: `spectralFlux.m`

Spectral flux represents the amount of variation of the spectrum over time. It is defined as:

$$\text{flux}(t_m) = \frac{\sqrt{\sum_{k=1}^{K} |a_k(t_m) - (a_k(t_{m-1})|^2}}{K}$$

Low values indicate low amplitude waveform segments or a stationary spectrum, which can be noisy, harmonic, or inharmonic. High values indicate strong spectral changes (e.g., due to transients, or new note events).

Spectral Variation: `spectralVariation.m`

Similar to spectral flux, spectral variation is another measure of spectrum variability over time. It is defined as:

$$\text{variation}(t_m) = 1 - \frac{\sum_{k=1}^{K} a_k(t_{m-1}) a_k(t_m)}{\sqrt{\sum_{k=1}^{K} a_k(t_{m-1})^2} \sqrt{\sum_{k=1}^{K} a_k(t_m)^2}}$$

Harmonic Descriptors

Harmonic descriptors are computed from the `HARMrep`.

Note that the accuracy of most harmonic descriptors depends on the accuracy of the pitch estimation algorithm (`swipep.m`). The harmonics are estimated as multiples of the estimated pitch.

Harmonic Energy: `harmonicEnergy.m`

The function `harmonicEnergy` computes various descriptors related to the energy of the signal.

- **Harmonic Energy** is defined as:

  $$E_H = \sum_{h=1}^{H} a_h^2$$

  Note that depending on the settings of `inharmonicityTolerance` (see the respective section of Signal Representations) a_h may refer to harmonic or inharmonic partials. However, only one partial per harmonic rank (i.e., the strongest one) is taken into account (see also the computation of `inharmonicity.m`).

- **Noise Energy** Is computed by subtracting the energy of all tracked partials (i.e., not just one partial per harmonic rank) from the total energy:

  $$E_N = E_T - E_P$$

  where Partial Energy

  $$E_P = \sum_{p=1}^{P} a_p^2$$

- **Noisiness** is defined as the ratio of noise energy to total energy:

  $$N = E_N / E_T$$

- **Harmonic to Noise Ratio** is defined as:

$$\mathrm{HNR} = E_H / E_N$$

- **Partial Energy to Noise Ratio** is defined as:

$$\mathrm{PNR} = E_P / E_N$$

Harmonic Odd to Even Ratio: `harmonicOddToEvenRatio.m`

The odd-to-even ratio is the ratio of energies of the odd to the even harmonics. High odd-to-even ratio indicates more energy at the odd harmonics (e.g., the clarinet) and often results in "hollow" sounds. A lower odd-to-even ratio indicates a smoother spectrum and a "fuller" sound (e.g., the trumpet). The square and triangle waveforms used in analog synthesis also have a high odd-to-even ratio (energy only in the odd harmonics), whereas the richer spectrum of the sawtooth waveform has a lower ratio (energy in both the odd and even harmonics). The odd-to-even ratio for (even *H*) is defined as:

$$\mathrm{OER} = \frac{\sum_{h=1}^{H/2} a_{2h-1}^2}{\sum_{h=1}^{H/2} a_{2h}^2}$$

Harmonic Spectral Deviation: `harmonicEnergy.m`

Harmonic spectral deviation measures the deviation of partials' amplitudes from a global spectral envelope (*SE*). It is similar to the odd-to-even ratio but can also be applied to inharmonic sounds. It is defined as the normalized sum of differences between the magnitude of a center frequency and the average magnitude of itself and its two neighboring frequencies. It is a measure of "jaggedness" of the spectrum. Some sounds tend to sound "nasal" as spectral deviation increases (depending on the fundamental frequency). However, extremely low spectral deviation indicates a flat spectrum which often sounds "harsh" (depending on the spacing of the overtones). It is defined as:

$$\mathrm{HDEV} = \frac{1}{H}\sum_{h=1}^{H}(a_h - SE(f_h))$$

The spectral envelope is defined as the average of three adjacent harmonic amplitudes:

$$\mathrm{SE}(f_h) = \frac{1}{3}(a_{h-1} + a_h + a_{h+1})$$

Inharmonicity: `inharmonicity.m`

Inharmonicity is measured according to:

$$\mathrm{inh} = \frac{\sum_{p=1}^{P} d_p a_p^2}{\sum_{p=1}^{P} a_p^2}$$

where *d* is a factor that measures the deviation of each partial from its ideal harmonic rank:

$$d_h = \begin{cases} 2 \cdot (h - \lfloor h \rfloor), & h - \lfloor h \rfloor \leq 0.5 \\ 2 \cdot (1 - (h - \lfloor h \rfloor)), & h - \lfloor h \rfloor > 0.5 \end{cases}$$

p refers to a partial's index.
h is the actual (possibly non-integer) harmonic index of the partial with respect to the estimated pitch.
d reaches its maximum value for partials that fall in the middle of two successive harmonics and decreases for partials located closer to either the next or the previous harmonic rank.

