9

Adaptive digital audio effects

V. Verfaille, D. Arfib, F. Keiler, A. von dem Knesebeck and U. Zölzer

9.1 Introduction

The idea of controlling a sound transformation according to the musical scene it is displayed in may be as old as the idea of composition. The particular case of adaptive DAFX is when the sound to be transformed is also used as the source of the modification control parameter(s): some information about the sound is then collected and used to modify the sound with a coherent evolution. With such a perspective, adaptive DAFX can be considered as an extension of, or inspired by, composition processes similar to counterpoint. This process can also be compared to what happens in natural physical phenomena, such as the harmonic enrichment happening in brass instruments when the sound level increases. This chapter presents and formalizes the use of adaptive control in order to build new digital audio effects. It presents a general framework that encompasses existing DAFX and opens doors to the development of new and refined DAFX. It is aimed at presenting the adaptive control of effects, describing the steps to build an adaptive effect and explaining the necessary modifications of signal-processing techniques.

Adaptive DAFX can be developed and used with various goals in mind: to provide a high-level control for usual DAFX, to propose new production and/or creative tools by offering new DAFX, or to strengthen the relationship between Fx, control and perception. The control and mapping aspects of adaptive DAFX are important matters. In any DAFX, the control is sometimes part of the effect (e.g., compressor), sometimes part of the mapping (e.g., pitch-shift amount). Also, the gestural control mapping of DAFX is unfortunately often limited to a one-to-one mapping (one gestural parameter is directly mapped to one effect parameter), with the noticeable exception of the GRM Tools [Fav01, INA03], which allow for linear interpolation between presets of N parameters. In order to define a general framework we will look at some famous simple examples of adaptive DAFX. As explained in Section 4.2 the compressor (see Figure 9.1) transforms the input sound

DAFX: Digital Audio Effects, Second Edition. Edited by Udo Zölzer.
 ISBN: 978-0-470-66599-2

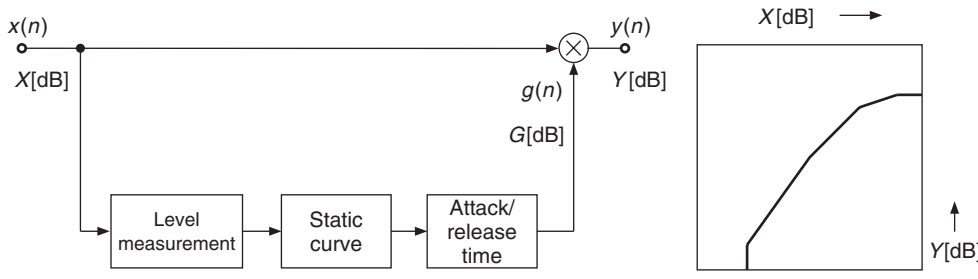

Figure 9.1 Compressor/expander with input/output level relationship.

level to the output sound level according to a gain control signal. The input sound level is mapped depending on a non-linear curve into the control signal with an attack and release time (inducing some hysteresis).

The principle of auto-tuning a musical sound consists in applying a time-varying transposition (or pitch shift), which is controlled by the fundamental frequency of the input signal so that the pitch becomes tuned to a given musical scale (see Figure 9.2). It is also called "pitch discretization to temperate scale" (see Chapter 10), even though it would work for any non-temperature scale too. The auto-tune effect modifies the pitch by applying a f_0-dependent pitch shift, as depicted in Figure 9.2. The output fundamental frequency depends on: (i) the input fundamental frequency, (ii) a non-linear mapping law that applies a discretization of fundamental frequency to the tempered scale (see Figure 9.3), and (iii) a musical scale that may change from time to time depending on the music signature and chord changes.

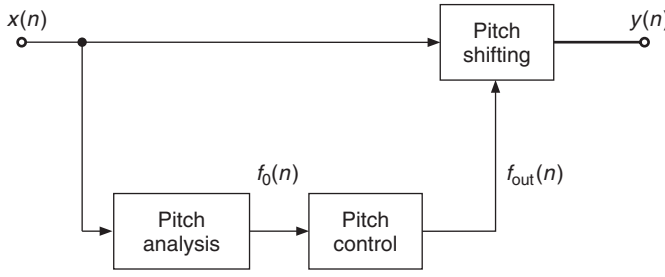

Figure 9.2 Auto-tune.

Some adaptive DAFX have existed and been in use for a long time. They were designed and finely crafted with a specific musical or music production goal. Their properties and design are now used to generalize the adaption principle, allowing one to define several forms of adaptive DAFX and to perform a more systematic investigation of sound-feature control. When extracting sound features, one performs some kind of indirect acquisition of the physical gesture used to produce the sound. Then, the musical gesture conveyed by the sound used to extract features is implicitly used to shape the sound transformation with meaning, because '*the intelligence is* (already) *in the sound signal.*' Such generalizing approach has been helpful to finding new and creative DAFX.

Definition of adaptive DAFX

Adaptive DAFX are made by combining an audio effect with an adaptive control stage. More specifically, adaptive DAFX are defined as audio effects with a time-varying control derived from

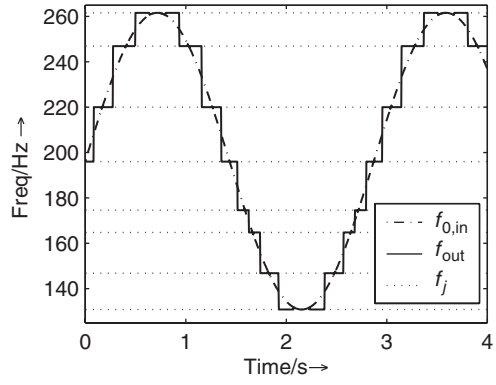

Figure 9.3 Fundamental frequency of an auto-tuned FM sound

sound features[1] transformed into valid control parameter values using specific mapping functions [VA01], as depicted in Figure 9.4. This definition generalizes previous observations of existing adaptive effects (compressor, auto-tune, cross-synthesis), but is also inspired by the combination of an amplitude/pitch follower with a voltage-controlled oscillator [Moo65].

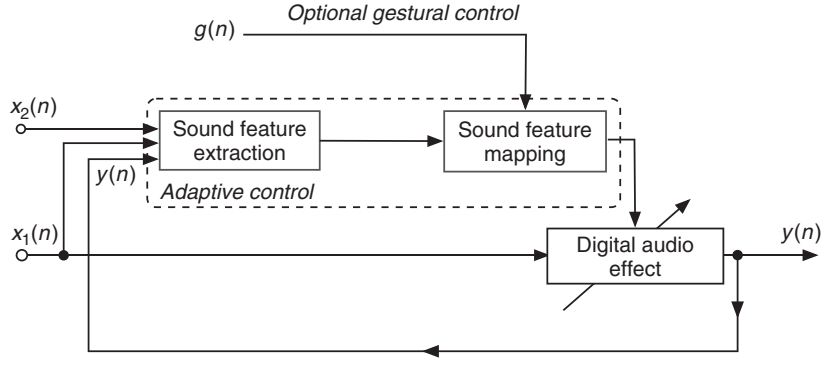

Figure 9.4 Diagram of the adaptive effect. Sound features are extracted from an input signal $x_1(n)$ or $x_2(n)$, or from the output signal $y(n)$. The mapping between sound features and the control parameters of the effect is modified by a gestural control. Figure reprinted with IEEE permission from [VZA06].

Adaptive DAFX are called differently according to the authors, for instance 'dynamic processing' (even though 'dynamic' will mean 'adaptive' for some people, or more generally 'time-varying' without adaptation for other people), 'intelligent effects' [Arf98], 'content-based transformations' [ABL+03], etc. The corresponding acronym to adaptive DAFX is ADAFX. With ADAFX, it is to be noted that the effect name can be non-specific to the sound descriptor (for instance: cross-synthesis, adaptive spectral panning, adaptive time-scaling), but it also can have a control-based name (for instance: auto-tune, compressor). Such names are based on the perceptual effect

[1] Recent literature tends to use 'sound descriptors' instead of 'sound features'; both describe the same reality, that is to say parameters that describe the sound signal, and/or some of its statistical, acoustical or perceptual properties.

produced by the signal-processing technique. Generally speaking, ADAFX don't have their control limited to the adaptive part, but rather it is two-fold:

- Adaptive control derived from sound features;
- Gestural control – used for real-time access through input devices.

The mapping laws to transform sound features into DAFX control parameters make use of non-linear warping curves, possibly hysteresis, as well as feature combination (see Section 9.3).

The various forms ADAFX may take depend on the input signal used for feature extraction:

- **Auto-adaptive** effects have their features extracted from the input signal $x_1(n)$. Common examples are compressor/expander, and auto-tune.
- **Adaptive** or **external-adaptive** effects have their features extracted from at least one other input signal $x_2(n)$. An example is cross-ducking, where the sound level of $x_1(n)$ is lowered when the sound level of $x_2(n)$ goes above a given threshold.
- **Feedback adaptive** effects have their features extracted from the output signal $y(n)$; it follows that auto-adaptive and external-adaptive effects are feed-forward-adaptive effects.
- **Cross-adaptive** effects are a combination of at least two external-adaptive effects (not depicted in Figure 9.4); they use at least two input signals $x_1(n)$ and $x_2(n)$. Each signal is processed using the features of another signal as controls. Examples are cross-synthesis and automatic mixing (see Chapter 13).

Note that cross synthesis can be considered as both an external-adaptive effect and a cross-adaptive effect Therefore, the four forms of ADAFX do not provide a good classification of ADAFX since they are not exclusive; they do, however, provide a way to better describe, directly in the effect name, where the sound descriptor come from. The next two sections investigate sound-feature extraction (Section 9.2), and mapping strategies related to adaptive DAFX (Section 9.3).

9.2 Sound-feature extraction

9.2.1 General comments

Sound features are parameters that describe the sound. They are used in a wide variety of applications such as coding, automatic transcription, automatic score following, and analysis-synthesis. As an example, the research field of music information retrieval has been rapidly and increasingly developing during the last ten years, offering various analysis systems and many sound features. A musical sound has some perceptive features that can be extracted from a time-frequency representation. As an example, pitch is a function of time that is very important for musicians, but richness of timbre, inharmonicity, balance between odd and even harmonics, and noise level are other examples of such time-varying parameters. These parameters are global in the sense that they are observations of the sound without any analytical separation of these components, which will be discussed in Chapter 10. They are related to perceptive cues and are based on hearing and psychoacoustics. These global parameters can be extracted from time-frequency or source-filter representations using classical tools of signal processing, where psychoacoustic fundamentals also have to be taken into account. The use of these parameters for digital audio effects is twofold: one can use them inside the effect algorithm itself, or one can use these features as control variables for other effects, which is the purpose of this chapter. Pitch tracking as a source of control is a well-known application. Examples of audio effects using feature extraction inside the algorithm are the correction of tuning, which uses pitch extraction (auto-tune), or even the compression of a sound, which uses amplitude extraction.

Their computation is based on a representation of the sound (for instance a source-filter, a time-frequency or a spectral model), but not necessarily the same as the one used to apply the adaptive effect. This means that the two signal-processing blocks of sound-feature extraction and audio effect may be decorrelated, and the implementation of one may not rely on the implementation of the other, resulting in an increase in CPU use. For this reason, we will present the computation of various sound features, each with various implementations, depending on the model domain. Depending on the application, sound features may require more- or less-accurate computation. For instance, an automatic score-following system must have accurate pitch and rhythm detection. Another example is the evaluation of brightness, acoustical correlate of which is the spectral centroid. Therefore, brightness can either be computed from a psychoacoustic model of brightness [vA85, ZF99], or approximated by the spectral centroid, with an optional correction factor [Bea82], or by the zero-crossing rate or the spectral slope. In the context of adaptive control, any feature can provide good control: depending on its mapping to the effect control parameters, it may provide a transformation that sounds. This is not systematically related to the accuracy of the feature computation, since the feature is extracted and then mapped to a control. For example, a pitch model using the auto-correlation function does not always provide a good pitch estimation; this may be a problem for automatic transcription or auto-tune, but not if it is low-pass filtered and drives the frequency of a tremolo. As already explained in Chapter 1, there is a complex and subjective equation involving the sound to be processed, the audio effect, the mapping, the feature, and the will of the musician. For that reason, no restriction is given *a priori* to existing and eventually redundant features; however, perceptual features seem to be a better starting point when investigating the adaptive control of an effect.

Sound-feature classification

At least six viewpoints can be considered in order to classify the sound features or descriptors: (i) the description level, (ii) the acquisition method, (iii) the integration time, (iv) the type of feature, (v) the causality, and (vi) the computational domain (time or frequency).

Description level We consider low-level sound features as features that are related to acoustical or signal-processing parameters, whereas high-level sound features are related to sound perception and/or cognition. For instance, sound level, spectral centroid, fundamental frequency, and signal-to-noise ratio are signal/acoustical properties, whereas (perceived) loudness, brightness, pitch, and noisiness are perceptual properties of the sound. Those two categories can be further separated as follows:

- Low level: signal level itself, acoustical properties, without the need for a signal model
  - direct computation, e.g., amplitude by RMS, spectral centroid
  - indirect computation, e.g., using complex computation and sound models (partials' amplitude and frequency)
- High level: perceptual attributes and cognitive descriptions
  - perceptually relevant parameters, but not psychoacoustical models (e.g., jitter and shimmer of harmonics, related to timbral properties)
  - perceptual attributes: pitch, loudness, timbral attributes (brightness, noisiness, roughness), spatial cues, etc.
  - cognitive parameters and expressiveness-related parameters: harmony, vibrato, etc.

There are some relationships between low-level features and perceptual features, as some perceptual features are approximated by some low-level sound features. A non-exhaustive set of sound features [VZA06] shown in Figure 9.5 will contain various sound features that are commonly used for

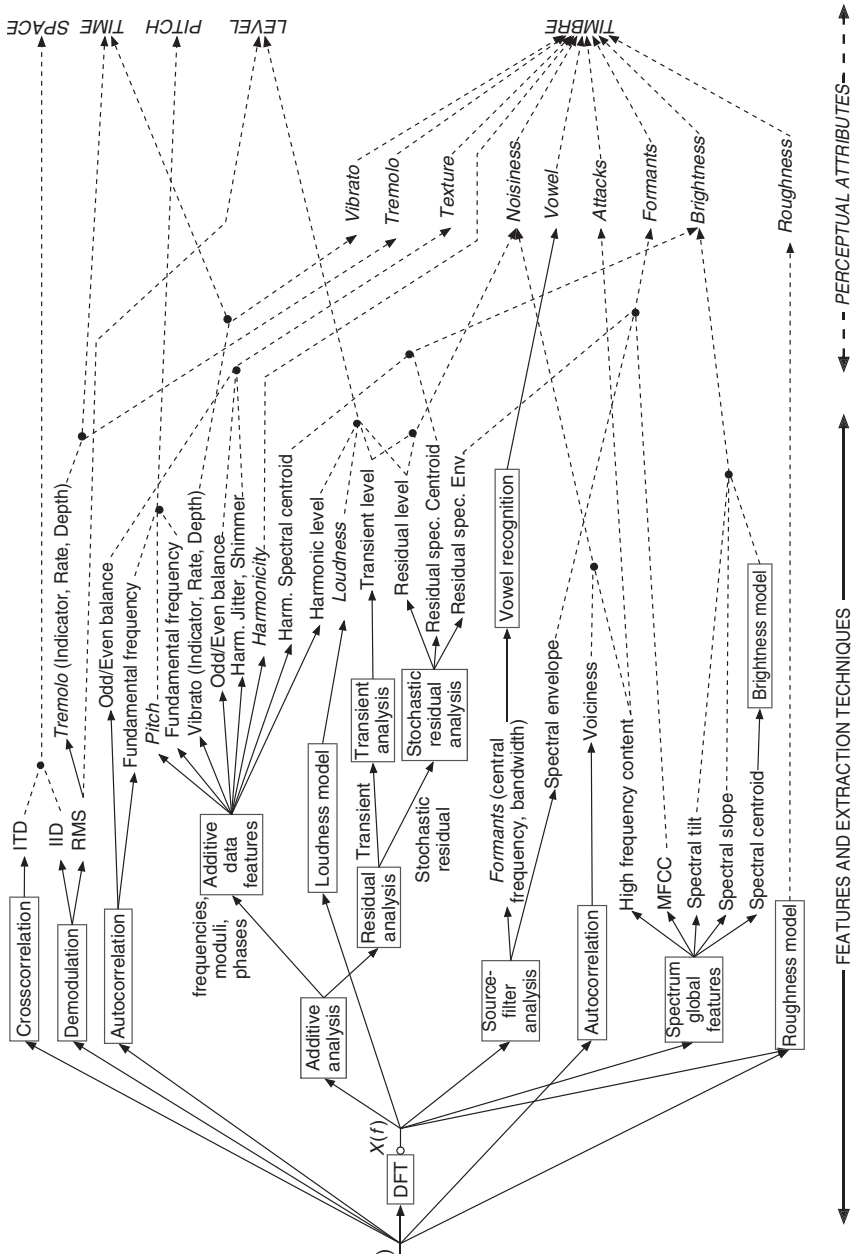

Figure 9.5 Example set of sound features that can be used as control parameters in adaptive effects [VZA06]. The arrow line types indicate the techniques used for extraction (left and plain lines) and the related perceptual attribute (right and dashed lines). Italic words refer to perceptual attributes. Figure reprinted with IEEE permission from [VZA06].

timbre space description (for instance based on MPEG-7 proposals [PMH00]) as well as features computed by perceptual models such as the ones extracted by the PsySound software [Cab99] in an offline (non-real-time) context: pitch, loudness, brightness, roughness, etc. Therefore, the wider the sound-feature set, the more chances we have to grasp the sound information along all perceptual attributes, with a consequent increase in information redundancy.

Acquisition method Various acquisition and computation methods can be used. A sound feature $f(m)$ can be obtained directly, as is the sound amplitude by RMS. It can also be obtained indirectly, as the derivative:

$$\mathrm{der}_1(m) = f_S \frac{f(m) - f(m-1)}{R_a} \tag{9.1}$$

$$\mathrm{der}_2(m) = f_S \frac{f(m+1) - f(m)}{R_a}, \tag{9.2}$$

with R_a the analysis increment step, f_S the sampling rate. While instantaneous sound features provide information about acoustical or perceptual features, their derivatives inform about the evolution of such features. It can also be obtained indirectly as the integration, the absolute value, the mean of the standard deviation of another parameter. In this latter case, it can be seen as obtained through a given mapping. However, as soon as this new feature has a descriptive meaning, it can also be considered as a new feature.

Integration time Each parameter is computed and has a meaning only for a given time interval. Due to the time-frequency uncertainty (similar to the Heisenberg principle in quantum mechanics), no sound feature can be considered as really instantaneous, but rather is obtained from a certain number of sound samples, and is then quasi-instantaneous. This remark can also be expanded to samples themselves, as they represent the sound wave of a given time interval with a single value. We will then consider that *quasi-instantaneous* features correspond to parameters extracted from a 512 to 2048 samples of a signal sampled at 44.1 kHz, representing 12–46 ms. *Mid-term* sound features are features derived from the mean, the standard variance, the skewness or the kurtosis (i.e., the four first-order statistical moments) of N values of a parameter using a sliding window, or the beat, loudness. *Long-term* sound features are descriptors based on the signal description, and, for instance, relate to notes, presence/absence of acoustical effects such as vibrato, tremolo, flutter-tonguing. *Very long-term* sound features describe a whole musical excerpt in terms of a sound sequence as seen through music analysis: style, tonality, structure, etc.

The integration time is then related to sound segmentation and region attributes, on which sound feature statistics (mean, standard deviation) are useful. While sound features such as spectral centroid and fundamental frequency can be computed as (almost) instantaneous parameters, there are also sound descriptions that require longer-term estimation and processing. For instance, some loudness perceptual models account for time integration.

