MEGA

Project number: IST-1999-20410

GEM EyesWeb Audio Tools

software contributions to the MEGA SE

SoundAnalysis Library, v. 3.0

http://www.generalmusic.com

Contact:
Carlo Drioli
Amalia de Gotzen

drioli@dei.unipd.it
corvo@dei.unipd.it

SoundAnalysis Library

The SoundAnalysis library is intended to add audio analysis features to the EyesWeb platform. The library is contained in the files EywSoundAnalysis.dll and EywSoundAnalysisDatatype.dll, and is based on the Intel SPLib and RecLib libraries which have to be installed on the system. Some of the features provided by the library are FFT and IFFT operators, frequency-domain cue extraction, pitch detection, mel-frequency cepstral analysis and spectral smoothing. The blocks provided by the SoundAnalysis library are contained in the Sound.Analysis folder in the EyesWeb block browser.

1.Frequency analysis

Time-frequency analysis is based on the Fast Fourier Transform (FFT) performed on short frames (or windows) of the input signal. It is therefore called Short Time Fourier Transform (STFT). To perform real-time STFT, a data type and a block for FFT computation have been implemented. The inverse FFT operator (IFFT) has been implemented as well.

1.1.FFT data type

Frequency analysis requires the definition of a specific data type. The structure proposed is shown in Fig. 2.1. More details on the implementation can be found in Appendix A. The data type is identified within EyesWeb as IDT1DFftBuffer .

AudioBufferSize	N. of channels	Sampling Freq	Hop Size	FFT Size
FFT Complex Data				

Fig 1.1. FFT data type

Description:

AudioBufferSize: the length (in samples) of the original input audio buffer.

FFT Size: the length (in bin) of the FFT. Common values are 256, 512, 1024, 2048, 4096. If the input audio buffer is smaller than the FFT Size, the input signal is zero-padded.

Hop Size: the amount of input frames overlapping – not yet implemented (i.e., is HopSize==AudioBufferSize).

FFT Complex Data: a complex array of length FFT Size, in which the result of the FFT is stored.

1.2.FFT and IFFT operators

The FFT analysis is implemented as an EyesWeb block (**1DFFT**) which accept an audio buffer as input (IDTSoundBuffer), and outputs an FFT data type (IDT1DFftBuffer) . The FFT block allows to select an FFT Size (256, 512, 1024, 2048, 4096, 8192, 16384). It is up to the user to select an opportune length for the input audio buffer (if the audio buffer length exceeds the FFT Size, an initialisation error is generated; if the FFT Size exceeds the audio buffer length, the input signal is zero-padded). The output of the FFT block can be displayed as a *spectrogram* with the *SpectrumViewer* block in the *Sound.Operations* folder, or can be used to compute frequency-domain features. The inverse Fourier transform (IFFT) is implemented as a block (**1DIFFT**) which accept an FFT data type (IDT1DFftBuffer) as input and outputs an (IDTSoundBuffer).

2.Frequency-domain cues

The algorithms for the representation of STFT data and the extraction of spectral cues are all collected in the EyesWeb block called **FFTCues**. The block is provided with a combo box that let the user choose among the following options: **power spectrum**, **rms level, centroid, MFCCs**, **MFCC spectrum envelope**.

2.1.FFT display

The result of the FFT operation described in the previous section is a complex vector X which represents the two-sided spectrum of the input signal. Since the spectrum of real signal is symmetrical around DC, usually the second half of the spectrum is discarded. In order to keep the energy of the whole spectrum, this conversion is performed by taking

$$S(i) = 2 \cdot S_{TS}(i), \quad i = 1,..,N/2$$

[illegible]
Hz (linear scale) or in *mel* (logarithmic scale). The *Hz* to *mel* conversion is performed with the following formula:

$$mel(f) = 2595 \cdot \log_{10}(1 + f/700) .$$

2.2.RMS and Centroid from STFT spectrum

The intensity level (RMS) and the spectral centroid (which measures the position in *Hz* of the center of mass of the spectrum) are computed respectively as:

$$RMS = \sqrt{\frac{1}{N}\sum_{k=1}^{N} |X(k)|^2}$$

$$Centr = \frac{\sum_{k=0}^{N/2} f_k \,|\, X(k) \,|}{\sum_{k=0}^{N/2} |\, X(k) \,|}$$

where *N* is the FFT size, *X(k)*, *k=0..N* is the FFT of the input signal *x*, and *fk, k=1..N,* is the *k*-th frequency bin. In both cases, the output of the block is a scalar (double) value.