Note that as in the case of *Partial Energy* (described above), the computation of inharmonicity includes all tracked partials: there may be many partials corresponding to the same harmonic rank.

Tristimulus Values: `harmonicEnergy.m`

The 1st tristimulus value is defined as the relative amplitude of the first harmonic compared to the sum of amplitudes of all harmonics:

$$\mathrm{T1} = \frac{a_1}{\sum_{h=1}^{H} a_h}$$

The 2nd tristimulus value is defined as the relative amplitude of harmonics 2–4:

$$\mathrm{T2} = \frac{a_2 + a_3 + a_4}{\sum_{h=1}^{H} a_h}$$

The 3rd tristimulus value is defined as the relative amplitude all harmonics above the 4th:

$$\mathrm{T3} = \frac{\sum_{h=5}^{H} a_h}{\sum_{h=1}^{H} a_h}$$

Temporal Energy Envelope Descriptors

These descriptors are calculated from the `TEErep`.

Note that these descriptors are not time-varying and the exported summary statistics (under the label `Value`) are the same as those time-series.

Attack Time and Log-Attack Time: `attackTime.m`

By default, attack time is defined as the time it takes the waveform to reach its maximum level. However, a threshold other than the waveform's maximum value (e.g., –6 dBFS) may be specified (see the section Signal Representations), where FS = full scale digital amplitude representation.

The estimation is based on the squared values of the (rectified) waveform.

This descriptor will also compute *log_{10}-attack time*.

Attack Slope: `attackSlope.m`

Attack slope measures the rate of increase of the signal energy during the attack time.

The estimation is based on the squared values of the (rectified) waveform.

The slope is computed as the average of a set of local slopes computed at different energy levels. By default, the first slope is computed at –48 dBFS and the rest are computed by 6 dB increments up to the threshold that is used to estimate the attack time.
These settings can be easily changed inside the function `attackSlope.m`

Decrease Slope: `decreaseSlope.m`

Decrease Slope measures the rate of decrease of the signal energy during the sustained part of the sound.

The estimation is based on the squared values of the (rectified) waveform.

The sustain segment extends from the maximum value of the waveform until the last sample that is –12dB relative to the maximum value. This threshold (`decreaseSlopeTHD`) can be set to a different value (see the section Signal Representations).

Temporal Centroid: `temporalCentroid.m`

The temporal centroid represents the center of gravity of the energy envelope, the time point that has half of the energy on either side of it over the event duration. In general, low values indicate a percussive sound and high values a sustained sound. However, the temporal centroid of the attack has also recently been shown to be causally related to a spectrotemporal dimension in timbre spaces:

Kazazis, S., Depalle, P. & McAdams, S. (2021). Ordinal scaling of temporal audio descriptors and perceptual significance of attack temporal centroid in timbre spaces. *Journal of the Acoustical Society of America*, 150(5), 3461-3473

It is defined as:

$$\mathrm{tc} = \frac{\sum_{n=n1}^{n=n2} n \cdot e(n)}{f_s \cdot \sum_{n=n1}^{n=n2} e(n)} \quad ,$$

where *n1* and *n2* are the first and last samples of the energy envelope *e(n)* that are above the noise threshold (`noiseTHD`, see also the subsection Temporal Energy Envelope (TEErep)), and f_s is the sampling frequency.

Effective Duration: `effectiveDuration.m`

The effective duration is a crude approximation of the perceived duration of the sound.

It is estimated by the amount of time the energy envelope is above a given threshold (`effectiveDurationTHD`). By default, this threshold is set to –20 dBFS. It is computed as:

$$\mathrm{efdur} = (n_2 - n_1)/f_s \quad ,$$

where *n1* and *n2* are the first and last samples of the energy envelope that are above a given threshold (`effectiveDurationTHD`) and f_s is the sampling frequency.

Energy Modulation: `energyModulation.m`

This function computes the frequency (`FrequencyOfEnergyModulation`) and amplitude (`AmplitudeOfEnergyModulation`) of the energy modulation.

These descriptors roughly correspond to a tremolo model. They represent the modulation of energy over time using a sinusoidal component computed from the Discrete Fourier Transform (DFT).

The DFT is performed on a segment of the signal. The segment begins at the point in which the first local maximum of the energy envelope occurs. The local maximum should also be above a given threshold: by default, this threshold is set to the value used to compute the *effective duration* (`effectiveDurationTHD`), but it can be easily changed inside the function `energyModulation.m`. The segment ends at the point up to *decrease slope* is computed. This is done in order to avoid the transient region and decay parts of the sound. The maximum peak of the DFT in the 1 to 20 Hz range is then estimated and used as an estimate of the modulation amplitude and frequency.