Moreover, one may want to segment a note into attack/decay/sustain/release and extract specific features for each section. One may also want to segment a musical excerpt into notes and silences, and then process differently each note, or each sequence of notes, depending on some criterion. This is the case for auto-tune (even though this is somewhat done on-the-fly), swing change (adaptive time-scaling with onset modification depending on the rhythm), etc. Such adaptive segmentation processes may use both low-level and/or high-level features. Note that even in the context of real-time implementation, sound features are not really instantaneous, as they often are computed with a block-by-block approach. Therefore, we need, when describing low-level and high-level features, to indicate their instantaneous and segmental aspects as well. Sound segmentation, which is a common resource in automatic speech-recognition systems, as well as in automatic music-transcription systems, score-following systems, etc., has a central place in ADAFX. Since the late

1990s, music segmentation applications [Ros98] started to use techniques originally developed in the context of speech segmentation [VM90], such as those based on pattern recognition or knowledge-based methodologies.

Type We may consider at least two main types of sound descriptors: continuous and indicators. *Indicators* are parameters that can take a small and limited number of discrete values, for instance the presence of an acoustical effect (vibrato, tremolo, flutter-tonguing) or the identification of a sung vowel among a subset of five. Conversely, *continuous* features are features that can take any value in a given range, for instance the amplitude with RMS in the interval $[0, 1]$ or the fundamental frequency in the interval [20, 20000] Hz. An intertwined case is the *probability* or *pseudo probability* that is continuous, but can also be used as an indicator as soon as it is coupled with a threshold. The voiciness computed from autocorrelation is such a feature, as it can indicate if the sound is harmonic/voicy or not depending on a continuous value in $[0, 1]$ and a threshold around 0.8. By definition, indicators are higher-level features than continuous descriptors.

Causality A sound feature can be computed either only from past samples, in which case it is a *causal* process, or from past and future samples, in which case it is an *anti-causal* process. For instance, when considering a time frame, the sound intensity computed by RMS, using the last frame sample as the reference time of the frame will make this computation causal (it can be obtained in real-time), whereas attributing this value to the frame central time will consider this as an anti-causal process. Only causal processing can be performed in real-time (with a latency depending on both the frame size and the computational time), thus limiting the possible real-time applications to the sound features available in real-time.

Time and time-frequency domains A sound feature directly computed from the wave form uses a time-domain computation, whereas a sound feature computed from the Fourier transform of a time frame uses a frequency-domain computation. We will use this classification. While these six ways to look at sound features inform us about their properties and computation, the next sections will use the perceptual attribute described by the sound feature as a way to organize descriptions of various sound features. As proposed in Section 1.2.2 of Chapter 1, we now present sound features classified according to the perceptual attribute they relate to, namely: loudness, time, pitch, spatial hearing, and timbre.

9.2.2 Loudness-related sound features

Amplitude envelope

One very important feature that can be used for adaptive effects is the amplitude envelope of a sound evolving with time. Even the modulation of a sound by the envelope of another sound is an effect by itself. But more generally the amplitude envelope can be used to control many variables of an effect. Applications of amplitude detection can be found in dynamics processing (see Chapter 4), but can also be integrated into many effects as an external control parameter.

Except for the fact that we want to write a signal as $x(n) = \text{amp}(n) \cdot \text{sig}(n)$, there is no unique definition of an amplitude envelope of a sound. The ear is devised in such a way that slow variations of amplitude (under 10 Hz) are considered as a time envelope while more rapid variations would be heard as a sound. This distinction between an envelope and a signal is known in electroacoustic music as the difference between a 'shape' and a 'matter,' two terms well developed by P. Schaeffer in his *Traité des objets musicaux* [Sch66].

The RMS (root mean square) algorithm has been largely used in Chapter 4 as an amplitude detector based on filtering the squared input samples and taking the square root of the filter output. The RMS value is a good indication of the temporal variation of the energy of a sound, as shown in Figure 9.6. This filtering can also be performed by a FIR filter, and in this case can be inserted into

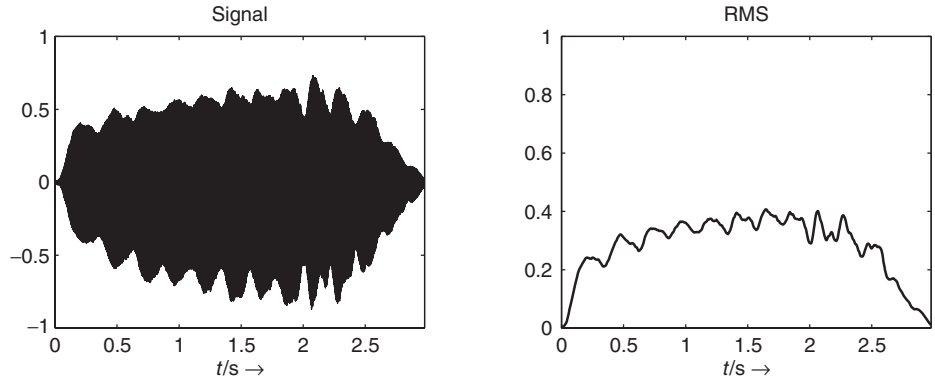

Figure 9.6 Signal and amplitude envelope (RMS value) of the signal.

an FFT/IFFT-based analysis-synthesis scheme for a digital audio effect. The FFT window can be considered a lowpass FIR filter, and one of the reasons for the crucial choice of window size for a short-time Fourier transform is found in the separation between shape and matter: if the window is too short, the envelope will follow rapid oscillations which should not be included. If the window is too large, the envelope will not take into account tremolos which should be included. The following M-file 9.1 calculates the amplitude envelope of a signal according to an RMS algorithm.

M-file 9.1 (UX_rms.m)

```
% Author: Verfaille, Arfib, Keiler, Zölzer
clear; clf
%----- USER DATA -----
[DAFx_in, FS]  = wavread('x1.wav');
hop            = 256;  % hop size between two FFTs
WLen           = 1024; % length of the windows
w              = hanningz(WLen);
%----- some initializations -----
WLen2          = WLen/2;
normW          = norm(w,2);
pft            = 1;
lf             = floor((length(DAFx_in) - WLen)/hop);
feature_rms    = zeros(lf,1);
tic
%=========================================
pin  = 0;
pend = length(DAFx_in) - WLen;

while pin<pend
   grain = DAFx_in(pin+1:pin+WLen).* w;
   feature_rms(pft) = norm(grain,2) / normW;
   pft   = pft + 1;
   pin   = pin + hop;
end
% =========================================
toc
subplot(2,2,1); plot(DAFx_in); axis([1 pend -1 1])
subplot(2,2,2); plot(feature_rms); axis([1 lf -1 1])
```

When using a spectral model, one can separate the amplitude of the sinusoidal component and the amplitude of the residual component. The amplitude of the sinusoidal component is computed for a given frame m as the sum of the amplitude for all L harmonics given by

$$a_{sin,lin}(m) = \sum_{i=1}^{L} a_i(m), \tag{9.3}$$

with $a_i(m)$ the linear amplitude of the ith harmonic. It can also be expressed in dB as

$$a_{sin,dB}(m) = 20\log_{10}\left(\sum_{i=1}^{L} a_i\right). \tag{9.4}$$

The amplitude of the residual component is the sum of the absolute values of the residual of one frame, and can also be computed according to

$$\begin{aligned} a_{res,dB} &= 20\log_{10}\left(\sum_{n=0}^{M-1} |x_R(n)|\right) \\ &= 20\log_{10}\left(\sum_{k=0}^{N-1} |X_R(k)|\right) \end{aligned} \tag{9.5}$$

where $x_R(n)$ is the residual sound, M is the block size, $X_R(k)$ is residual sound spectrum, and N the magnitude spectrum size.

Sound energy

The instantaneous sound energy is the squared amplitude. It can either be derived from the wave form by

$$e_1(n) = [x(n)]^2, \tag{9.6}$$

from a time-frequency representation for the mth block by

$$e_2(m) = \sum_{k=0}^{N-1} |X(m,k)|^2, \tag{9.7}$$

or also from a spectral representation by

$$e_3(m) = \sum_{i=1}^{L} [a_i(m)]^2. \tag{9.8}$$

Loudness

There exist various computational models of loudness, starting from Zwicker's model [ZS65, Zwi77] to various improvements by Zwicker and Fastl [ZF99], Moore and Glasberg [MG96, MGB97], most of them being implemented in the Psysound3[2] [Cab99, Cab00]. Generally speaking, these models consider a critical band in the spectrum, onto which the energy is summed after accounting for frequency masking. More improved models also account for time integration and masking. Since the loudness of a sound depends on the sound level at which it is played, various different loudness curves can be obtained as control curves for adaptive effect control.

Tremolo description

Using a sinusoidal model of the amplitude envelope, one can describe the time evolution of a tremolo in terms of rate or frequency (in Hz), amplitude or depth (in dB), and phase at the start.

[2] http://psysound.wikidot.com/

Those parameters can be used as sound features for other purposes. An example is the special sinusoidal model [MR04], when it is used for time-scaling of vibrato sounds, while preserving the vibrato attributes.

9.2.3 Time features: beat detection and tracking

A useful feature for controlling parameters of digital audio effects is the actual tempo or so-called beats-per-minute measure. In a live playing situation usually a drummer or conductor counts the tempo in by hitting his stick, which gives the first precise estimate for the beat. So the beat per minute is known from the very beginning. In a recording situation the first tracks are usually recorded with a so-called click track which is given by the recording system and everything is played in sync to that track. In both situations the beat is detected by simple means of listening and playing in sync to the beat. The beat signal can be used to control effect parameters, such as delay-time settings, speed of parameter modulation, gates for reverb and so on. A variety of beat-synced effects can be found in the Adrenalinn product. In the following, a robust and proven concept based on [BFC05, Fit04, Fit10] will be introduced which offers beat detection and tracking. The beat detection and tracking has a pre-processing stage which separates harmonic and percussive parts for further onset detection of harmonic and percussive signals. The harmonic/percussion separation is based on processing on the spectrogram, making use of median filtering across each frame in frequency direction and across each frequency bin in the time direction (see Figure 9.7).

This separation technique is based on the idea that as a first approximation, broadband noise signals such as drums can be regarded as stable vertical ridges in a spectrogram. Therefore, the

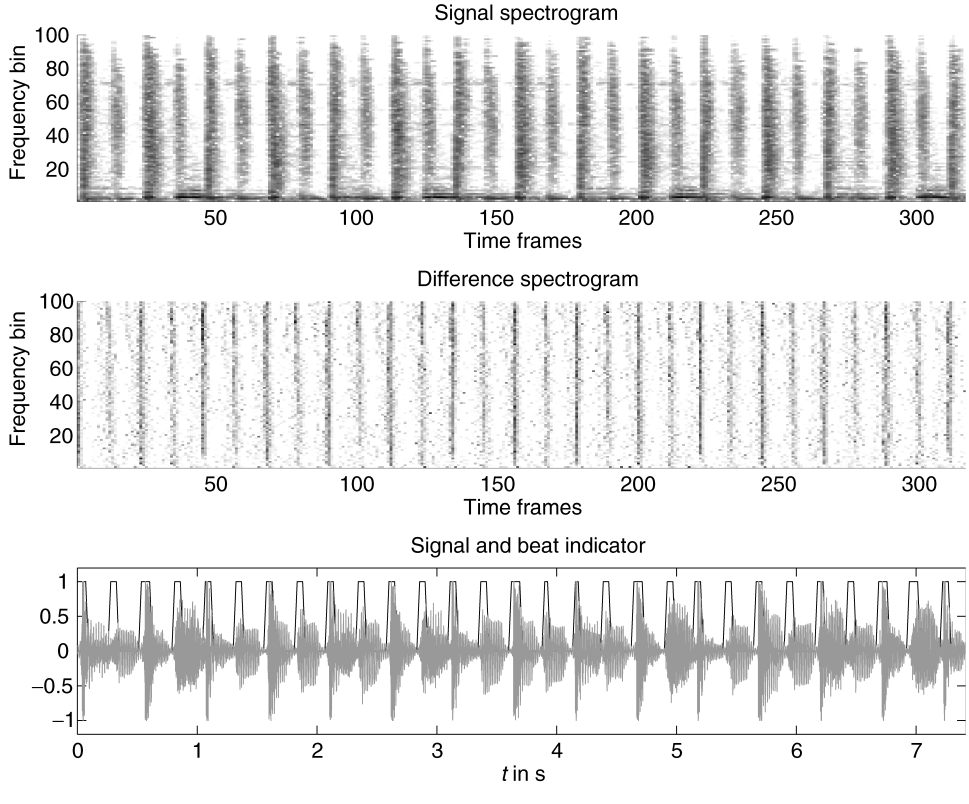

Figure 9.7 Signal spectrogram, difference spectrogram, and time-domain signal and its beat indicators.

presence of narrow band signals, such as the harmonics from a pitched instrument, will result in an increase in energy within a bin over and above that due to the drum instrument, resulting in outliers in the spectrogram frame. These can be removed by the use of a median filter, thereby suppressing the harmonic information within the spectrogram frame. The median filtered frames are stored in a percussion enhanced spectrogram, denoted $P(n, k)$. Similarly, the harmonics of pitched instruments can be regarded as stable horizontal ridges in a spectrogram. In this case, a sudden onset due to a drum or percussion instrument will result in a large increase in energy across time within a given frequency slice. This will again result in outliers in the evolution of the frequency slice with time, which can be removed by median filtering, thereby suppressing percussive events within the frequency bin. The median-filtered frequency slices are then stored in a harmonic-enhanced spectrogram $H(n, k)$. However, as median filtering is a non-linear operation which introduces artifacts into the spectrogram, it is better to use the enhanced spectrograms obtained from median filtering to create masks to be applied to the original spectrogram. This can be done using a Wiener-filtering-based approach, which delivers the harmonic spectrogram

$$\hat{H}(n, k) = X(n, k) \cdot \frac{H^p(n, k)}{H^p(n, k) + P^p(n, k)} \tag{9.9}$$

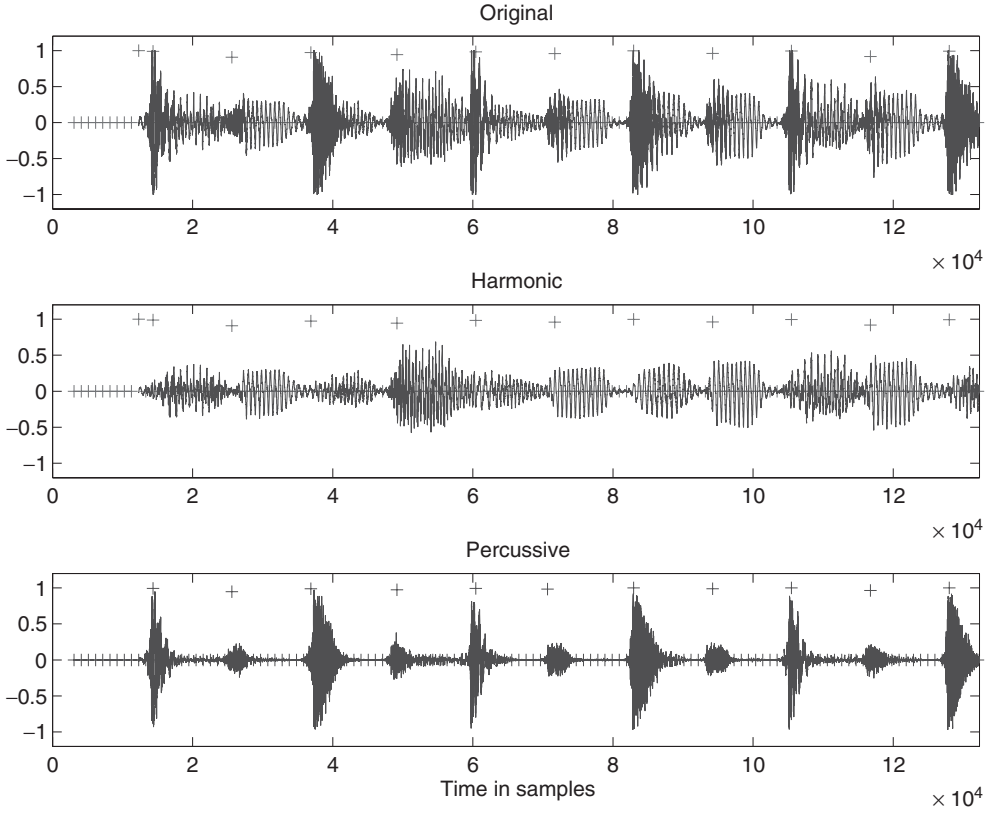

Figure 9.8 Original signal with onset markers, harmonic signal with onset markers, and percussive signal with beat markers.

and the percussive spectrogram

$$\hat{P}(n,k) = X(n,k) \cdot \frac{P^p(n,k)}{H^p(n,k) + P^p(n,k)}. \tag{9.10}$$

These spectrograms can then be inverted to the time domain by the inverse short-time Fourier transform. Beats are then detected by carrying out differentiation between successive frames $X(n,k) - X(n-1,k)$ and counting the number of frequency bins which have a positive energy change. This is a simple method for detecting the occurrence of a broadband noise signal such as a drum. As this method is not based on the amount of energy in the signal, it can be used to detect low-energy noise-based events. By setting a threshold of how many bins have to be positive before an event is considered to be detected, and by keeping only local maxima in this function, a beat detection function can be obtained. This is illustrated on both the original signal (spectrogram), as well as the separated harmonic and percussion signal (spectrogram) (see Figure 9.8). The following M-file 9.2 shows the harmonic/percussion separation. The code assumes odd length median filters with p as the power to which the separated spectrogram frames are raised when generating masks to be applied to the original spectrogram frame. The output of the percussive separation is then used to generate an onset detection function for use with beat-driven effects.