2.3.Mel-Frequency Cepstral Coefficients (MFCC)

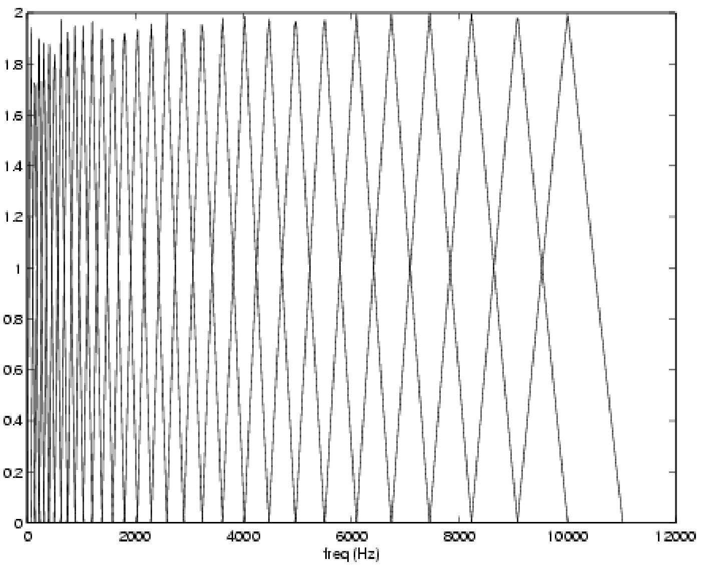
The mel-frequency cepstrum analysis is a perceptual-based sound analysis framework. It is widely accepted as a standard in the speech technology field for a number of challenging tasks, including speech recognition and speaker identification. The main feature of the mel-cepstrum analysis is to represent the frequency content of the signal with a small set of coefficients (MFCCs). The MFCCs ae computed as the discrete-cosine transform (DCT) of the log-energy output of an equally mel-spaced filterbank (see Fig.).

The Mel-frequncy cepstral coefficients, the input sound buffer is Fourier-transformed and filtered in the frequency domain with a set of overlapping triangular filters equally-spaced on a mel-frequency axis (see Fig. 2.3.1):

$$c_k = \sum_{i=1}^{N} Y_k \cos[k(n-1/2)\pi / N], \quad k = 0,..K$$

where Y_i are the log-energy outputs of the filterbank, *N* is the FFT length, and *K* is the number of band-pass filters. The first coefficient c_0 is the energy of the signal and can be discarded for energy normalization purposes.

Fig. 2.3.1: frequency response of an ideal mel-frequency equally-spaced filterbank.



The output of the filterbank (i.e., the mel-warped spectrum) can be recovered from the MFCCs by IDCT. Usually, a smoothed version of the spectral envelope is preferred and can be easily obtained by inverse transforming only the first few coefficients (see Fig. . 2.3.2).