By default, if the segment is shorter than 100 ms, the modulation amplitude and frequency are set to zero. This setting (`minSustainDuration`) can be easily changed inside the function `energyModulation.m`.

Audio Signal Descriptors

These descriptors are calculated from the `ASrep`.

Zero-Crossing Rate: `zeroCrossingRate.m`

The zero-crossing rate is defined as the number of times the value of the signal *s(n)* crosses the zero level. Low values indicate a periodic sound whereas high values indicate a noisy sound.

It is computed as:

$$\mathrm{zcr} = \frac{1}{2 \cdot L_n} \sum_{n=n1}^{n=n2} |\mathrm{sgn}[s(n)] - \mathrm{sgn}[s(n-1)]| \quad ,$$

where L_n is the frame length in samples, and *n1* and *n2* are the first and last frame samples of each frame. The (local) DC offset of each frame is first removed before performing the above computation.

Autocorrelation Coefficients: `autocorrelationCoefficients.m`

The auto correlations coefficients *c* represent the spectral distribution of the signal in the time domain. They are computed according to:

$$\mathrm{xcorr}(c) = \frac{1}{\mathrm{xcorr}(0)} \sum_{n=0}^{L_n - c - 1} s(n)s(n+c) \quad ,$$

where *s* and L_n denote the signal and the frame length in samples, respectively. Only the first 20 coefficients are returned. Ideally, all signals should have the same sampling rate because this descriptor depends on sampling rate.

Frame Energy: `frameEnergy.m`

The frame energy is computed as the sum of squared sample values.

Root Mean Square (RMS) Energy: `rmsEnergy.m`

The RMS energy is computed as the root mean of *frame energy* of the signal *s*:

$$\mathrm{RMS} = \sqrt{\frac{1}{L_n}\sum_{n=n1}^{n=n2} s(n)^2} \quad ,$$

where L_n is the frame length in samples, and *n1* and *n2* are the first and last frame samples of each frame.

Output Format

The analysis results are automatically stored as `.mat` files in a user-specified location. These include the *time-series* of descriptor values and the respective *summary statistics*.

The default folder in which results are stored is named "`Results`" and is located inside the Timbre Toolbox's `Documentation` folder.

Time-series

The results for each representation are stored in different sub-folders. These sub-folders have the same name as the specified representation.

Each `.mat` file has the same name with its corresponding audio file.

The results are displayed in MATLAB's `table` format.

For audio representations that compute time-varying descriptors, the first column is labeled `TimeStamps` and corresponds to the times at which descriptors are computed (i.e., the middle of the analysis window). The only exception is the `TEErep` which does not compute time-varying descriptors. The other columns display the descriptor values. `HARMrep` also returns the estimated pitch (in the `Pitch` column) along the requested audio descriptors.

Summary Statistics

The summary statistics are saved in a single `struct`: `summaryStatistics.mat`

The fields of `summaryStatistics` have the same name as the requested representation. The sub-fields of each representation correspond to the requested summary statistics. Since TEErep has no time-varying descriptors, there is only one sub-field, which is called `Value`.

All summary statistics are displayed in `table` format: the first column displays the name of the audio file; the other columns display the values of the requested descriptors.

The summary statistics can also be exported in `.csv` format either from the GUI or from the command line by typing:

```
>> sumStats_to_CSV(resultsDirectory)
```

The argument `resultsDirectory` is the user-specified folder's path in which results will be stored. Each .csv file corresponds to the requested representation.