M-file 9.2 (HPseparation.m)

```
% Author: Derry FitzGerald
%----- user data -----
WLen            =4096;
hopsize         =1024;
lh              =17; % length of the harmonic median filter
lp              =17; % length of the percussive median filter
p               =2;
w1              =hanning(WLen,'periodic');
w2              =w1;
hlh             =floor(lh/2)+1;
th              =2500;
[DAFx_in, FS]   =wavread('filename');
L               = length(DAFx_in);
DAFx_in         = [zeros(WLen, 1); DAFx_in; ...
   zeros(WLen-mod(L,hopsize),1)] / max(abs(DAFx_in));
DAFx_out1 = zeros(length(DAFx_in),1);
DAFx_out2 = zeros(length(DAFx_in),1);

%----- initialisations -----
grain           = zeros(WLen,1);
buffer          = zeros(WLen,lh);
buffercomplex   = zeros(WLen,lh);
oldperframe     = zeros(WLen,1);
onall         = [];
onperc        = [];

tic
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
pin  = 0;
pout = 0;
pend = length(DAFx_in)-WLen;

while pin<pend
    grain = DAFx_in(pin+1:pin+WLen).* w1;
```

```
%=========================================
    fc  = fft(fftshift(grain));
    fa  = abs(fc);
    % remove oldest frame from buffers and add
    % current frame to buffers
    buffercomplex(:,1:lh-1)=buffercomplex(:,2:end);
    buffercomplex(:,lh)=fc;
    buffer(:,1:lh-1)=buffer(:,2:end);
    buffer(:,lh)=fa;
    % do median filtering within frame to suppress harmonic instruments
    Per = medfilt1(buffer(:,hlh),lp);
    % do median filtering on buffer to suppress percussion instruments
    Har = median(buffer,2);
    % use these Percussion and Harmonic enhanced frames to generate masks
    maskHar = (Har.^p)./(Har.^p + Per.^p);
    maskPer = (Per.^p)./(Har.^p + Per.^p);
    % apply masks to middle frame in buffer
    % Note: this is the "current" frame from the point of view of the median
    % filtering
    curframe=buffercomplex(:,hlh);
    perframe=curframe.*maskPer;
    harframe=curframe.*maskHar;
    grain1 = fftshift(real(ifft(perframe))).*w2;
    grain2 = fftshift(real(ifft(harframe))).*w2;
    % onset detection functions
    % difference of frames
    dall=buffer(:,hlh)-buffer(:,hlh-1);
    dperc=abs(perframe)-oldperframe;
    oall=sum(dall>0);
    operc=sum(dperc>0);
    onall = [onall oall];
    onperc = [onperc operc];
    oldperframe=abs(perframe);
%=========================================
    DAFx_out1(pout+1:pout+WLen) = ...
       DAFx_out1(pout+1:pout+WLen) + grain1;
   DAFx_out2(pout+1:pout+WLen) = ...
       DAFx_out2(pout+1:pout+WLen) + grain2;
    pin  = pin + hopsize;
    pout = pout + hopsize;
end
%UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
toc
% process onset detection function to get beats
[or,oc]=size(onall);
omin=min(onall);
% get peaks
v1 = (onall > [omin, onall(1:(oc-1))]);
% allow for greater-than-or-equal
v2 = (onall >= [onall(2:oc), omin]);
% simple Beat tracking function
omax = onall .* (onall > th).* v1 .* v2;
% now do the same for the percussion onset detection function
% process onset detection function to get beats
[opr,opc]=size(onperc);
opmin=min(onperc);
% get peaks
```

```
p1 = (onperc > [opmin, onperc(1:(opc-1))]);
% allow for greater-than-or-equal
p2 = (onperc >= [onperc(2:opc), opmin]);
% simple Beat tracking function
opmax = onperc .* (onperc > th).* p1 .* p2;
%----- listening and saving the output -----
DAFx_out1 = DAFx_out1((WLen + hopsize*(hlh-1)) ...
                :length(DAFx_out1))/max(abs(DAFx_out1));
DAFx_out2 = DAFx_out2(WLen + (hopsize*(hlh-1)) ...
                :length(DAFx_out2))/max(abs(DAFx_out2));
% soundsc(DAFx_out1, FS);
% soundsc(DAFx_out2, FS);
wavwrite(DAFx_out1, FS, 'ex-percussion.wav');
wavwrite(DAFx_out2, FS, 'ex-harmonic.wav');
```

Avoiding latency and improving the robustness of the detection function can be obtained by using high-order prediction approaches in frame and frequency direction [NZ08] compared to the simple one-frame prediction (difference). Several further features can be derived from such kinds of low-level beat detector [Fit04, UH03, SZST10].

9.2.4 Pitch extraction

The main task of pitch extraction is to estimate a fundamental frequency f_0, which in musical terms is the pitch of a sound segment, and follow the fundamental frequency over the time. We can use this pitch information to control effects like time stretching and pitch shifting based on the PSOLA method, which is described in Chapter 6, but it also plays a major role in sound modeling with spectral models, which is treated extensively in Chapter 10. Moreover, the fundamental frequency can be used as a control parameter for a variety of audio effects based either on time-domain or on frequency-domain processing.

There is no definitive technique for pitch extraction and tracking, and only the bases of existing algorithms will be described here. We will consider pitch extraction both in the frequency domain and in the time domain. An overview and comparison of the presented algorithms can be found in [KZ10]. Most often an algorithm first looks for candidates of a pitch, then selects one and tries to improve the precision of the choice. After the calculation of pitch candidates post-processing, for example, pitch tracking has to be applied. During post-processing the estimation of the fundamental frequency from the pitch candidates can be improved by taking the frequency relationships between the detected candidates into account, which should ideally be multiples of the fundamental frequency.

FFT-based approach

In this subsection we describe the calculation of pitch candidates from the FFT of a signal segment where the phase information is used. This approach is similar to the technique used in the phase vocoder, see Section 7.3. The main structure of the algorithm is depicted in Figure 9.9, where a segment of length N is extracted every R samples and then applied to FFTs.

Considering the calculation of an N-point FFT, the frequency resolution of the FFT is

$$\Delta f = \frac{f_S}{N}, \tag{9.11}$$

with the sampling frequency $f_S = 1/T_S$. From the input signal $x(n)$ we use a block

$$x_1(n) = x(n_0 + n), \quad n = 0, \ldots, N-1 \tag{9.12}$$

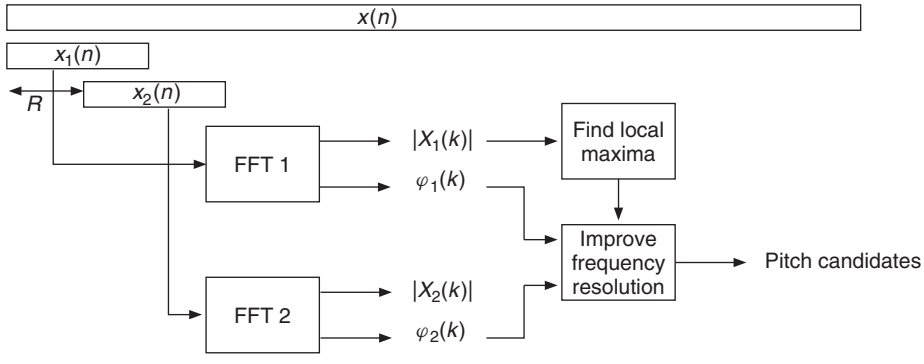

Figure 9.9 FFT-based pitch estimation structure with phase evaluation.

of N samples. After applying an appropriate window, the FFT yields $X_1(k)$ with $k = 0, \ldots, N-1$. At the FFT index k_0 a local maximum of the FFT magnitude $|X_1(k)|$ is detected. From this FFT maximum, the initial estimate of the fundamental frequency is

$$\tilde{f_0} = k_0 \cdot \Delta f = k_0 \frac{f_S}{N}. \tag{9.13}$$

The corresponding normalized frequency is

$$\tilde{\Omega}_0 = 2\pi \, \tilde{f_0} T_S = k_0 \frac{2\pi}{N}. \tag{9.14}$$

To improve the frequency resolution, the phase information can be used, since for a harmonic signal $x_h(n) = \cos(\Omega_0 n + \varphi_0) = \cos(\phi(n))$ the fundamental frequency can be computed by the derivative

$$\Omega_0 = \frac{\mathrm{d}\phi(n)}{\mathrm{d}n}. \tag{9.15}$$

The derivative can be approximated by computing the phases of two FFTs separated by a hop size of R samples leading to

$$\hat{\Omega}_0 = \frac{\Delta\phi}{R}, \tag{9.16}$$

where $\Delta\phi$ is the phase difference between the two FFTs evaluated at the FFT index k_0. The second FFT of the signal segment

$$x_2(n) = x(n_0 + R + n), \quad n = 0, \ldots, N-1 \tag{9.17}$$

leads to $X_2(k)$. For the two FFTs, the phases at frequency $\tilde{f_0}$ are given by

$$\varphi_1 = \angle\{X_1(k_0)\} \tag{9.18}$$

$$\varphi_2 = \angle\{X_2(k_0)\}. \tag{9.19}$$

Both phases φ_1 and φ_2 are obtained in the range $[-\pi, \pi]$. We now calculate an 'unwrapped' φ_2 value corresponding to the value of an instantaneous phase, see also Section 7.3.5 and

Figure 7.17. Assuming that the signal contains a harmonic component with a frequency $\tilde{f}_0 = k_0 \cdot \Delta f$, the expected target phase after a hop size of R samples is

$$\varphi_{2t} = \varphi_1 + \tilde{\Omega}_0 R = \varphi_1 + \frac{2\pi}{N} k_0 R. \tag{9.20}$$

The phase error between the unwrapped value φ_2 and the target phase can be computed by

$$\varphi_{2err} = \text{princarg}(\varphi_2 - \varphi_{2t}). \tag{9.21}$$

The function 'princarg' computes the principal phase argument in the range $[-\pi, \pi]$. It is assumed that the unwrapped phase differs from the target phase by a maximum of π. The unwrapped phase is obtained by

$$\varphi_{2u} = \varphi_{2t} + \varphi_{2err}. \tag{9.22}$$

The final estimate of the fundamental frequency is then obtained by

$$\hat{f}_0 = \frac{1}{2\pi} \hat{\Omega}_0 \cdot f_S = \frac{1}{2\pi} \cdot \frac{\varphi_{2u} - \varphi_1}{R} \cdot f_S. \tag{9.23}$$

Normally we assume that the first pitch estimation $\tilde{f}_0$ differs from the fundamental frequency by a maximum of $\Delta f/2$. Thus the maximum amount for the absolute value of the phase error φ_{2err} is

$$\varphi_{2err,max} = \frac{1}{2} \frac{2\pi}{N} R = \frac{R}{N} \pi. \tag{9.24}$$

We should accept phase errors with slightly higher values to have some tolerance in the pitch estimation.

One simple example of an ideal sine wave at a fundamental frequency of 420 Hz at $f_S = 44.1$ kHz analyzed with the FFT length $N = 1024$ using a Hanning window and hop size $R = 1$ leads to the following results: $k_0 = 10$, $\tilde{f}_0 = k_0 \frac{f_S}{N} = 430.66$ Hz, $\varphi_1/\pi = -0.2474$, $\varphi_{2t}/\pi = -0.2278$, $\varphi_2/\pi = -0.2283$, $\hat{f}_0 = 419.9996$ Hz. Thus the original sine frequency is almost ideally recovered by the described algorithm.

Figure 9.10 shows an example of the described algorithm applied to a short signal of the female utterance 'la' analyzed at an FFT length $N = 1024$. The top plot shows the FFT magnitude, the middle plot the estimated pitch, and the bottom plot the phase error φ_{2err} for frequencies up to 1500 Hz. For this example the frequency evaluation is performed for all FFT bins and not only for those with detected magnitude maxima. The circles show the positions of detected maxima in the FFT magnitude. The dashed lines in the bottom plot show the used threshold for the phase error. In this example the first maximum is detected at FFT index $k_0 = 6$, the corresponding bin frequency is 258.40 Hz, and the corrected pitch frequency is 274.99 Hz. Please notice that in this case the magnitude of the third harmonic (at appr. 820 Hz) has a greater value than the magnitude of the fundamental frequency.

M-file 9.3 presents a MATLAB® implementation to calculate the pitch candidates from a block of the input signal.

M-file 9.3 (`find_pitch_fft.m`)

```
function [FFTidx, Fp_est, Fp_corr] = ...
find_pitch_fft(x, win, Nfft, Fs, R, fmin, fmax, thres)
% [DAFXbook, 2nd ed., chapter 9]
%===== This function finds pitch candidates
%
% Inputs:
```

```
%   x: input signal of length Nfft+R
%   win: window for the FFT
%   Nfft: FFT length
%   Fs: sampling frequency
%   R: FFT hop size
%   fmin, fmax: minumum/ maximum pitch freqs to be detected
%   thres: %omit maxima more than thres dB below the main peak
% Outputs:
%   FFTidx: FFT indices
%   Fp_est: FFT bin frequencies
%   Fp_corr: corrected frequencies

FFTidx  = [];
Fp_est  = [];
Fp_corr = [];
dt      = R/Fs;          % time diff between FFTs
df      = Fs/Nfft;       % freq resolution
kp_min  = round(fmin/df);
kp_max  = round(fmax/df);
x1      = x(1:Nfft);     % 1st block
x2      = x((1:Nfft)+R); % 2nd block with hop size R
[X1, Phi1] = fftdb(x1.*win,Nfft);
[X2, Phi2] = fftdb(x2.*win,Nfft);
X1      = X1(1:kp_max+1);
Phi1    = Phi1(1:kp_max+1);
X2      = X2(1:kp_max+1);
Phi2    = Phi2(1:kp_max+1);
idx     = find_loc_max(X1);
Max     = max(X1(idx));
ii      = find(X1(idx)-Max>-thres);

%----- omit maxima more than thres dB below the main peak -----
idx     = idx(ii);
Nidx    = length(idx);   % number of detected maxima
maxerr  = R/Nfft;        % max phase diff error/pi
                         % (pitch max. 0.5 bins wrong)
maxerr  = maxerr*1.2;    % some tolerance
for ii=1:Nidx
   k        = idx(ii) - 1;          % FFT bin with maximum
   phi1     = Phi1(k+1);            % phase of x1 in [-pi,pi]
   phi2_t   = phi1 + 2*pi/Nfft*k*R; % expected target phase
                                    % after hop size R
   phi2     = Phi2(k+1);            % phase of x2 in [-pi,pi]
   phi2_err = princarg(phi2-phi2_t);
   phi2_unwrap = phi2_t+phi2_err;
   dphi     = phi2_unwrap - phi1;   % phase diff
   if (k>kp_min) & (abs(phi2_err)/pi<maxerr)
      Fp_corr = [Fp_corr; dphi/(2*pi*dt)];
      FFTidx  = [FFTidx; k];
      Fp_est  = [Fp_est; k*df];
   end
end
```

In addition to the algorithm described, the magnitude values of the detected FFT maxima are checked. In the given code those maxima are omitted whose FFT magnitudes are more than `thres`

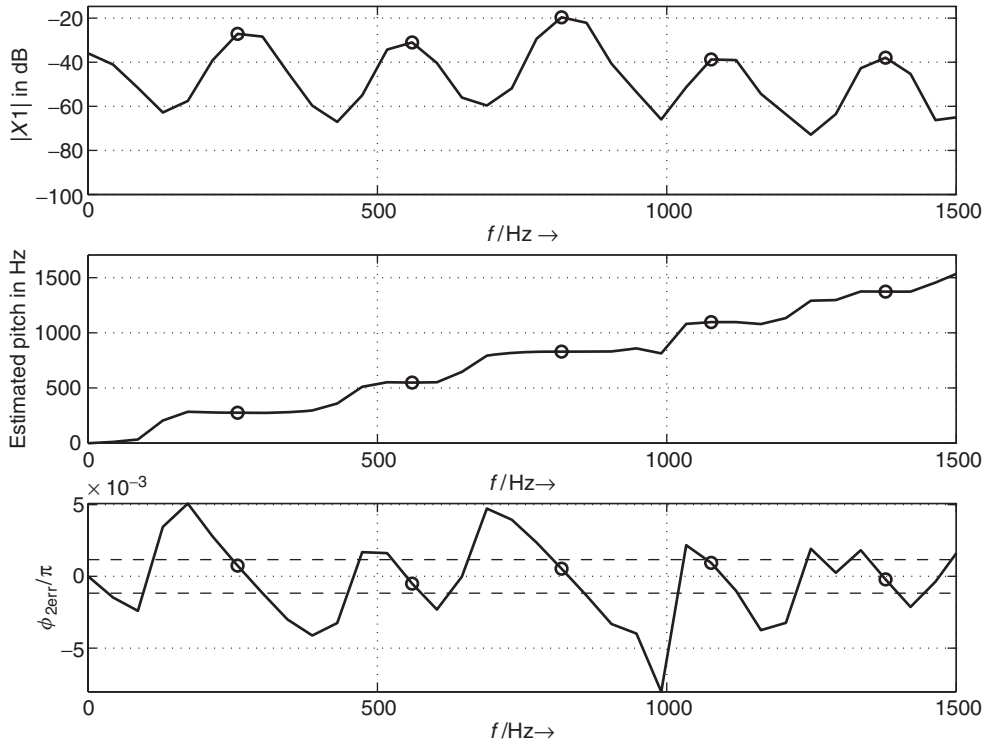

Figure 9.10 Example of pitch estimation of speech signal 'la.'

dB below the global maximum. Typical values for the parameter `thres` lie in the range from 30 to 50. The function `princarg` is given in Figure 7.17. The following function `fftdb` (see M-file 9.4) returns the FFT magnitude in a dB scale and the phase.

M-file 9.4 (`fftdb.m`)

```
function [H, phi]  = fftdb(x, Nfft)
% [DAFXbook, 2nd ed., chapter 9]
%==== This function discards values in FFT bins for which magnitude >Â thresh
if nargin<2
  Nfft = length(x);
end

F      = fft(x,Nfft);
F      = F(1:Nfft/2+1);       % f=0,..,Fs/2
phi    = angle(F);            % phase in [-pi,pi]
F      = abs(F)/Nfft*2;       % normalize to FFT length
%----- return -100 db for F==0 to avoid "log of zero" warnings -----
H      = -100*ones(size(F));
idx    = find(F~=0);
H(idx) = 20*log10(F(idx));    % non-zero values in dB
```

The following function `find_loc_max` (see M-file 9.5) searches for local maxima using the derivative.

M-file 9.5 (`find_loc_max.m`)

```
function [idx, idx0] = find_loc_max(x)
% [DAFXbook, 2nd ed., chapter 9]
%=====  This function finds local maxima in vector x
%  Inputs:
%    x: any vector
%  Outputs:
%    idx : positions of local max.
%    idx0: positions of local max. with 2 identical values
%    if only 1 return value: positions of all maxima

N    = length(x);
dx   = diff(x);             % derivation
                            % to find sign changes from + to -
dx1  = dx(2:N-1);
dx2  = dx(1:N-2);
prod = dx1.*dx2;
idx1 = find(prod<0);        % sign change in dx1
idx2 = find(dx1(idx1)<0);   % only change from + to -
idx  = idx1(idx2)+1;        % positions of single maxima
%----- zeros in dx? => maxima with 2 identical values -----
idx3 = find(dx==0);
idx4 = find(x(idx3)>0);     % only maxima
idx0 = idx3(idx4);
%----- positions of double maxima, same values at idx3(idx4)+1 -----
if nargout==1               % output 1 vector
                            % with positions of all maxima
   idx = sort([idx,idx0]); % (for double max. only 1st position)
end
```

Now we present an example where the algorithm is applied to a signal segment of Suzanne Vega's 'Tom's Diner.' Figure 9.11 shows time-frequency representations of the analysis results. The top plot shows the spectrogram of the signal. The middle plot shows the FFT bin frequencies of detected pitch candidates while the bottom plot shows the corrected frequency values. In the bottom plot the text of the sung words is also shown. In all plots frequencies up to 800 Hz are shown. For the spectrogram an FFT length of 4096 points is used. The pitch-estimation algorithm is performed with an FFT length of 1024 points. This example shows that the melody of the sound can be recognized in the bottom plot of Figure 9.11. The applied algorithm improves the frequency resolution of the FFT shown in the middle plot. To choose the correct pitch among the detected candidates some post-processing is required. Other methods to improve the frequency resolution of the FFT are described in [Can98, DM00, Mar00, Mar98, AKZ99] and in Chapter 10.