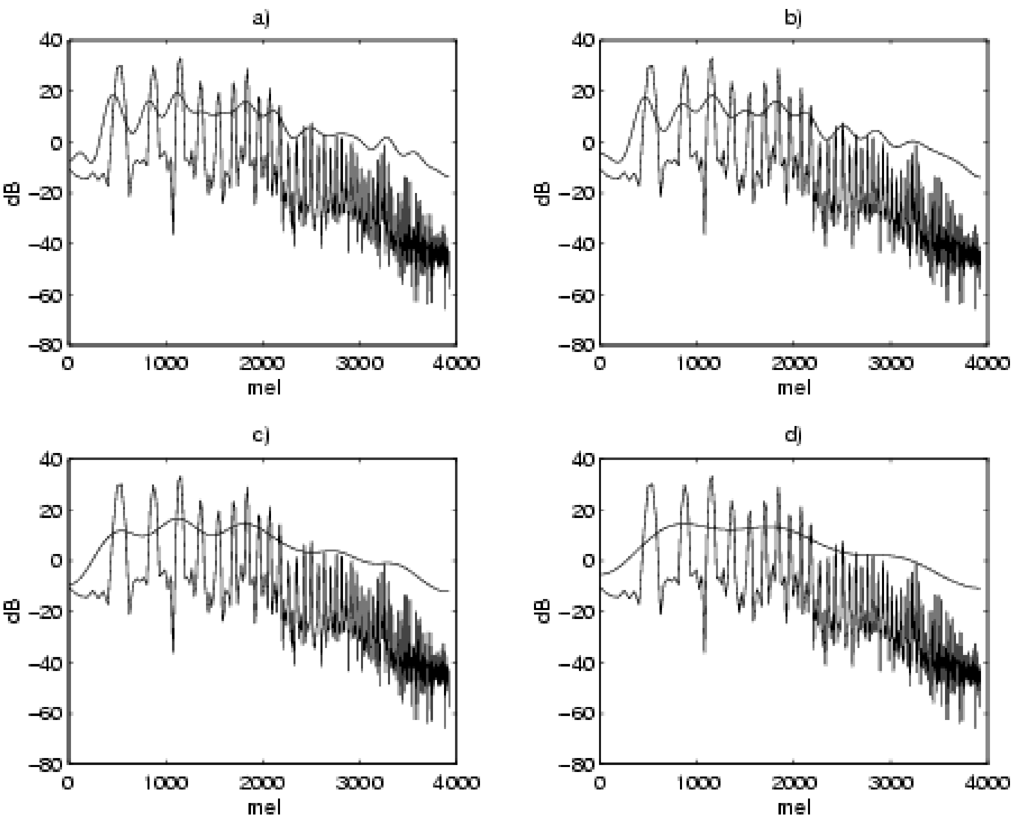


Fig. 2.3.2: Spectral envelope smoothing based on the MFC coefficients (a) 35 filters, 35 coeffs.; b) 35 filters, 30 coeffs; c) 35 filters, 15 coeffs; d) 35 filters, 8 coeffs).

The block **FFTCues** offers the possibility to get as output either the set of MFCCs, or the smoothed and warped spectrum for display purposes. In the last case, the output is a 2-column matrix, the first column being the mel frequency axis and the second column being the spectral envelope. Further parameters that can be set for this cue extraction are the number of filters in the filterbank and the number of coefficients to be used in the IDCT for the spectrum envelope computation, with number of coeffs. not exceeding the number of filters.

3.Pitch detection (autocorrelation)

The block **PitchEst_Corr** estimates the pitch of the input sound by means of an autocorrelation-based algorithm (time-domain approach). A moving average optional filter is provided to smooth the output of the pitch tracking algorithm.