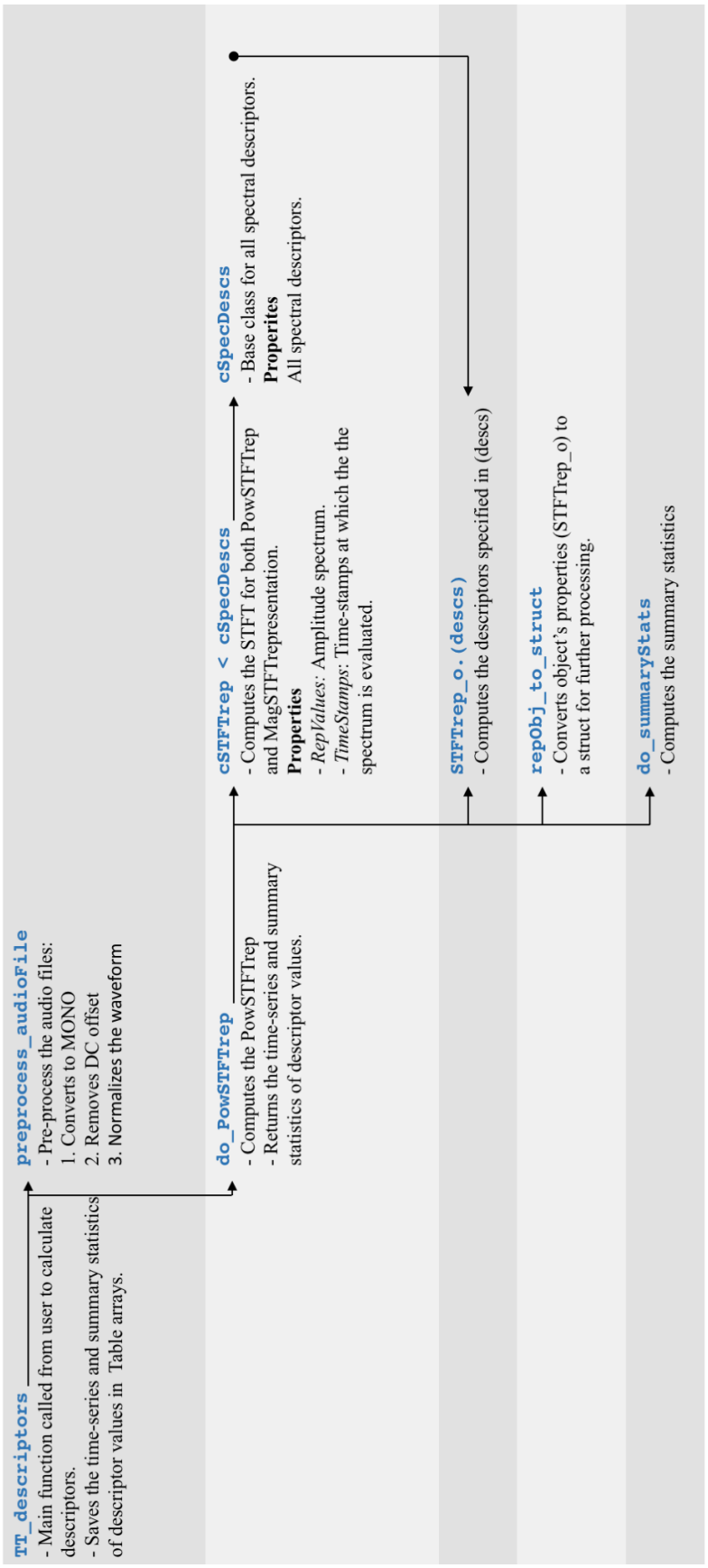
On Programming

The current version of The Timbre Toolbox uses object-oriented programming.

The main function is `TT_descriptors.m`, which first calls `cTTconfig.m` to validate the user input. The subsequent steps are shown in the first of the two following diagrams.

In the next step, `TT_descriptors.m`, calls the function `preprocess_audioFile.m` and then computes an input representation. The second diagram illustrates the computation `PowSTFTrep` which is computed from the `do_PowSTFTrep.m`. All the other audio representations have a similar workflow.


TT_descriptors
- Main function called from user to calculate descriptors.
- Saves the time-series and summary statistics of descriptor values in Table arrays.
cTTconfig
- Validates user input through static functions.
Properties
- Representations: Which representations to compute and configuration settings.
- SummaryStatistics: Which summary statistics to compute.
- SoundFiles: A list of audio files to process.
get_Reps
- Validates user input of representations and configuration settings that are used to compute them.
get_default_STFTconfig
- Default configuration settings for the STFT representation.
check_STFTconfig
- Checks user-supplied configuration settings.
get_Descs
- Validates user supplied audio descriptors.
get_default_STFTdescs
- Loads all descriptors than can be computed from the STFT representation.
check_STFTdescs
- Removes any descriptors that are not supplied from the user.
get_invalid_STFTdescs
- Outputs a warning in case of user-supplied invalid descriptors.
get_soundFiles
- Checks if input folder exists and returns a list of audio files to process.
get_Stats
- Which summary statistics to compute.
gen_resultsSubfolders
- Checks whether the folder to store results exists.
- Generates sub-folders to store time-series of descriptor values.


TT_descriptors
- Main function called from user to calculate descriptors.
- Saves the time-series and summary statistics of descriptor values in Table arrays.
preprocess_audioFile
- Pre-process the audio files:
1. Converts to MONO
2. Removes DC offset
3. Normalizes the waveform
do_PowSTFTrep
- Computes the PowSTFTrep
- Returns the time-series and summary statistics of descriptor values.
cSTFTrep < cSpecDescs
- Computes the STFT for both PowSTFTrep and MagSTFTrepresentation.
Properties
- RepValues: Amplitude spectrum.
- TimeStamps: Time-stamps at which the the spectrum is evaluated.
cSpecDescs
- Base class for all spectral descriptors.
Properites
All spectral descriptors.
STFTrep_o.(descs)
- Computes the descriptors specified in (descs)
repObj_to_struct
- Converts object's properties (STFTrep_o) to a struct for further processing.
do_summaryStats
- Computes the summary statistics