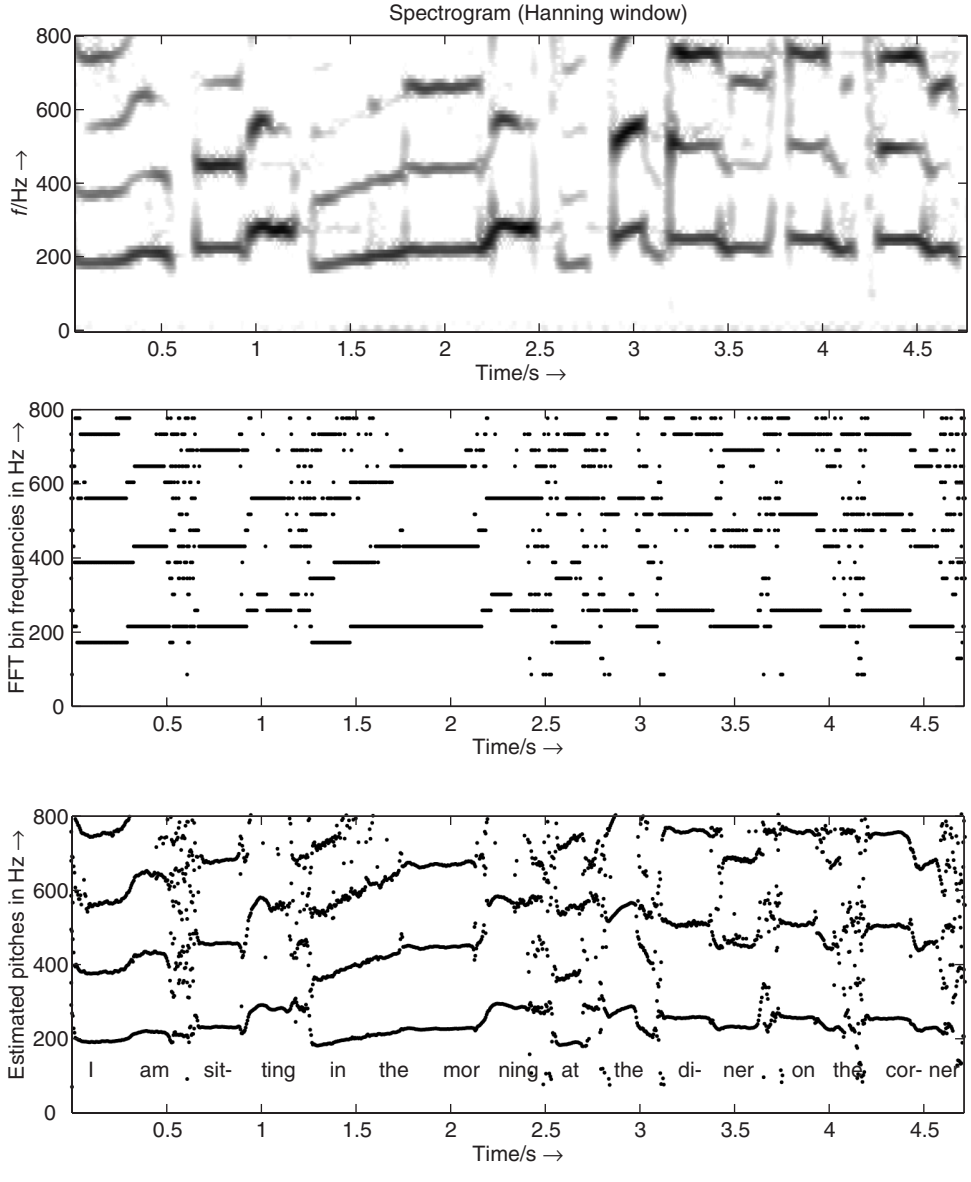

Figure 9.11 Time/frequency planes for pitch estimation example of an excerpt from Suzanne Vega's 'Tom's Diner.' Top: spectrogram, middle: FFT bin frequencies of pitch candidates, bottom: corrected frequency values of pitch candidates.

M-file 9.6 demonstrates a pitch-tracking algorithm in a block-based implementation.

M-file 9.6 (Pitch_Tracker_FFT_Main.m)

```
% Pitch_Tracker_FFT_Main.m    [DAFXbook, 2nd ed., chapter 9]
%===== This function demonstrates a pitch tracking algorithm
%===== in a block-based implementation

%----- initializations -----
fname='Toms_diner';
n0=2000; %start index
n1=210000;

Nfft=1024;
R=1;      % FFT hop size for pitch estimation
K=200;    % hop size for time resolution of pitch estimation
thres=50; % threshold for FFT maxima
% checked pitch range in Hz:
fmin=50;
fmax=800;
p_fac_thres=1.05;  % threshold for voiced detection
                   % deviation of pitch from mean value
win=hanning(Nfft)';% window for FFT
Nx=n1-n0+1+R;      % signal length
blocks=floor(Nx/K);
Nx=(blocks-1)*K+Nfft+R;
n1=n0+Nx;          % new end index
[X,Fs]=wavread(fname,[n0,n1]);
X=X(:,1)';

%----- pitch extraction per block -----
pitches=zeros(1,blocks);
for b=1:blocks
  x=X((b-1)*K+1+(1:Nfft+R));
  [FFTidx, F0_est, F0_corr]= ...
     find_pitch_fft(x,win,Nfft,Fs,R,fmin,fmax,thres);
  if ~isempty(F0_corr)
    pitches(b)=F0_corr(1); % take candidate with lowest pitch
  else
    pitches(b)=0;
  end
end
%----- post-processing -----
L=9;              % odd number of blocks for mean calculation
D=(L-1)/2;        % delay
h=ones(1,L)./L;  % impulse response for mean calculation
%----- mirror beginning and end for "non-causal" filtering -----
p=[pitches(D+1:-1:2),pitches,pitches(blocks-1:-1:blocks-D)];
y=conv(p,h);           % length: blocks+2D+2D
pm=y((1:blocks)+2*D); % cut result

Fac=zeros(1,blocks);
idx=find(pm~=0);      % don't divide by zero
Fac(idx)=pitches(idx)./pm(idx);
ii=find(Fac<1 & Fac~=0);
Fac(ii)=1./Fac(ii);   % all non-zero elements are now > 1
```

```
%----- voiced/unvoiced detection -----
voiced=Fac~=0 & Fac<p_fac_thres;

T=40;                 % time in ms for segment lengths
M=round(T/1000*Fs/K); % min. number of consecutive blocks
[V,p2]=segmentation(voiced, M, pitches);
p2=V.*p2;             % set pitches to zero for unvoiced

%----- plotting and drawing figure -----
figure(1),clf,
time=(0:blocks-1)*K+1; % start sample of blocks
time=time/Fs;          % time in seconds
t=(0:length(X)-1)/Fs;  % time in sec for original
subplot(211)
plot(t,X),title('original x(n)')
axis([0 max([t,time]) -1.1*max(abs(X)) 1.1*max(abs(X))])
subplot(212)
idx=find(p2~=0);
plot_split(idx,time, p2),title('pitch in Hz');
xlabel('time/s \rightarrow');
axis([0 max([t,time]) .9*min(p2(idx)) 1.1*max(p2(idx))])
```

In the above implementation the post-processing is performed by choosing the lowest pitch candidate in each block. Then the mean pitch of surrounding blocks is computed and compared to the detected pitch. If the deviation from the mean value is higher than a given threshold, this block is considered as 'unvoiced.' Finally a segmentation is performed to get a minimum number of consecutive blocks that are voiced/unvoiced (to avoid very short segments). M-file 9.7 presents an implementation for the segmentation.

M-file 9.7 (segmentation.m)

```
function [V,pitches2] = segmentation(voiced, M, pitches)
% function [V,pitches2] = segmentation(voiced, M, pitches)
%    [DAFXbook, 2nd ed., chapter 9]
%===== This function implements the pitch segmentation
%
%  Inputs:
%    voiced:   original voiced/unvoiced detection
%    M:        min. number of consecutive blocks with same voiced flag
%    pitches:  original pitches
%  Outputs:
%    V:        changed voiced flag
%    pitches2: changed pitches

blocks=length(voiced); % get number of blocks
pitches2=pitches;
V=voiced;
Nv=length(V);

%%%%%%%%%% step1: eliminate too short voiced segments:
V(Nv+1)=~V(Nv);  % change at end to get length of last segment
dv=[0, diff(V)]; % derivative
idx=find(dv~=0); % changes in voiced
di=[idx(1)-1,diff(idx)];  % segment lengths
```

```
v0=V(1);         % status of 1st segment
k0=1;
ii=1; % counter for segments, idx(ii)-1 is end of segment
if v0==0
  k0=idx(1); % start of voiced
  ii=ii+1;   % first change voiced to unvoiced
end
while ii<=length(idx);
  L=di(ii);
  k1=idx(ii)-1;   % end of voiced segment
  if L<M
     V(k0:k1)=zeros(1,k1-k0+1);
  end
  if ii<length(idx)
    k0=idx(ii+1); % start of next voiced segment
  end
  ii=ii+2;
end

%%%%%%%%%%% step2: eliminate too short unvoiced segments:
V(Nv+1)=~V(Nv);           % one more change at end
dv=[0, diff(V)];
idx=find(dv~=0);          % changes in voiced
di=[idx(1)-1,diff(idx)];  % segment lengths
if length(idx)>1          % changes in V
  v0=V(1);                % status of 1st segment
  k0=1;
  ii=1;    % counter for segments, idx(ii)-1 is end of segment
  if v0==0
    k0=idx(2); % start of unvoiced
    ii=ii+2;   % first change unvoiced to voiced
  end
  while ii<=length(idx);
    L=di(ii);
    k1=idx(ii)-1;   % end of unvoiced segment
    if L<M
       if k1<blocks % NOT last unvoiced segment
         V(k0:k1)=ones(1,k1-k0+1);
         % linear pitch interpolation:
         p0=pitches(k0-1);
         p1=pitches(k1+1);
         N=k1-k0+1;
         pitches2(k0:k1)=(1:N)*(p1-p0)/(N+1)+p0;
      end
    end
    if ii<length(idx)
      k0=idx(ii+1); % start of next unvoiced segment
    end
    ii=ii+2;
  end
end

V=V(1:Nv); % cut last element
```

The `plot_split` function is given by M-file 9.8.

M-file 9.8 (`plot_split.m`)

```
function plot_split(idx, t, x)
% function plot_split(idx, t, x)  [DAFXbook, 2nd ed., chapter 9]
%===== This function plots the segmented pitch curve
%  Inputs:
%    idx: vector with positions of vector x to be plotted
%    t: time indexes
%    x is segmented into parts

di=diff(idx);
L=length(di);

n0=1;
pos_di=find(di>1);
ii=1; % counter for pos_di

hold off
while ii<=length(pos_di) %n0<=length(x)
  n1=pos_di(ii);
  plot(t(idx(n0:n1)),x(idx(n0:n1)))
  hold on
  n0=n1+1;
  ii=ii+1;
end

n1=length(idx);
plot(t(idx(n0:n1)),x(idx(n0:n1)))
hold off
```

The result of the pitch-tracking algorithm is illustrated in Figure 9.12. The bottom plot shows the pitch over time calculated using the block-based FFT approach.

Any FFT-based pitch estimator can be improved by detecting the harmonic structure of the sound. If the harmonics of the fundamental frequency are detected, the greatest common divisor of these harmonic frequencies can be used in the estimation of the fundamental frequency [O'S00, p. 220]. M.R. Schroeder mentions for speech processing in [Sch99, p. 65], 'the pitch problem was finally laid to rest with the invention of cepstrum pitch detectors' [Nol64]. The cepstrum technique allows the estimation of the pitch period directly from the cepstrum sequence $c(n)$. Schroeder also suggested a 'harmonic product spectrum' [Sch68] to improve the fundamental frequency estimation, which sometimes outperforms the cepstrum method [Sch99, p. 65]. A further improvement of the pitch estimates can be achieved by applying a peak-continuation algorithm to the detected pitches of adjacent frames, which is described in Chapter 10.

General remarks on time-domain pitch extraction

In the time domain the task of pitch extraction leads us to find the corresponding *pitch period.* The pitch period is the time duration of one period. With the fundamental frequency f_0 (to be detected) the pitch period is given by

$$T_0 = \frac{1}{f_0}. \tag{9.25}$$

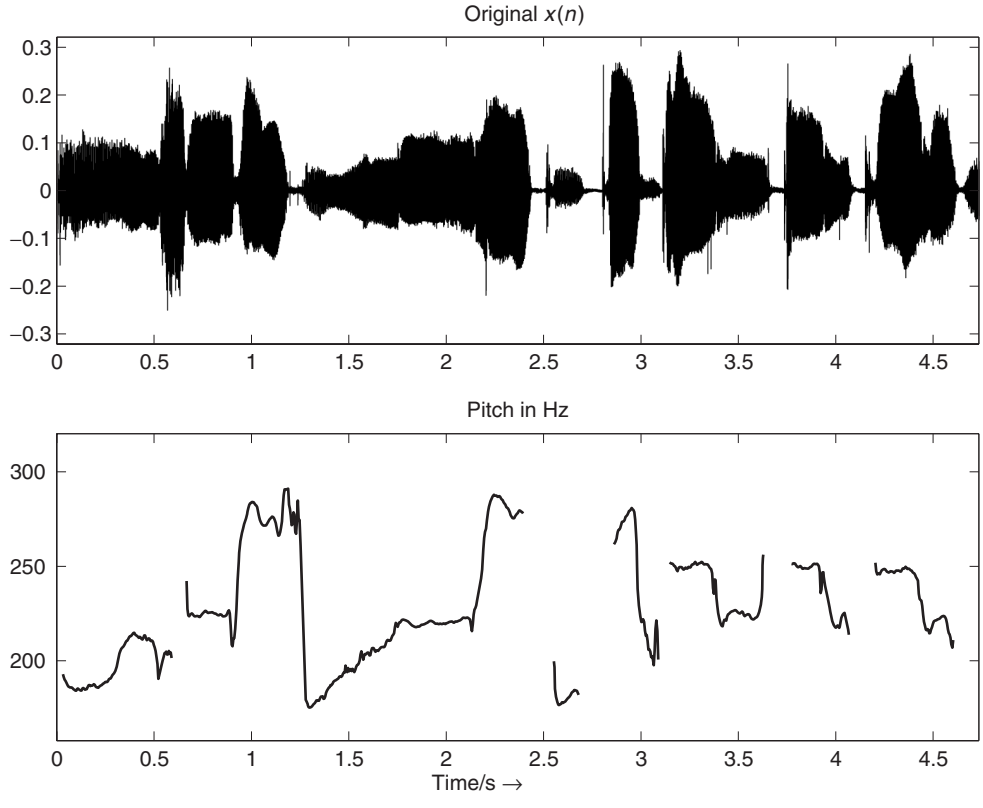

Figure 9.12 Pitch over time from the FFT with phase vocoder approach for a signal segment of Suzanne Vega's 'Tom's Diner.'

For a discrete-time signal sampled at $f_S = \frac{1}{T_S}$ we have to find the *pitch lag* M, which is the number of samples in one period. The pitch period is $T_0 = M \cdot T_S$, which leads to

$$M = \frac{T_0}{T_S} = \frac{f_S}{f_0}. \tag{9.26}$$

Since only integer-valued pitch lags can be detected, we have a certain frequency resolution in detecting the fundamental frequency dependent on f_0 and f_S. Now we are assuming the case of $\tilde{M} = M + 0.5$ where $\tilde{M}$ is the detected integer pitch lag. The detected fundamental frequency is $\tilde{f}_0 = \frac{f_S}{\tilde{M}}$ instead of the exact pitch $f_0 = \frac{f_S}{M}$. The frequency error factor is in this case

$$\alpha(f_0) = \frac{f_0}{\tilde{f}_0} = \frac{\tilde{M}}{M} = 1 + \frac{0.5}{M} = 1 + 0.5\frac{f_0}{f_S}. \tag{9.27}$$

With the halftone factor $\alpha_{ht} = \sqrt[12]{2}$ and setting $\alpha(f_0) = \alpha_{ht}^x$, the frequency error in halftones is

$$x = \frac{\ln \alpha(f_0)}{\ln \alpha_{ht}}. \tag{9.28}$$

Figure 9.13 shows the frequency error both as factor $\alpha(f_0)$ and as percentage of halftones for pitches in the range from 50 to 5000 Hz at the sampling frequency $f_S = 44.1$ kHz. The maximum

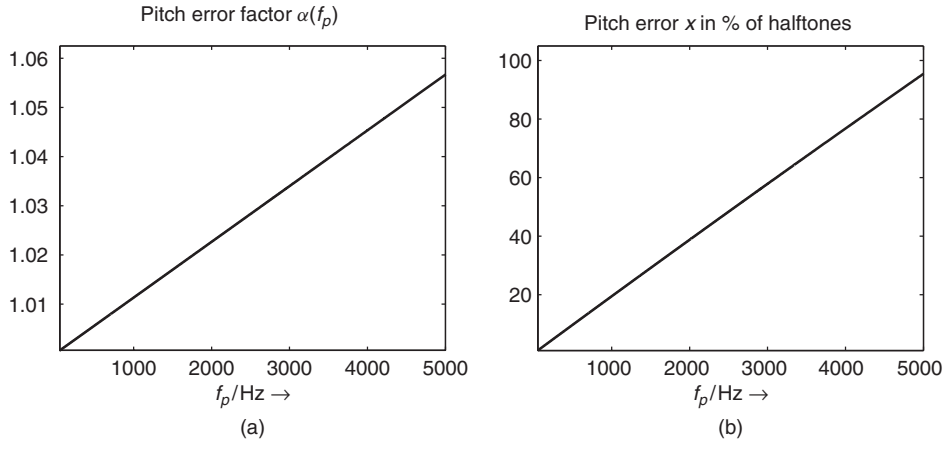

Figure 9.13 Resolution of time-domain pitch detection at $f_S = 44.1$ kHz, (a) frequency error factor, (b) pitch error in percentage of a halftone.

frequency error is approximately 6%, or one halftone, for pitches up to 5000 Hz. For a fundamental frequency of 1000 Hz the frequency error is only 20% of a halftone which is reasonably accurate precision.

Normally the pitch estimation in the time domain is performed in three steps [O'S00, p. 218]:

(1) Segmentation of the input signal into overlapping blocks and pre-processing of each block, for example lowpass filtering (see segmentation shown in Figure 9.9).

(2) Basic pitch estimation algorithm applied to the pre-processed block.

(3) Post-processing for an error correction of the pitch estimates and smoothing of pitch trajectories.

Auto-correlation and LPC

The auto-correlation sequence can also be used to detect the pitch period of a signal segment. First, we present different definitions of autocorrelation sequences:

- Using one block

$$r_{xx}(m) = \sum_{n=m}^{N-1} x(n)x(n-m) \tag{9.29}$$

- Using one windowed block

$$r_{xx}(m) = \sum_{n=m}^{N-1} u(n)u(n-m), \tag{9.30}$$

  with $u(n) = x(n) \cdot w(n)$ (window function $w(n)$)

- and Using the exact signal, thus using samples preceding the considered block

$$\tilde{r}_{xx}(m) = \sum_{n=0}^{N-1} x(n)x(n-m). \tag{9.31}$$

Notice, that in the definitions given by (9.29)–(9.31) no normalization to the block length N is applied.

Figure 9.14 shows the three different auto-correlation sequences for an excerpt of the speech signal "la." Here the same input signal is used as in Figure 9.10. In this example the pitch lag corresponding to the fundamental frequency is $M = 160$ samples, and thus at the third maximum of the auto-correlation. Normally we expect the first maximum in the auto-correlation at the pitch lag. But sometimes, as in this example, the first maximum in the auto-correlation function is not at this position. In general, the auto-correlation has maxima at the pitch lag M and at its multiples, since, for a periodic signal, the same correlation occurs if comparing the signal with the same signal delayed by multiples of the pitch period. Since, in the example of Figures 9.10 and 9.14, the third harmonic is more dominant than the fundamental frequency, the first maximum in the auto-correlation is located at $M/3$. Conversely there can be a higher peak in the auto-correlation after the true pitch period.