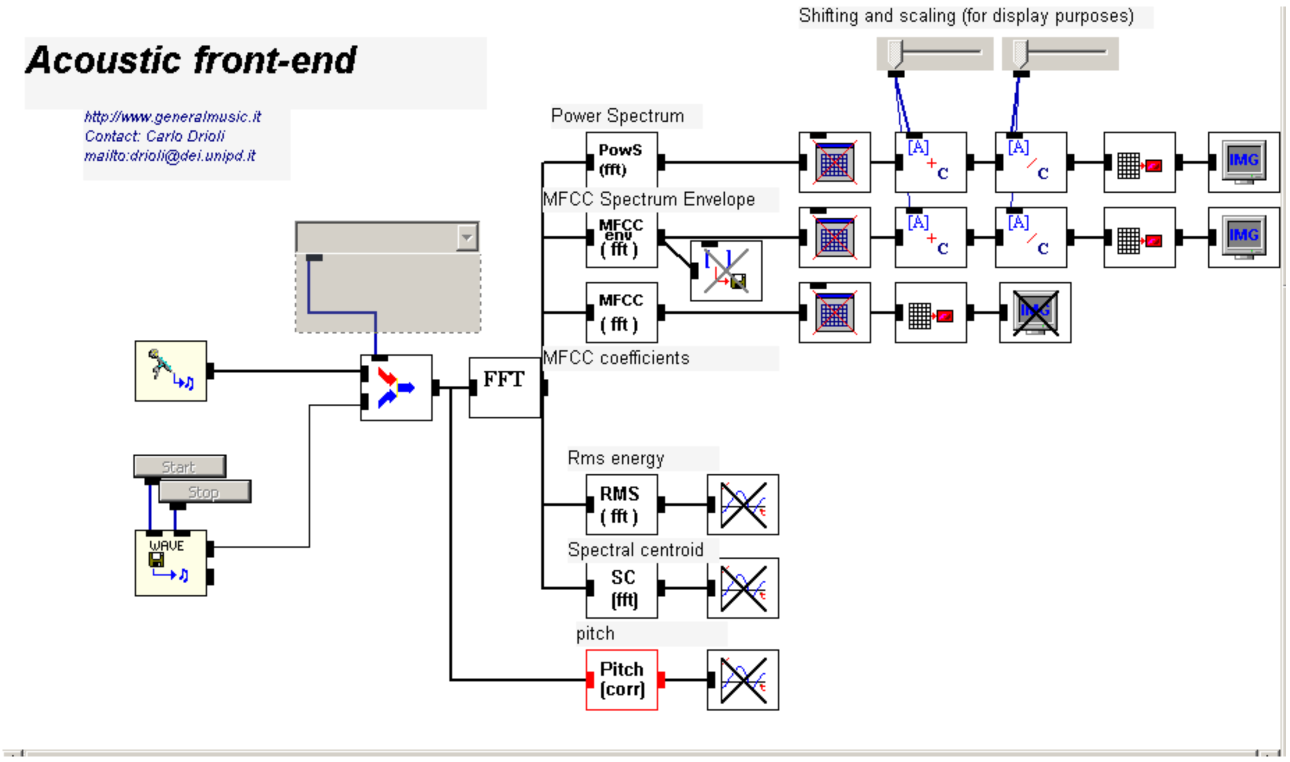


Fig. 3.1: Example of an acoustic front-end realized with the sound analysis blocks described.

4.Vibrato (Vibrato Through hand/arm gesture analysis)

This block allows a performer to control the vibrato feature of an audio source through the analysis of hand/arm gestures. It processes the numerical description (provided by a "MotionAnalysis.Cues.ContractionIndex" block) of the live image of a waving hand and as a result it generates a sinusoidal waveform, continuously changing in amplitude and frequency, that acts as a relative (normalized) pitch reference for the audio source. The parameters that describe the mapping between gesture data and output waveform are widely configurable. Figure 4 shown the block used in a patch.