Often the prediction error of an LPC analysis contains peaks spaced by the pitch period, see Figure 8.9. Thus it might be promising to try to estimate the pitch period from the prediction error

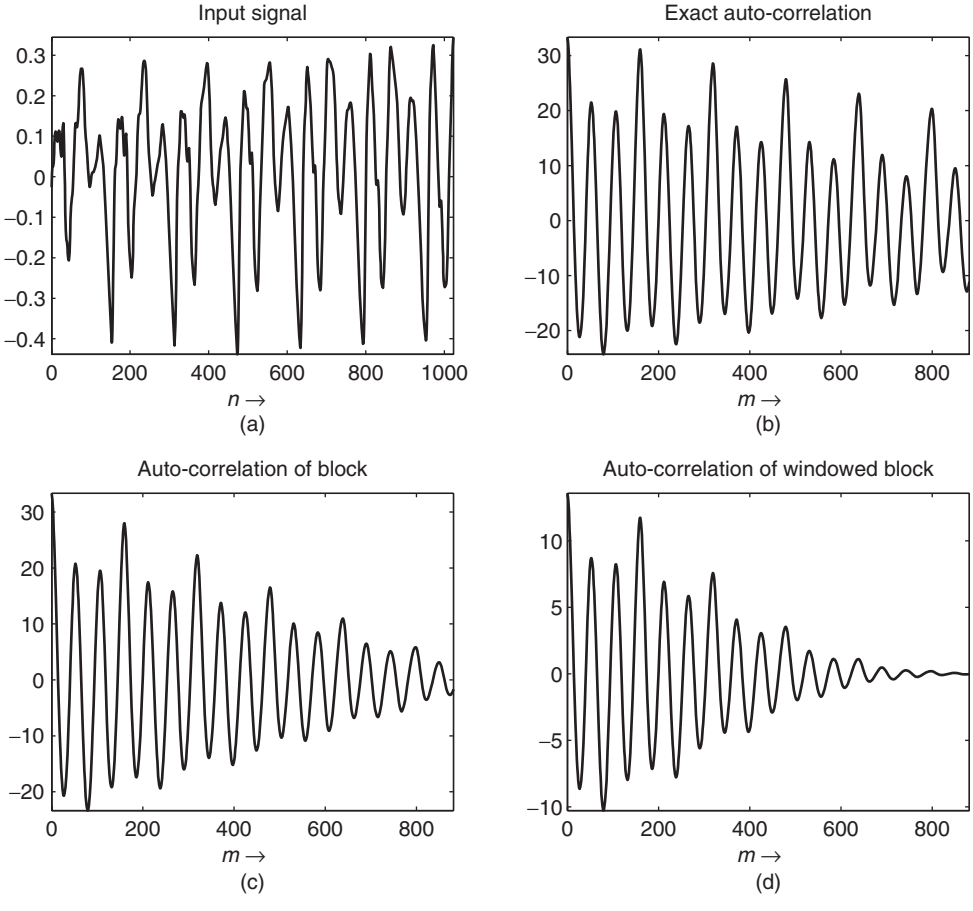

Figure 9.14 Comparison between different autocorrelation computations for speech signal 'la.': (a) input block $x(n)$, (b) exact auto-correlation $\tilde{r}_{xx}(m)$, (c) standard auto-correlation $r_{xx}(m)$ using block, (d) standard auto-correlation $r_{xx}(m)$ using windowed block.

instead of using the original signal. The SIFT algorithm [Mar72], which has been developed for voice, is based on removing the spectral envelope by inverse filtering in a linear prediction scheme. But in some cases it is not possible to estimate the pitch period from the prediction error, because the linear prediction has removed all pitch redundancies from the signal. Figure 9.15 compares two excerpts of a speech signal where the input block (top) and the auto-correlations of both the input signal (middle) and the prediction error (bottom) are shown. An LPC analysis of order $p = 8$ using the auto-correlation method has been applied.

For the example presented in subplots (a)–(c) the pitch period can be well detected in the auto-correlation of the prediction error (same excerpt as in Figures 9.10 and 9.14). For the other excerpt presented in subplots (d)–(f) it is not possible to detect the pitch in the auto-correlation of the prediction error while the auto-correlation of the input signal has a local maximum at the correct pitch period. Notice that in the plots of the auto-correlation sequences only time-lag ranges from 29 to 882 are shown. This corresponds to pitch frequencies from 1500 down to 50 Hz at sampling frequency 44.1 kHz.

Another time-domain method for the extraction of the fundamental frequency is based on 'center clipping' the input signal and subsequent auto-correlation analysis [Son68]. First, the input signal is bandlimited by a lowpass filter. If the filter output signal exceeds a certain threshold $\pm c$ the operation $x_{clip}(n) = x(n) \mp c$ is performed, otherwise $x_{clip}(n) = 0$ [RS78, Son68]. The result of this pre-processing is illustrated in Figure 9.16. The auto-correlation sequence $r_{xx}(m)$ of the center-clipped signal $x_{clip}(n)$ shows a strong positive peak at the time lag of the pitch period.

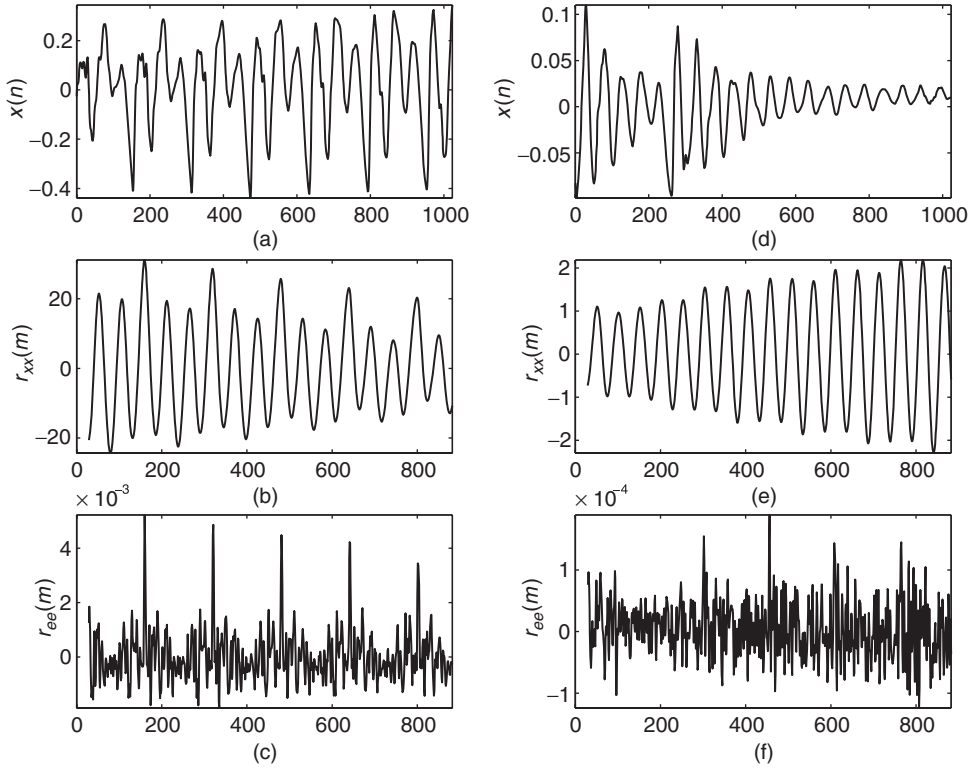

Figure 9.15 Auto-correlation sequences for input signal and prediction error for two excerpts of the speech signal 'la.' Input signals (a, d), auto-correlation of input (b, e), auto-correlation of prediction error (c, f).

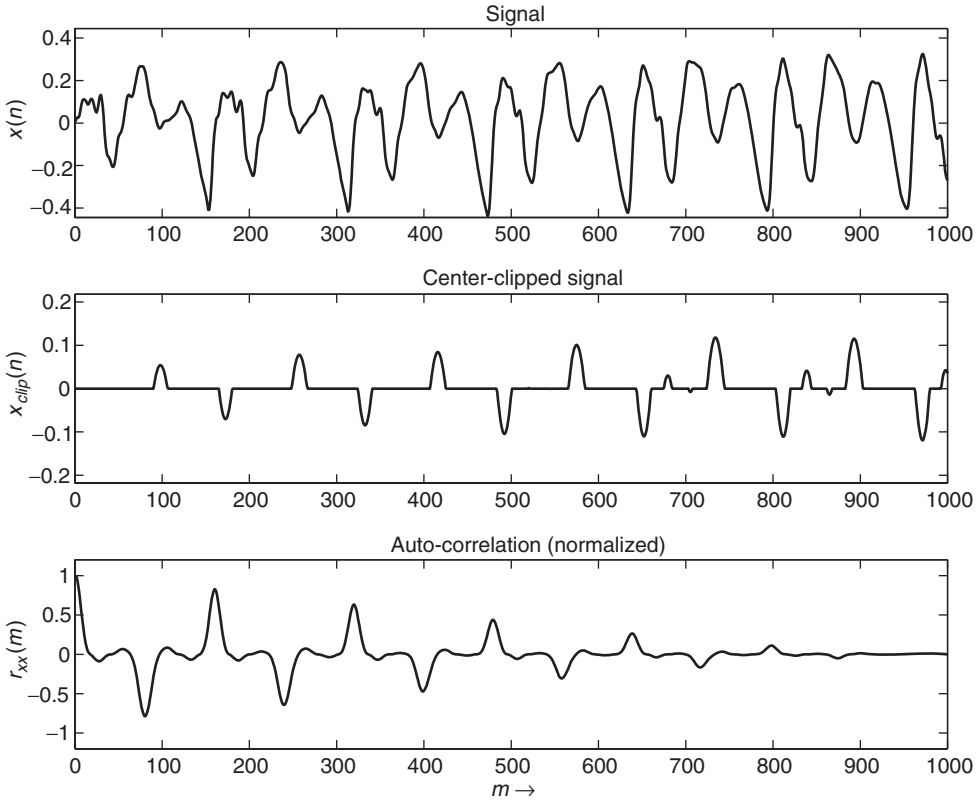

Figure 9.16 Center clipping and subsequent auto-correlation analysis: input signal, lowpass filtered, and center-clipped signal (notice the time delay) and auto-correlation.

Narrowed correlation: the YIN algorithm

Another way to determine the periodicity of a signal is to calculate the sum of differences of a time frame with its shifted version analogous to the ACF. The average magnitude difference function (AMDF) [RSC+74] is defined as

$$d(l) = \frac{1}{N} \sum_{n=0}^{N-1} |x(n) - x(n-l)|. \tag{9.32}$$

The function properties are similar to the ACF, but the AMDF shows dips at the lags of high correlation instead of peaks like the ACF does. De Cheveigné [dCK02] defined the difference function as sum of squared differences

$$d_t(l) = \sum_{n=1}^{N-1} (x(n) - x(n+l))^2. \tag{9.33}$$

The ACF and the difference function are both sensitive to amplitude changes of the time signal. An increasing amplitude of the time signal leads to higher peaks at later lags for the ACF and lower dips for the difference function respectively. To avoid this a cumulative normalization is

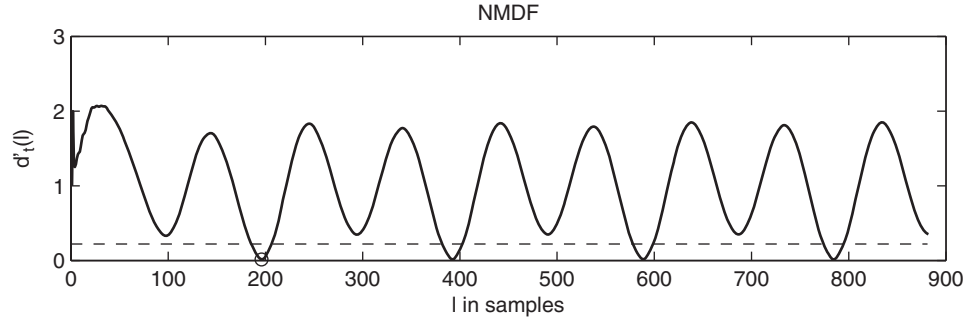

Figure 9.17 YIN algorithm: normalized mean difference function (NDMF) for a signal segment of Suzanne Vega's 'Tom's Diner.'

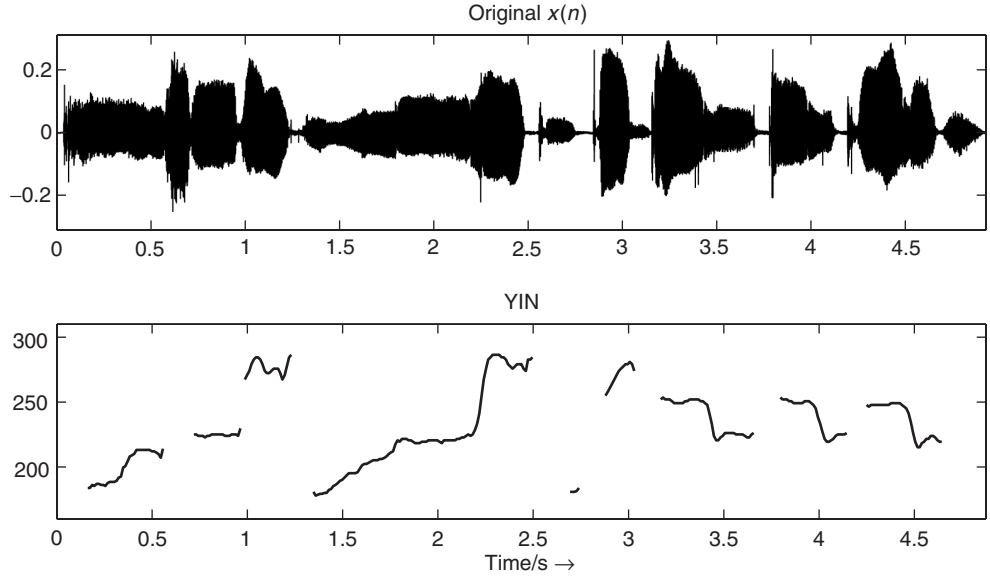

Figure 9.18 Pitch over time from the YIN algorithm for a signal segment of Suzanne Vega's 'Tom's Diner.'

applied to average the current lag value with the previous values. The result is the normalized mean difference function (NMDF) defined as

$$d'_t(l) = \begin{cases} 1, & l = 0 \\ \dfrac{d_t(l)}{\frac{1}{l}\sum_{n=1}^{l} d_t(n)}, & \text{else.} \end{cases} \tag{9.34}$$

The NMDF starts at a value of 1 and drops below 1 only where the current lag value is below the average of all previous lags. This allows one to define a threshold, which a local minimum $d'_t(l)$ has to fall below in order to consider it as a valid pitch candidate (see figure 9.17). To increase the frequency resolution of the YIN algorithm the local minima of $d'_t(l)$ can be refined by parabolic interpolation with their neighboring lag values. The minimum of the parabola is used as the refined

lag value. The result of the presented pitch-tracking algorithm is illustrated in Figure 9.18. The bottom plot shows the pitch over time calculated using the YIN algorithm.

M-file 9.9 (yinDAFX.m)

```
function pitch = yinDAFX(x,fs,f0min,hop)
% function pitch = yinDAFX(x,fs,f0min,hop)
% Author: Adrian v.d. Knesebeck
% determines the pitches of the input signal x at a given hop size
%
% input:
% x         input signal
% fs        sampling frequency
% f0min     minimum detectable pitch
% hop       hop size
%
% output:
% pitch     pitch frequencies in Hz at the given hop size

% initialization
yinTolerance = 0.22;
taumax = round(1/f0min*fs);
yinLen = 1024;
k = 0;

% frame processing
for i = 1:hop:(length(x)-(yinLen+taumax))
    k=k+1;
    xframe = x(i:i+(yinLen+taumax));
    yinTemp = zeros(1,taumax);
    % calculate the square differences
        for tau=1:taumax
            for j=1:yinLen
                yinTemp(tau) = yinTemp(tau) + (xframe(j) - xframe(j+tau))^2;
            end
        end

    % calculate cumulated normalization
        tmp = 0;
        yinTemp(1) = 1;
        for tau=2:taumax
            tmp = tmp + yinTemp(tau);
            yinTemp(tau) = yinTemp(tau) *(tau/tmp);
        end

    % determine lowest pitch
      tau=1;
      while(tau<taumax)
          if(yinTemp(tau) < yinTolerance)
              % search turning point
              while (yinTemp(tau+1) < yinTemp(tau))
                  tau = tau+1;
              end
              pitch(k) = fs/tau;
              break
          else
```

```
            tau = tau+1;
        end
    % if no pitch detected
    pitch(k) = 0;
    end
end
end
```

Long-term prediction (LTP)

A further method of estimating the fundamental frequency is based on long-term prediction. A common approach to remove pitch-period redundancies from a signal is to use a short FIR prediction filter after a delay line of M samples, where M is the pitch lag [KA90]. Thus the long-term prediction error or residual is given by

$$d(n) = x(n) - \sum_{k=0}^{q} b_k x(n - M - k), \tag{9.35}$$

where the order $q + 1$ is normally in the range $\{1, 2, 3\}$ [KA90]. Considering the case of a one-tap filter, the residual simplifies to

$$d(n) = x(n) - b_0 \cdot x(n - M), \tag{9.36}$$

which is shown in Figure 9.19.

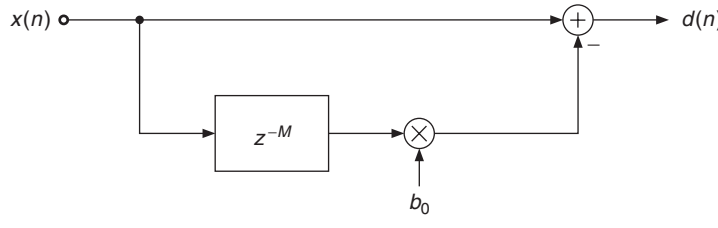

Figure 9.19 Long-term prediction with a one-tap filter.

For minimizing the energy of $d(n)$ over one block of length N we set the derivative with respect to b_0 to zero. This leads to the optimal filter coefficient

$$b_0 = \frac{\tilde{r}_{xx}(M)}{r_{xx0}(M)}, \tag{9.37}$$

with $\tilde{r}_{xx}(m)$ as defined in (9.31) and

$$r_{xx0}(m) = \sum_{n=0}^{N-1} x^2(n - m), \tag{9.38}$$

which is the energy of a block delayed by m samples. Setting this solution into (9.36) leads to the error energy

$$E_d = \sum_{n=0}^{N-1} x^2(n) - r_{xx,norm}(M) \tag{9.39}$$

dependent on M with

$$r_{xx,norm}(m) = \frac{\tilde{r}_{xx}^2(m)}{r_{xx0}(m)}. \tag{9.40}$$

Figure 9.20 shows an example where the input signal shown in Figure 9.15(a) is used. The top plot shows the exact autocorrelation $\tilde{r}_{xx}(m)$, the middle plot shows the normalized autocorrelation $r_{xx,norm}(m)$, and the bottom plot shows the LTP coefficient $b_0(m)$ dependent on the lag m.

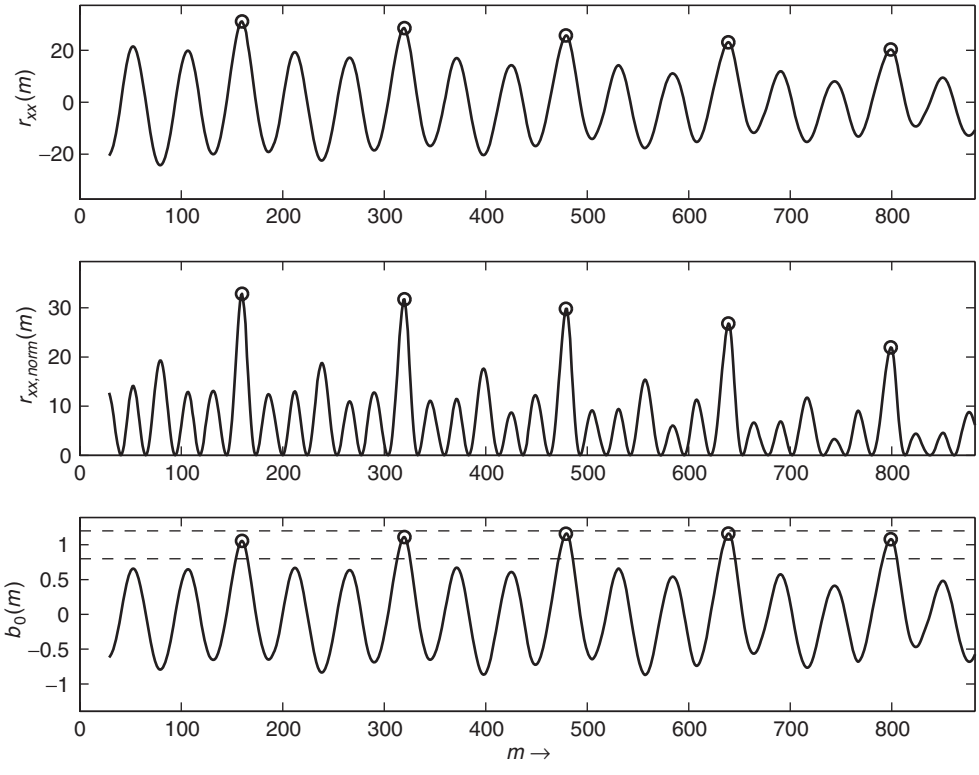

Figure 9.20 Auto-correlation, normalized auto-correlation and LTP coefficient dependent on lag m for excerpt from the speech signal 'la.' The circles show the pitch lag candidates, the dashed lines the accepted b_0 values.

In $r_{xx,norm}(m)$ the lag $m = M$ has to be found where $r_{xx,norm}(m)$ is maximized to minimize the residual energy. Considering one block of length N, the numerator of (9.40) is the squared auto-correlation, while the denominator is the energy of the block delayed by M samples. The function $r_{xx,norm}(m)$ therefore represents a kind of normalized auto-correlation sequence with only positive values. If used for the detection of the pitch period, $r_{xx,norm}(m)$ does not need to have a global maximum at $m = M$, but it is expected to have a local maximum at that position.

To find candidates of the pitch lag M, first local maxima in $r_{xx,norm}(m)$ are searched. In a second step, from these maxima only those ones are considered where the auto-correlation $\tilde{r}_{xx}(m)$ is positive valued. The function $r_{xx,norm}(m)$ also has maxima at positions where $\tilde{r}_{xx}(m)$ has minima. In a third step the $b_0(m)$ values are considered. The value of the coefficient b_0 is close to one for

voiced sounds and close to zero for noise-like sounds [JN84, p. 315]. Thus the value of b_0 can serve as a quality check for the estimate of the computed pitch lag.

In the example in Figure 9.20, b_0 values in the range $0.8, \ldots 1.2$ are accepted. This range is shown by the dashed lines in the bottom plot. The circles represent the positions of pitch-lag candidates. Thus, at these positions $r_{xx,norm}(m)$ has a local maximum, $\tilde{r}_{xx}(m)$ is positive valued, and $b_0(m)$ lies in the described range. In this example, the first pitch lag candidate corresponds to the pitch of the sound segment.

The described algorithm for the computation of pitch-lag candidates from a signal block is implemented by the following M-file 9.10.

M-file 9.10 (find_pitch_ltp.m)

```
function [M,Fp] = find_pitch_ltp(xp, lmin, lmax, Nblock, Fs, b0_thres)
% function [M,Fp] = find_pitch_ltp(xp, lmin, lmax, Nblock, Fs, b0_thres)
%    [DAFXbook, 2nd ed., chapter 9]
%===== This function computes the pitch lag candidates from a signal block
%
%  Inputs:
%      xp      : input block including lmax pre-samples
%                for correct autocorrelation
%      lmin    : min. checked pitch lag
%      lmax    : max. checked pitch lag
%      Nblock  : block length without pre-samples
%      Fs      : sampling freq.
%      b0_thres: max b0 deviation from 1
%
%  Outputs:
%    M: pitch lags
%    Fp: pitch frequencies

lags  = lmin:lmax;        % tested lag range
Nlag  = length(lags);     % no. of lags
[rxx_norm, rxx, rxx0] = xcorr_norm(xp, lmin, lmax, Nblock);

%----- calc. autocorr sequences -----
B0    = rxx./rxx0;        % LTP coeffs for all lags
idx   = find_loc_max(rxx_norm);
i     = find(rxx(idx)>0); % only max. where r_xx>0
idx   = idx(i);           % indices of maxima candidates
i     = find(abs(B0(idx)-1)<b0_thres);

%----- only max. where LTP coeff is close to 1 -----
idx   = idx(i);           % indices of maxima candidates

%----- vectors for all pitch candidates: -----
M     = lags(idx);
M     = M(:);             % pitch lags
Fp    = Fs./M;
Fp    = Fp(:);            % pitch freqs
```

The function `find_loc_max` is given in Section 9.2.4. The function `xcorr_norm` to compute the auto-correlation sequences is given by M-file 9.11.

M-file 9.11 (`xcorr_norm.m`)

```
function [rxx_norm, rxx, rxx0] = xcorr_norm(xp, lmin, lmax, Nblock)
% function [rxx_norm, rxx, rxx0] = xcorr_norm(xp, lmin, lmax, Nblock)
%    [DAFXbook, 2nd ed., chapter 9]
%===== This function computes the normalized autocorrelation
%  Inputs:
%    xp: input block
%    lmin: min of tested lag range
%    lmax: max of tested lag range
%    Nblock: block size
%
%  Outputs:
%    rxx_norm: normalized autocorr. sequence
%    rxx: autocorr. sequence
%    rxx0: energy of delayed blocks

%----- initializations -----
x        = xp((1:Nblock)+lmax); % input block without pre-samples
lags     = lmin:lmax;           % tested lag range
Nlag     = length(lags);        % no. of lags

%----- empty output variables -----
rxx      = zeros(1,Nlag);
rxx0     = zeros(1,Nlag);
rxx_norm = zeros(1,Nlag);

%----- computes autocorrelation(s) -----
for l=1:Nlag
   ii      = lags(l);           % tested lag
   rxx0(l) = sum(xp((1:Nblock)+lmax-lags(l)).^2);
   %----- energy of delayed block
   rxx(l)  = sum(x.*xp((1:Nblock)+lmax-lags(l)));
end
rxx_norm=rxx.^2./rxx0;          % normalized autocorr. sequence
```

The performance of the function `xcorr_norm` is quite slow in **MATLAB**. The computation speed can be improved if using a C-MEX function. Thus the function is implemented in C and a 'MEX' file is created with the C compiler (on Windows systems the MEX file is a dll). In this example the computation speed is improved by a factor of approximately 50, if using the MEX file instead of the **MATLAB** function.

The described LTP algorithm may also be applied to the prediction error of a linear prediction approach. Figure 9.21 compares LTP applied to original signals and their prediction errors. In this example the same signal segments are used as in Figure 9.15. The circles denote the detected global maxima in the normalized auto-correlation. For the first signal shown in plots (a)–(c) the computed LTP coefficients are $b_{0x} = 1.055$ for the input signal and $b_{0e} = 0.663$ for the prediction error. The LTP coefficients for the second signal are $b_{0x} = 0.704$ and $b_{0e} = 0.141$, respectively. As in Figure 9.15, the pitch estimation from the prediction error works well for the first signal while this approach fails for the second signal. For the second signal the value of the LTP coefficient indicates that the prediction error is noise-like.

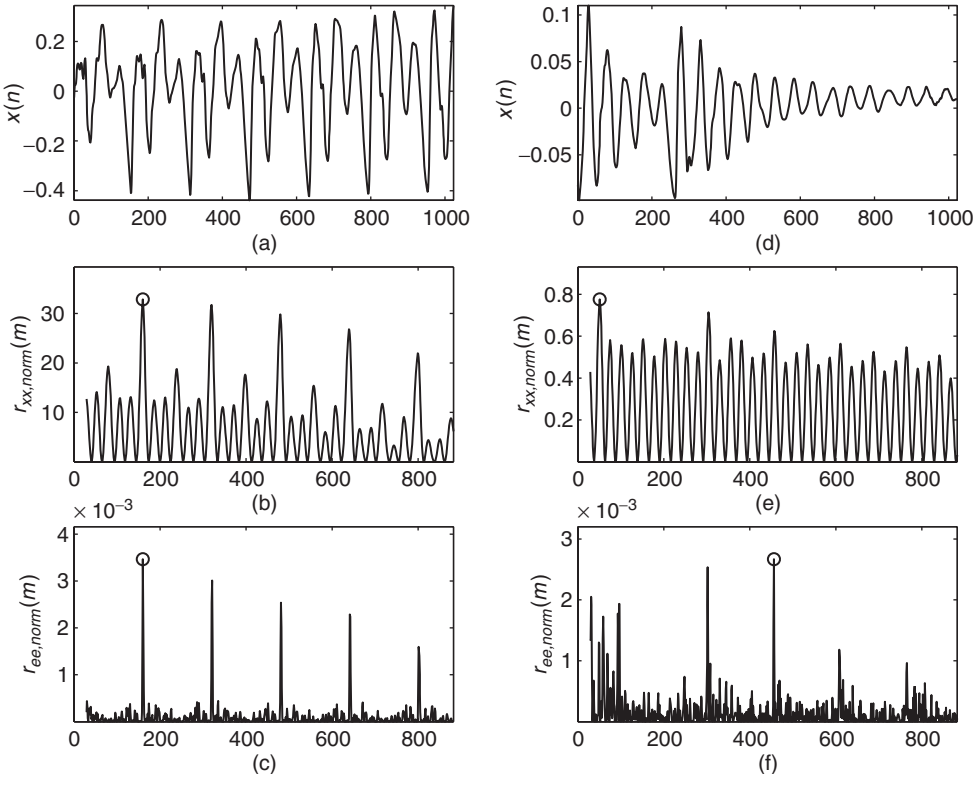

Figure 9.21 Normalized auto-correlation sequences for input signal and prediction error for two excerpts of the speech signal 'la.' Input signals (a, d), normalized auto-correlation of input (b, e), normalized auto-correlation of prediction error (c, f).

Figure 9.22 shows the detected pitch-lag candidates and the corresponding frequencies over time for a signal segment of Suzanne Vega's 'Tom's Diner.' It is the same example as presented in Figure 9.11 where also a spectrogram of this sound signal is given. The top plot of Figure 9.22 shows the detected pitch-lag candidates computed by the LTP algorithm applied to the input signal. The parameter `b0_thres` is set to 0.3, thus b_0 values in the range $0.7, \ldots, 1.3$ are accepted. The corresponding pitch frequencies in the bottom plot are computed by $f_p = f_S/M$ (see (9.26)).

In the top plot of Figure 9.22 the lowest detected pitch lag normally corresponds to the pitch of the signal frame. The algorithm detects other candidates at multiples of this pitch lag. In some parts of this signal (for example, between 3 and 4 s) the third harmonic of the real pitch is more dominant than the fundamental frequency. In these parts the lowest detected pitch lag is not the one to be chosen. In this time-domain approach the precision of the detected pitch lags can be improved if the greatest common divisor of the pitch-lag candidates is used. The algorithm computes only integer-valued pitch-lag candidates. LTP with a higher precision (non-integer pitch lag M) is presented in [LVKL96, KA90]. As in the FFT-based approach a post-processing should be applied to choose one of the detected candidates for each frame. For a more reliable pitch estimation both time and frequency domain approaches may be combined.

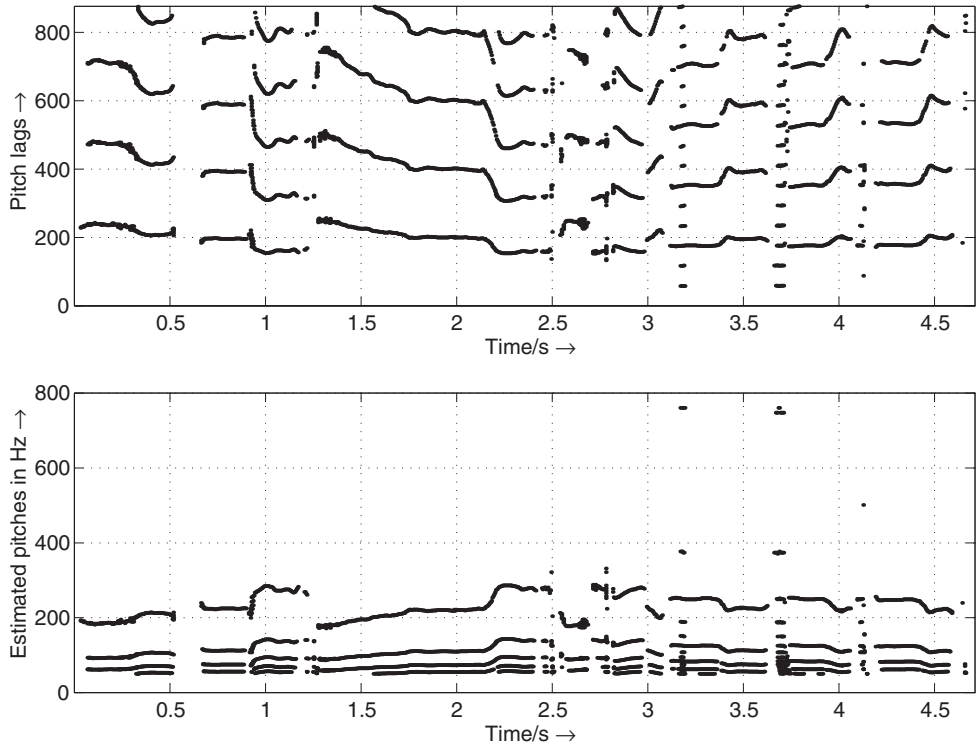

Figure 9.22 Pitch-lag candidates and corresponding frequencies.

The following M-file 9.12 presents an implementation of a pitch tracker based on the LTP approach.

M-file 9.12 (Pitch_Tracker_LTP.m)

```
% Pitch_Tracker_LTP.m    [DAFXbook, 2nd ed., chapter 9]
%===== This function demonstrates a pitch tracker based
%===== on the Long-Term Prediction

%----- initializations -----
fname='Toms_diner';
n0=2000; % start index
n1=210000;
K=200;   % hop size for time resolution of pitch estimation
N=1024;  % block length
% checked pitch range in Hz:
fmin=50;
fmax=800;
b0_thres=.2;      % threshold for LTP coeff
p_fac_thres=1.05; % threshold for voiced detection
                  % deviation of pitch from mean value

[xin,Fs]=wavread(fname,[n0 n0]); %get Fs
% lag range in samples:
```

```
lmin=floor(Fs/fmax);
lmax=ceil(Fs/fmin);
pre=lmax;   % number of pre-samples
if n0-pre<1
  n0=pre+1;
end
Nx=n1-n0+1; % signal length
blocks=floor(Nx/K);
Nx=(blocks-1)*K+N;
[X,Fs]=wavread(fname,[n0-pre n0+Nx]);
X=X(:,1)';

pitches=zeros(1,blocks);
for b=1:blocks
  x=X((b-1)*K+(1:N+pre));
  [M, F0]=find_pitch_ltp(x, lmin, lmax, N, Fs, b0_thres);
  if ~isempty(M)
    pitches(b)=Fs/M(1); % take candidate with lowest pitch
  else
    pitches(b)=0;
  end
end

%----- post-processing -----
L=9;              % number of blocks for mean calculation
if mod(L,2)==0   % L is even
  L=L+1;
end
D=(L-1)/2;        % delay
h=ones(1,L)./L;  % impulse response for mean calculation
% mirror start and end for "non-causal" filtering:
p=[pitches(D+1:-1:2), pitches, pitches(blocks-1:-1:blocks-D)];
y=conv(p,h);          % length: blocks+2D+2D
pm=y((1:blocks)+2*D); % cut result

Fac=zeros(1,blocks);
idx=find(pm~=0); % don't divide by zero
Fac(idx)=pitches(idx)./pm(idx);
ii=find(Fac<1 & Fac~=0);
Fac(ii)=1./Fac(ii); % all non-zero element are now > 1
% voiced/unvoiced detection:
voiced=Fac~=0 & Fac<p_fac_thres;

T=40;                 % time in ms for segment lengths
M=round(T/1000*Fs/K); % min. number of blocks in a row
[V,p2]=segmentation(voiced, M, pitches);
p2=V.*p2;  % set pitches to zero for unvoiced

figure(1),clf;
time=(0:blocks-1)*K+1; % start sample of blocks
time=time/Fs;          % time in seconds
t=(0:length(X)-1)/Fs;  % time in sec for original
subplot(211)
plot(t, X),title('original x(n)');
axis([0 max([t,time]) -1.1*max(abs(X)) 1.1*max(abs(X))])

subplot(212)
```

```
idx=find(p2~=0);
plot_split(idx,time, p2),title('pitch in Hz');
xlabel('time/s \rightarrow');
axis([0 max([t,time]) .9*min(p2(idx)) 1.1*max(p2(idx))])
```

The result of the presented pitch-tracking algorithm is illustrated in Figure 9.23. The bottom plot shows the pitch over time calculated using the LTP method. In comparison to the FFT-based approach in Figure 9.12, the FFT approach performs better in the regions where unvoiced parts occur. The described approach performs well for singing-voice examples. The selection of the post-processing strategy depends on the specific sound or signal.

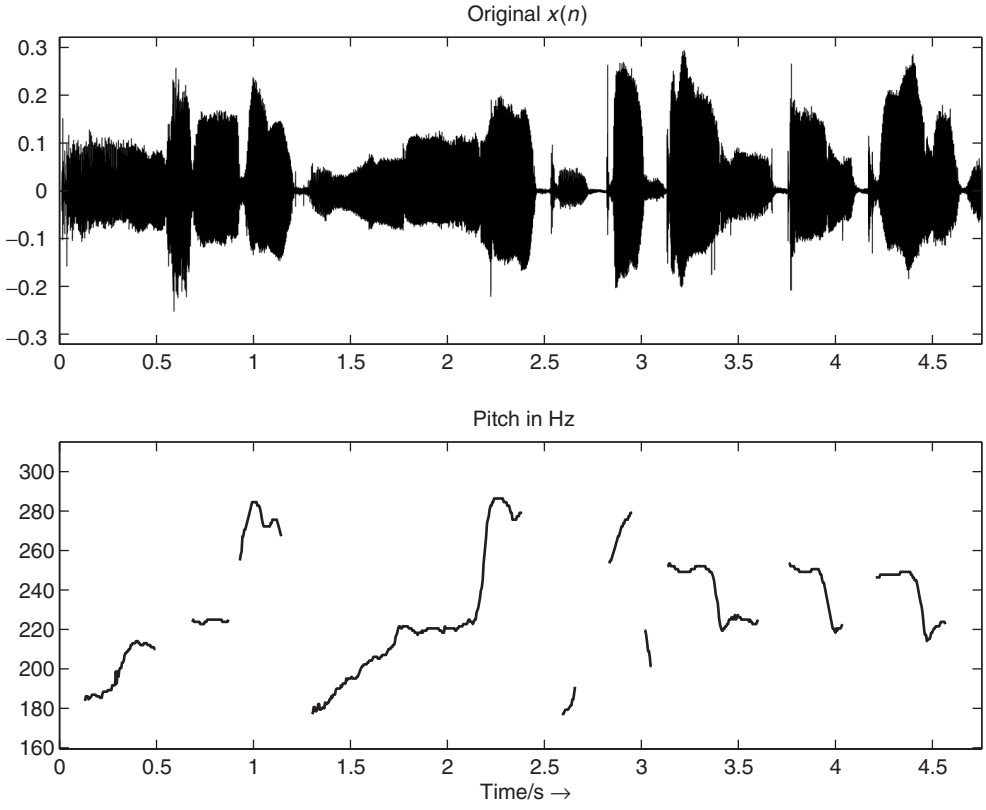

Figure 9.23 Pitch over time using the long-term prediction method for a signal segment of Suzanne Vega's 'Tom's Diner.'

9.2.5 Spatial hearing cues

Spatial hearing relates to source localization (in terms of distance, azimuth, elevation), motion (Doppler), and directivity, as well as to the room effect (reverberation, echo). Computational models that perform auditory scene analysis (ASA) rely on both inter-aural intensity (IID) and inter-aural time (ITD) differences [Bla83] in order to estimate the source azimuth. Elevation

and distance are more difficult to estimate, as they involve modeling knowing or estimating the head-related transfer function. While it is quite complex matter to properly deconvolve a signal in order to remove its reverberation, the same deconvolution techniques can perform a bit better to estimate a room echo. In any case, spatial-related sound features extracted from ASA software are good starting points for building an adaptive control that depends on spatial attributes.