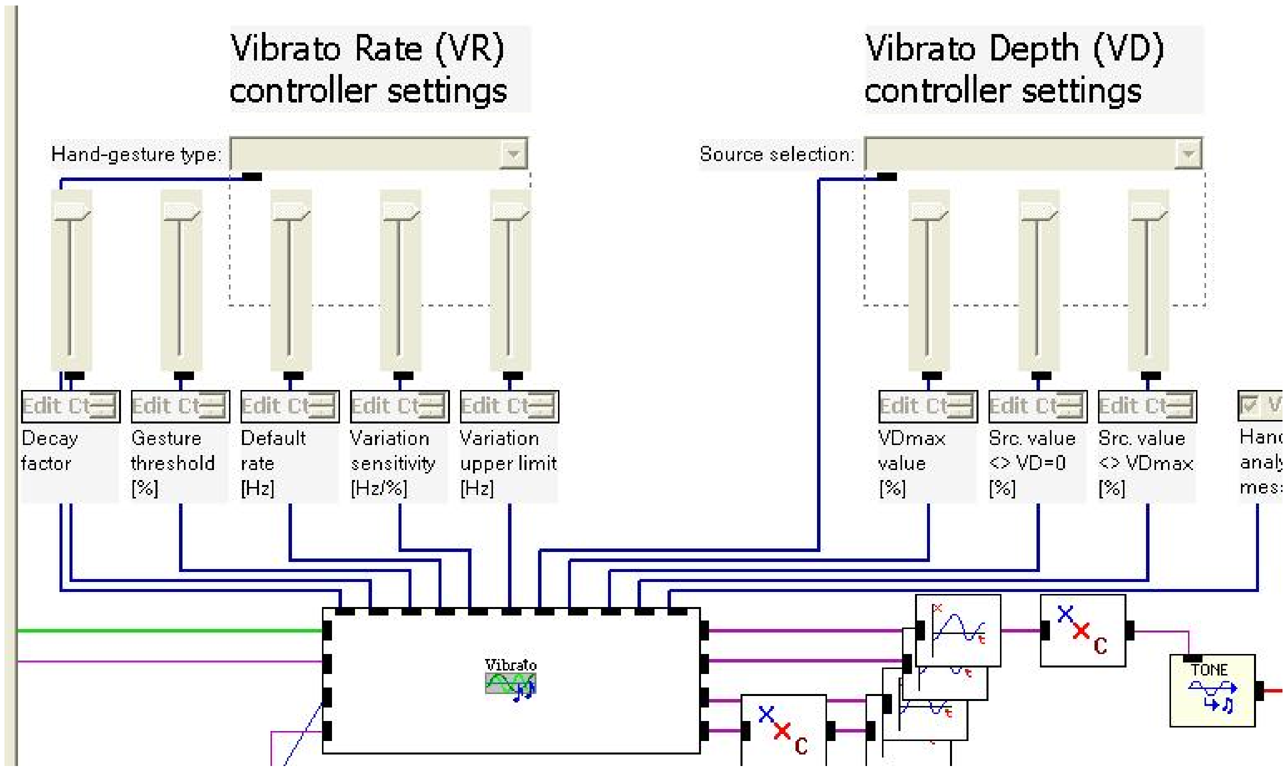


Figure 4: Vibrato block and his controller parameters.

Appendix A: 1-D FFT Data Type specification

Requirements:

nsp.h Intel DSP library (defines double-precision complex numbers DCplx)

Datatype

IDT1DFftBuffer: inherits from *IEyesWebDatatypes*
CLSID: ...
IID: ...

```
HRESULT GetFftBuffer(DWORD dwNumChannel, DCplx** ppdFftBuffer);
HRESULT GetNumChannels(DWORD* pdwNumChannels);
HRESULT GetSamplingFrequency(DWORD* pdwSamplingFrequency);
HRESULT GetFftSize(DWORD* pdwFftSize);
HRESULT GetFftOrder(DWORD* pdwFftOrder);
HRESULT GetHopSize(DWORD* pdwHopSize);
HRESULT GetAudioBufferSize(DWORD* pdwAudioBufferSize);
```

Initialization Data

```
typedef struct
{
        DWORD dwAudioBufferSize;
        DWORD dwNumChannels;
        DWORD dwSamplingFrequency;
        DWORD dwHopSize;
        DWORD dwFftSize;
        DWORD dwFftOrder;
}EYW_1DFFTBUFFER_INIT_DATA;
```

where:

- dwAudioBufferSize is the input audio buffer length
- dwNumChannels is the umber of channels
- dwSamplingFrequency is the sampling frequency (inHz)
- dwFftSize is the number of points in the FFT
- dwFftOrder is the power of two that gives the Fft size.
- dwHopSize is the window shift size (in msec)

Notes: information on the window type used for analysis should be added to the Data Type structure