9.2.6 Timbral features

Timbre is the most complex perceptual attribute for sound analysis. Here is a reminder of some of the sound features related to timbre:

- Brightness, correlated to spectral centroid [MWdSK95] – itself related to spectral slope, zero-crossing rate, and high frequency content [MB96] – and computed with various models [Cab99].
- Quality and noisiness, correlated to signal-to-noise ratio and to voiciness.
- Texture, related to jitter and shimmer of partials/harmonics [DT96a] (resulting from a statistical analysis of the partials' frequencies and amplitudes), to the balance of odd/even harmonics (given as the peak of the normalized auto-correlation sequence situated half way between the first- and second-highest peak values [AKZ02a]) and to harmonicity.
- Formants and especially vowels for the voice [Sun87] extracted from the spectral envelope; the spectral envelope of the residual; and the mel-frequency critical bands (MFCC), perceptual correlate of the spectral envelope.

Again, specialized software for computing perceptual models of timbre parameters are also good sources of sound features. For instance, PsySound performs the computation of brightness and roughness models, among others. For implementation purposes, sound features will be described by the family of signal-processing technique/domain used to compute them.

Auto-correlation features

We can extract important features from the auto-correlation sequence of a windowed signal: an estimation of the harmonic/non-harmonic content of a signal, the odd/even harmonics ratio in the case of harmonic sounds, and a voiced/unvoiced part in a speech signal. Several algorithms which determine whether a speech frame is voiced or unvoiced are known from speech research [Hes83]. Voiced/unvoiced detection is used either for speech recognition or for synthesis. For digital audio effects, such a feature is useful as a control parameter for an adaptive audio effect. The first peak value of the normalized auto-correlation sequence $r_{xx}(m)$ for $m > 0$ is a good indicator of the unvoiced or voiced part of a signal, as shown in Figure 9.24. When sounds are harmonic, the first auto-correlation peak ($m > 0$) on the abscissa corresponds to the pitch period of this sound. The value of this peak will be maximum if the sound is harmonic and minimum if the sound is noisy. If a window is used, which gives a better estimate for pitch extraction, this peak value will not go to one, but will be weighted by the auto-correlation of the two windows. This first peak value will be denoted pv at m_0 and is a good indication of voiced/noisy parts in a spoken or sung voice [BP89]. From then, voiciness is defined as the ratio between the amplitude of the first peak $m > 0$ divided by the auto-correlation for $m = 0$, which is the signal amplitude

$$\text{voiciness}(m) = \frac{pv}{r_{xx}(0)} = \frac{r_{xx}(m_0/2)}{r_{xx}(0)}. \tag{9.41}$$

In the case of harmonic sounds, it can be noted that the odd/even harmonics ratio can also be retrieved from the value at half of the time lag of the first peak. The percentage of even harmonics

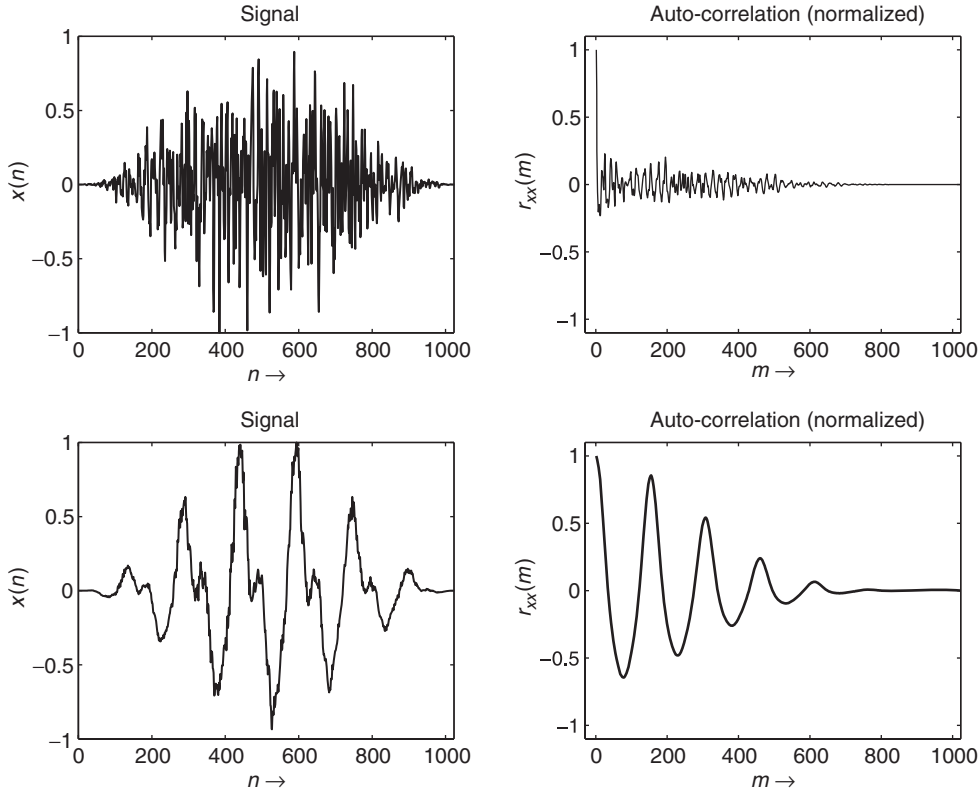

Figure 9.24 Unvoiced (upper part) and voiced (lower part) signals and the corresponding auto-correlation sequence $r_{xx}(m)$.

amplitudes is given as

$$\text{even}(m) = \frac{r_{xx}(m_0/2)}{r_{xx}(0)}. \tag{9.42}$$

An alternative computation of the auto-correlation sequence can be performed in the frequency domain [OS75]. Normally, the auto-correlation is computed from the power spectrum $|X(k)|^2$ of the input signal by $r_{xx}(m) = \text{IFFT}\left[|X(k)|^2\right]$. Here, we perform the IFFT of the magnitude $|X(k)|$ (square root of the power spectrum), which is computed from the FFT of a windowed signal. This last method is illustrated by the following M-file 9.13 that leads to a curve following the voiced/unvoiced feature, as shown in Figure 9.25.

M-file 9.13 (UX_voiced.m)

```
% UX_voiced.m

% feature_voice is a measure of the maximum of the second peak
% of the acf
clear;clf
%----- USER DATA -----
[DAFx_in, FS]  = wavread('x1.wav');
```

```
hop             = 256;  % hop size between two FFTs
WLen            = 1024; % length of the windows
w               = hanningz(WLen);
%----- some initializations -----
WLen2           = WLen/2;
tx              = (1:WLen2+1)';
normW           = norm(w,2);
coef            = (WLen/(2*pi));
pft             = 1;
lf              = floor((length(DAFx_in) - WLen)/hop);
feature_voiced = zeros(lf,1);
tic
%=========================================
pin  = 0;
pend = length(DAFx_in) - WLen;

while pin<pend
   grain    = DAFx_in(pin+1:pin+WLen).* w;
   f        = fft(grain)/WLen2;
   f2       = real(ifft(abs(f)));
   f2       = f2/f2(1);
   [v,i1]   = min(f2(1:WLen2)>0.);
   f2(1:i1) = zeros(i1,1);
   [v,imax] = max(f2(1:WLen2));
   feature_voiced(pft) = v;
   pft      = pft + 1;
   pin      = pin + hop;
end
% =========================================
toc
subplot(2,1,1)
plot(feature_voiced)
```

A particular way to use this feature is the construction of an adaptive time-stretching effect, where the stretching ratio α depends on this feature according to a mapping function $\alpha = 8^{pv}$ (see Figure 9.26). The time-stretching ratio will vary from 1 to 8, depending on the evolution of pv over time. A threshold detector can help to force this ratio to 1 in the case of silence. This leads to great improvements over regular time-stretching algorithms.

Center of gravity of a spectrum (spectral centroid)

An important feature of a sound is the evolution of the 'richness of harmonics' over time. It has been clearly pointed out at the beginning of computer music that sounds synthesized with a fixed waveform give only static sounds, and that the sound's harmonic content must evolve with time to give a lively sound impression. So algorithmic methods of synthesis have used this variation: additive synthesis uses the balance between harmonics or partials, waveshaping or FM synthesis use an index which changes the richness by the strength of the components.

A good indication of the instantaneous richness of a sound can be measured by the center of gravity of its spectrum, as depicted in Figure 9.27. A sound with a fixed pitch, but with stronger harmonics has a higher center of gravity. It should be noted here that this center of gravity is linked to the pitch of the sound, and that this should be taken into account during the use of this feature. Thus, a good indicator of the instantaneous richness of a sound can be the ratio of the center of gravity divided by the pitch.

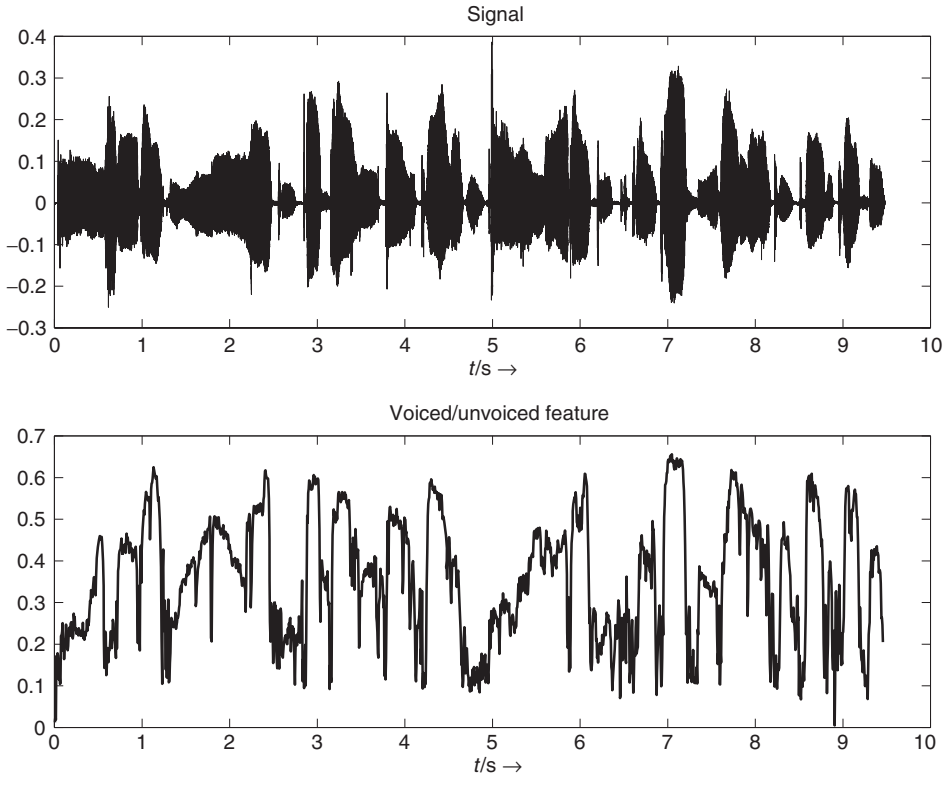

Figure 9.25 Vocal signal and the 'voiced/unvoiced' feature $pv(n)$.

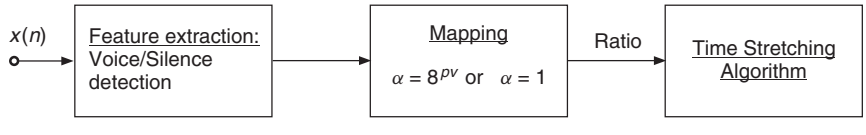

Figure 9.26 Adaptive time stretching based on auto-correlation feature.

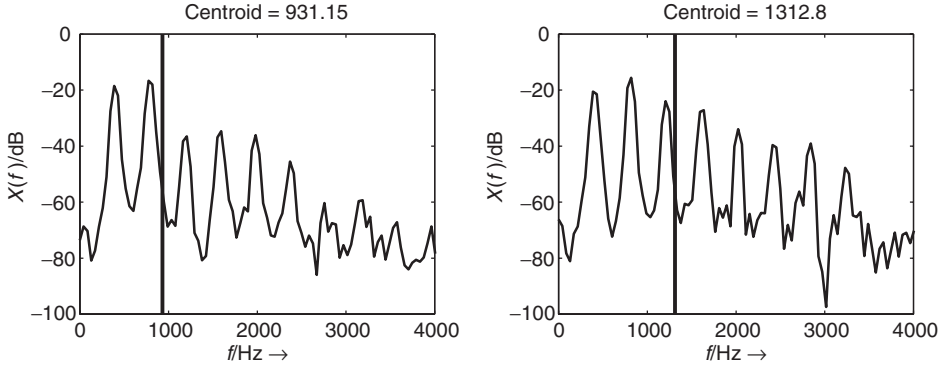

Figure 9.27 Center of gravity of a spectrum as a good indicator of the richness of a harmonic sound.

A straightforward method of calculating this centroid can be achieved inside an FFT/IFFT-based analysis-synthesis scheme. The spectral centroid is at the center of the spectral energy distribution and can be calculated by

$$\text{centroid}_a = \frac{\sum_{k=0}^{N/2} k \cdot |X(k)|}{\sum_{k=0}^{N/2} |X(k)|}. \tag{9.43}$$

The centroid is defined by the ratio of the sum of the magnitudes multiplied by the corresponding frequencies divided by the sum of the magnitudes and it is also possible to use the square of the magnitudes

$$\text{centroid}_e = \frac{\sum_{k=0}^{N/2} k \cdot |X(k)|}{\sum_{k=0}^{N/2} |X(k)|^2}. \tag{9.44}$$

Another method working in the time domain makes use of the property of the derivative of a sinusoid which gives $\frac{\mathrm{d}}{\mathrm{d}n} A_k \sin(\Omega_k n) = A_k \Omega_k \cdot \cos(\Omega_k n)$ with $\Omega_k = 2\pi \frac{f_k}{f_S}$. If we can express the input signal by a sum of sinusoids according to

$$x(n) = \sum_{k=0}^{N/2-1} A_k \sin(\Omega_k n), \tag{9.45}$$

the derivative of the input signal leads to

$$\frac{\mathrm{d}x(n)}{\mathrm{d}n} = \sum_{k=0}^{N/2-1} A_k \Omega_k \cos(\Omega_k n). \tag{9.46}$$

The spectral centroid can then be computed according to (9.43) by the ratio of the RMS value of the derivative of the input signal divided by the RMS value of the input signal itself. The derivative of the discrete-time input signal $x(n)$ can be approximated by $\Delta x(n) = x(n) - x(n-1)$. The described time-domain method is quite effective because it does not need any FFT and is suitable for real-time applications. The following M-file 9.14 illustrates these possibilities.

M-file 9.14 (UX_centroid.m)

```
% UX_centroid.m
[DAFXbook, 2nd ed., chapter 9]
% feature_centroid1 and 2 are centroids
% calculate by two different methods
clear;clf
%----- USER DATA -----
[DAFx_in, FS]  = wavread('x1.wav');
hop            = 256;  % hop size between two FFTs
WLen           = 1024; % length of the windows
w              = hanningz(WLen);
%----- some initializations -----
WLen2          = WLen/2;
tx             = (1:WLen2+1)';
normW          = norm(w,2);
coef           = (WLen/(2*pi));
pft            = 1;
lf             = floor((length(DAFx_in) - WLen)/hop);
feature_rms    = zeros(lf,1);
```

```
feature_centroid  = zeros(lf,1);
feature_centroid2 = zeros(lf,1);
tic
%=========================================
pin  = 0;
pend = length(DAFx_in) - WLen;

while pin<pend
   grain                  = DAFx_in(pin+1:pin+WLen).* w;
   feature_rms(pft)       = norm(grain,2) / normW;
   f                      = fft(grain)/WLen2;
   fx                     = abs(f(tx));
   feature_centroid(pft)  = sum(fx.*(tx-1)) / sum(fx);
   fx2                    = fx.*fx;
   feature_centroid2(pft) = sum(fx2.*(tx-1)) / sum(fx2);
   grain2                 = diff(DAFx_in(pin+1:pin+WLen+1)).* w;
   feature_deriv(pft)     = coef * norm(grain2,2) / norm(grain,2);
   pft                    = pft + 1;
   pin                    = pin + hop;
end
% =========================================
toc
subplot(4,1,1); plot(feature_rms); xlabel('RMS')
subplot(4,1,2); plot(feature_centroid); xlabel('centroid 1')
subplot(4,1,3); plot(feature_centroid2); xlabel('centroid 2')
subplot(4,1,4); plot(feature_deriv); xlabel('centroid 3')
```

For each method the center of gravity is calculated in frequency bins. Figure 9.28 illustrates the results for each method. It can be seen at the end of a flute sound that the centroid parameter is very important: the variations of the centroid are quite independent of the RMS values of the signal. The centroid takes some time to oscillate and then is maintained until the end of the sound. An alternative has been proposed, so that the centroid tends to 0 when the sound intensity level also tends to 0. It consists of adding a constant of small value in the centroid denominator [Bea82]:

$$\text{centroid}_{a,2} = \frac{\sum_{k=0}^{N/2} k \cdot |X(k)|}{c_0 + \sum_{k=0}^{N/2} |X(k)|}. \tag{9.47}$$

An artifact of this computation is that it alway biases the centroid value from the usual formulation one, whereas the following computation only biases for low signal intensity levels:

$$\text{centroid}_{a,2} = \frac{\sum_{k=0}^{N/2} k \cdot |X(k)|}{\max(c_0, \sum_{k=0}^{N/2} |X(k)|)}. \tag{9.48}$$

A digital effect which relies on the spectral centroid feature is the mimicking of a natural sound by a synthetic one. As an example, we can use a waveshaping synthesis method. This method calculates a synthetic sound by distorting a sine wave with the help of a waveshaping function (also called nonlinear transfer function) and multiplying the result by an amplitude variation (see Figure 9.29). The waveshaping function is usually a polynomial, because this function can be calculated with the aim of having a fixed output spectrum [Arf79, Bru79]. So apart from the pitch

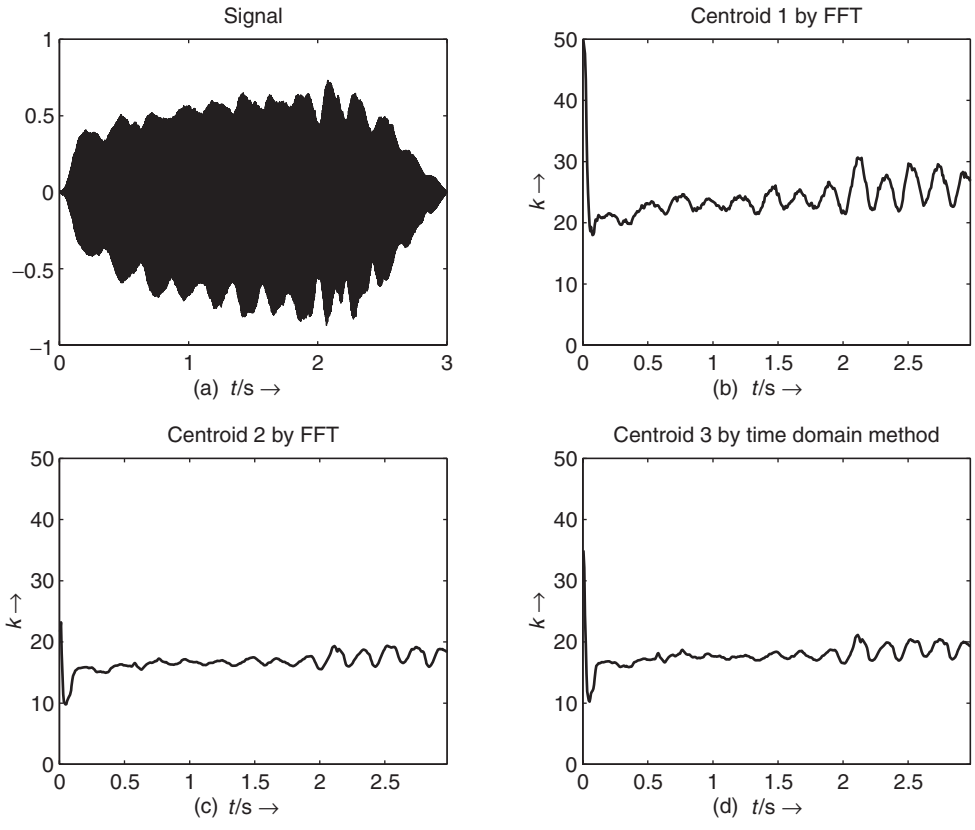

Figure 9.28 Spectral centroid computation in frequency and time domain. Plots (b)–(d) show the centroids in bins, where the corresponding tone pitch is given by $f_k = \frac{k}{N} f_S$ (with FFT length N and sampling frequency f_S).

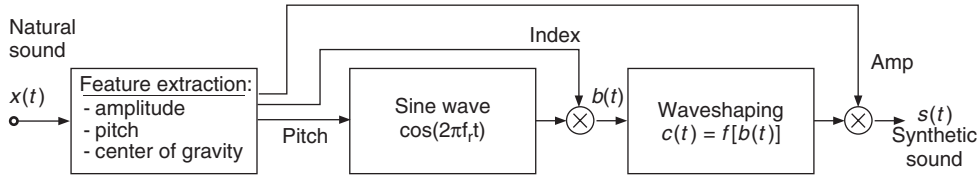

Figure 9.29 Mimicking with waveshaping.

of the sine wave, this method relies on the evolution of two parameters: the amplitude of the sine wave, which is called the 'index' because the sine wave is used as an input to the waveshaping function, and an amplitude factor, which is used as an amplitude envelope. If we extract the centroid and the RMS evolution from a natural sound, as well as the pitch, we can compute an index proportional to the ratio of the centroid towards the pitch (this proportional factor, apart from a necessary normalization, drives the general brightness of the sound) and an amplitude factor proportional to the extracted RMS feature. Thus we obtain a synthetic sound that retains

some characteristics of the initial sound and is given by

$$s(t) = \text{amp} \cdot f\left[\text{index} \cdot \cos(2\pi f_r t)\right] \tag{9.49}$$

$$f_r = \text{pitch}, \text{index} = \text{coef} \cdot \frac{\text{centroid}}{\text{pitch}}, \text{amp} = \text{rms}.$$

The mimicking is improved when the waveshaping function is calculated for the spectrum at one point of the initial sound. Two other further improvements can be added: an amplitude normalization factor due to the fact that the index should only change the centroid and not the amplitude, and a correcting factor for the index due to the fact that the index and centroid of the synthetic sound have no reason to be in a linear relationship. But even the simple process we have described gives a variety of allotropic sounds which all resemble the original in some way, but are purely harmonic and do not contain any noisy components.

Features from harmonic partials

Once a spectral model of a harmonic sound is computed, various sound features can be extracted from the harmonic partial amplitude and frequency, apart from the harmonic centroid we have already presented. The relative percentage of amplitude or energy of the even and odd harmonics can be computed as

$$\%A_{even}(m) = \frac{\sum_{l=1}^{L/2} a_{2l}(m)}{\sum_{l=1}^{L} a_l(m)} \tag{9.50}$$

$$\%A_{odd}(m) = \frac{\sum_{l=0}^{L/2} a_{2l+1}(m)}{\sum_{l=1}^{L} a_l(m)}. \tag{9.51}$$

These features are called *even/odd harmonic ratios*. Such computation is more accurately than the one proposed with the auto-correlation function, because the latter computes the amplitude of even harmonics relative to the whole sound amplitude, not only that of the harmonic partials. Alternatives are the ratio of odd versus even harmonics and the ratio of even versus odd harmonics.

The *tristimulus* is a timbral descriptor that was derived for sounds as an equivalent of the color attributes for vision [PJ82]. It is computed by

$$T_1(m) = \frac{a_1(m)}{\sum_{l=1}^{L} a_l(m)} \tag{9.52}$$

$$T_2(m) = \frac{\sum_{l=2}^{4} a_l(m)}{\sum_{l=1}^{L} a_l(m)} \tag{9.53}$$

$$T_3(m) = \frac{\sum_{l=5}^{L} a_l(m)}{\sum_{l=1}^{L} a_l(m)}. \tag{9.54}$$

From these equations, it follows that $T_1(m)$ describes the strength of fundamental amplitude compared to all harmonics. $T_2(m)$ describes how strong are the middle frequency harmonics and $T_3(m)$ describes how strong are the higher harmonic amplitudes. As for the spectral centroid, it can be modified using energies instead of amplitudes, and truncating the denominator for low-level signals, in which case the tristimuli becomes:

$$\overline{T}_1(m) = \frac{(a_1(m))^2}{\max\left(\varepsilon, \sum_{l=1}^{L} (a_l(m))^2\right)} \tag{9.55}$$

$$\overline{T}_2(m) = \frac{\sum_{l=2}^{4} (a_l(m))^2}{\max\left(\varepsilon, \sum_{l=1}^{L} (a_l(m))^2\right)} \tag{9.56}$$

$$\overline{T}_3(m) = \frac{\sum_{l=5}^{L} (a_l(m))^2}{\max\left(\varepsilon, \sum_{l=1}^{L} (a_l(m))^2\right)}. \tag{9.57}$$

9.2.7 Statistical features

As an example, Dubnov [DT96b] has obtained the classification of instrumental timbres on a 2-D map by using skew and kurtosis. These features can help in defining the texturization of a sound. The texture of a sound is very difficult to evaluate: why a trumpet does not sound like a string does not rely only on a spectral richness. The way the individual components are synchronized or not is an important key for defining a texture. Many other features can be extracted from a sound [HB98, RDS+99, DH00], just to mention a few from a very active field of research.

9.3 Mapping sound features to control parameters

9.3.1 The mapping structure

While recent studies define specific strategies of mapping for gestural control of sound synthesizers [Wan02] or audio effects [WD00], some mapping strategies for ADAFX were specifically derived from the three-layer mapping that uses a perceptive layer [ACKV02, AV03, VWD06], which is shown in Figure 9.30. To convert sound features $f_i(n), i = 1, ..., M$ into effect control parameters $c_j(n), j = 1, ..., N$, we use an M-to-N explicit mapping scheme[3] divided into two stages: sound-feature combination and control-signal conditioning (see Figure 9.30).

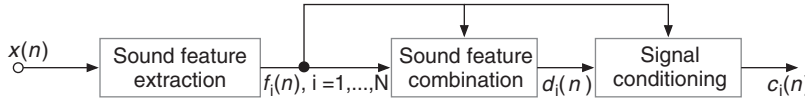

Figure 9.30 Mapping structure between sound features and one effect control $c_i(n)$: sound features are first combined, and then conditioned in order to provide a valid control to the effect. Figure reprinted with IEEE permission from [VZA06].

The sound features may often vary rapidly and with a constant sampling rate (synchronous data), whereas the gestural controls used in sound synthesis vary less frequently and sometimes in an asynchronous mode. For that reason, we chose sound features for direct control of the effect and optional gestural control for modifications of the mapping between sound features and effect control parameters [VWD06], thus providing navigation by interpolation between presets.

Defining a clear mapping structure offers a higher level of control and generalizes any effect: with adaptive control (remove the gestural control level), with gestural control (remove the adaptive control), or with both controls. Sound features are either short-term or long-term features; therefore they may have different and well-identified roles in the proposed mapping structure. Short-term features (e.g., energy, instantaneous pitch or loudness, voiciness, spectral centroid) provide a continuous adaptive control with a high rate that we consider equivalent to a modification gesture [Cad99] and useful as inputs (left horizontal arrows in Figures 9.31 and 9.32). Long-term features

[3] M is the number of features we use, usually between 1 and 5; N is the number of effect control parameters, usually between 1 and 20.

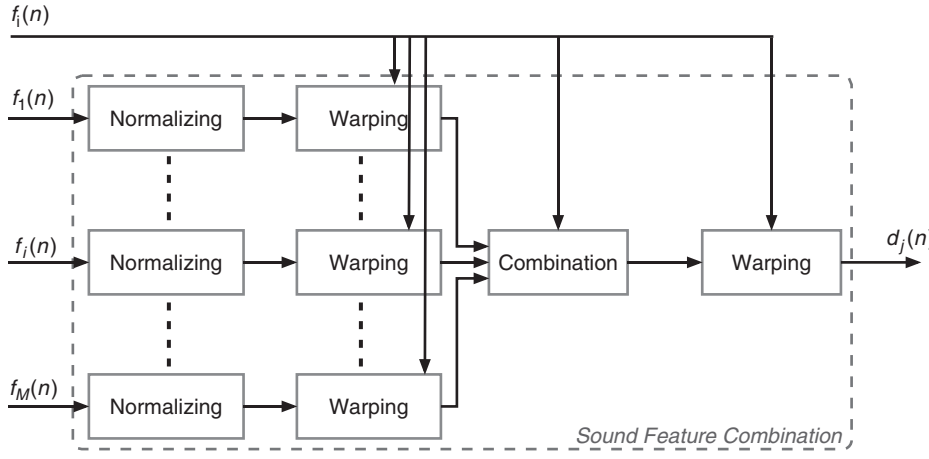

Figure 9.31 Feature combination, first stage of the sound-feature mapping. $f_i(n), i = 1, ..., M$ are the sound features, and $d_j(n), j = 1, ..., N$ are the combined features. Figure reprinted with IEEE permission from [VZA06].

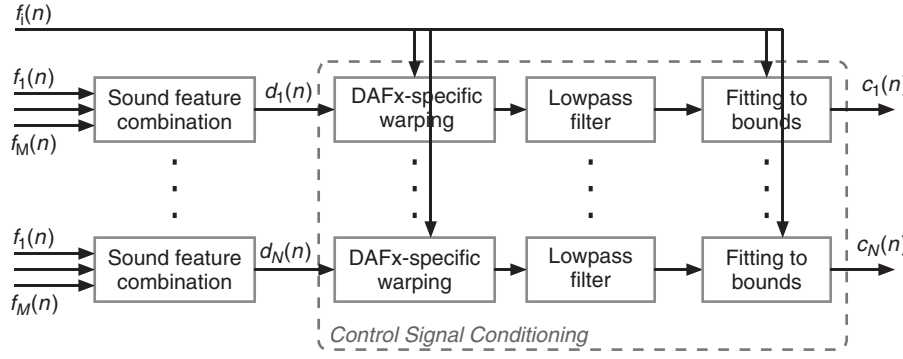

Figure 9.32 Signal conditioning, the second stage of the sound feature mapping. $c_i(n), n = 1, ..., N$ are the effect controls derived from sound features $f_i(n), i = 1, ..., M$. The DAFx-specific warping and the fitting to boundaries can be controlled by other sound features. Figure reprinted with IEEE permission from [VZA06].

computed after signal segmentation (e.g., vibrato, roughness, duration, note pitch, or loudness) are often used for content-based transformations [ABL+03]. They provide a sequential adaptive control with a low rate that we consider equivalent to a selection gesture, and that is useful for control of the mapping (upper vertical arrows in Figures 9.31 and 9.32).

9.3.2 Sound-feature combination

The first stage combines several features, as depicted in Figure 9.31. First, all the features are normalized in [0, 1] for unsigned values features and in [−1, 1] for signed value features. Second, a warping function – a transfer function that is not necessarily linear – can then be applied: a truncation of the feature in order to select an interesting part, low pass filtering, a scale change

(from linear to exponential or logarithmic), or any non-linear transfer function. Parameters of the warping function can also be derived from sound features (for example the truncation boundaries). Third, the feature combination is done by linear combination, except when weightings are derived from other sound features. Fourth and finally, a warping function can also be applied to the feature combination output in order to symmetrically provide modifications of features before and after combination.

9.3.3 Control-signal conditioning

Conditioning a signal consists of modifying the signal so that its behavior fits to pre-requisites in terms of boundaries and variation type. It is usually used to protect hardware from an input signal. The second mapping stage conditions the effect control signal $d_i(n)$ coming out from the feature combination box, as shown in Figure 9.32, so that it fits the required behavior of the effect controls. It uses three steps: an effect-specific warping, a low pass filter and scaling. First, the specific warping is effect dependent. It may consist of quantizing the pitch curve to the tempered scale (auto-tune effect), quantizing the control curve of the delay time (adaptive granular delay, cf. Section 9.4.6), or modifying a time-warping ratio varying with time in order to preserve the signal length (cf. Section 9.4.2). Second, the low pass filter ensures the suitability of the control signal for the selected application. Third and last, the control signal is scaled to the effect control boundaries given by the user, that are eventually adaptively controlled. When necessary, the control signal, sampled at the block rate is resampled to the audio sampling rate f_S.

9.4 Examples of adaptive DAFX

As explained in this chapter's introduction, several effects already made use of adaptive control a long time before recent studies tended to generalize this concept to all audio effects [VA01, ABL+03]. Such effects were developed for technical or musical purposes, as answers to specific needs (e.g., auto-tune, compressor, cross-synthesis). Using the framework presented in this chapter, we can now ride on the adaptive wave, and present old and new adaptive effects in terms of: their form, the sound features that are used, the mapping of sound features to effect control parameters, the modifications of implementation techniques that are required, and the perceptual effect of the ADAFX.

9.4.1 Adaptive effects on loudness

Adaptive sound-level change

By controlling the sound level from a sound feature, we perform an amplitude modulation with a signal which is not monosinusoidal. This generalizes compressor/expander, noise gate/limiter, and cross-duking, which all are specific cases in terms of sound-feature extraction and mapping. Adaptive amplitude modulation controlled by $c(n)$ provides the following signal

$$y(n) = x(n) \cdot (1 + c(n)) \,. \tag{9.58}$$

By deriving $c(n)$ from the sound-intensity level, one obtains compressor/expander and noise gate/limiter. By using the voiciness $v(n) \in [0, 1]$ and a $c(n) = (1 - \cos[\pi\, v(n)])\,/2$ mapping law, one obtains a 'voiciness gate,' which is a timbre effect that removes voicy sounds and leaves only noisy sounds (which differs from the de-esser [DZ02] that mainly removes the 's'). Adaptive sound-level change is also useful for attack modification of instrumental and electroacoustic sounds (differently from compressor/expander), thus modifying loudness and timbre. If using a computational model of perceived loudness, one can make a feedforward adaptive loudness change, that

tries to reach a target loudness by means of iterative computations on the output loudness depending on the gain applied.

Adaptive tremolo

A tremolo, of monosinusoidal amplitude modulation, is an effect for which either the rate (or modulation frequency $f_m(n)$) and the depth $d(n)$, or both, can be derived from sound features. The time-varying amplitude modulation is then expressed using a linear scale

$$c(n) = d(n) \cdot \sin\left(2\pi \frac{f_m(n)}{f_S} n\right), \tag{9.59}$$

where f_S the audio sampling rate. It may also be expressed using a logarithmic scale according to

$$c_{dB}(m) = 10^{d(m)\left(\sin\left(2\pi \frac{f_m(n)}{f_S} n\right)-1\right)/40}. \tag{9.60}$$

The modulation function is sinusoidal, but may be replaced by any other periodic function (e.g., triangular, exponential, logarithmic or other). The real-time implementation only requires an oscillator, a warping function, and an audio rate control. Adaptive tremolo allows for a more natural tremolo that accelerates/slows down (which also sounds like a rhythm modification) and emphasizes or de-emphasizes, depending on the sound content (which is perceived as a loudness modification). Figure 9.33 shows an example where the fundamental frequency $f_0(m) \in [780, 1420]$ Hz and the sound-intensity level $a(m) \in [0, 1]$ are mapped to the control rate and the depth according to the following mapping rules

$$f_m(m) = 1 + 13 \cdot \frac{1420 - f_0(m)}{1420 - 780} \tag{9.61}$$

$$d(m) = 100 \cdot a(m). \tag{9.62}$$

9.4.2 Adaptive effects on time

Adaptive time-warping

Time-warping is non-linear time-scaling. This non-real-time processing uses a time-scaling ratio $\gamma(m)$ that varies with m the block index. The sound is then alternatively locally time-expanded when $\gamma(m) > 1$, and locally time-compressed when $\gamma(m) < 1$. The adaptive control is provided with the input signal (feedforward adaption). The implementation can be achieved either using a constant analysis step increment R_A and a time-varying synthesis step increment $R_S(m)$, or using a time-varying $R_A(m)$ and constant R_S, thus providing more implementation efficiency. In the latter case, the recursive formulae of the analysis time index t_A and the synthesis time index t_S are given by

$$t_A(m) = t_A(m-1) + R_A(m-1) \tag{9.63}$$

$$t_S(m) = t_S(m-1) + R_S, \tag{9.64}$$

with the analysis step increment according to

$$R_A(m-1) = \gamma(t_A(m-1)) \cdot R_S. \tag{9.65}$$

Adaptive time-warping provides improvement to the usual time-scaling, for example by minimizing the timbre modification. It allows for time-scaling with attack preservation when using an attack/transient detector to vary the time-scaling ratio [Bon00, Pal03].

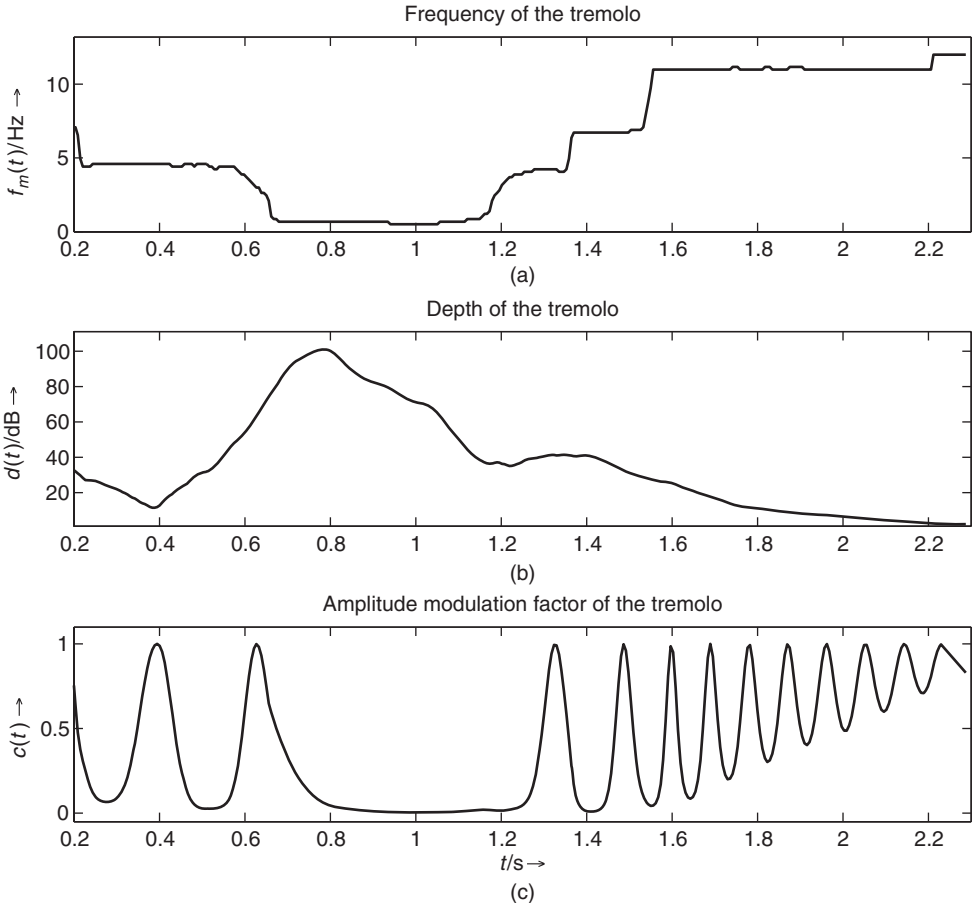

Figure 9.33 Control curves for the adaptive tremolo: (a) frequency $f_m(n)$ is derived from the fundamental frequency, as in Equation (9.61); (b) depth $d(m)$ is derived from the signal intensity level, as in Equation (9.62); (c) the amplitude modulation curve is computed with a logarithmic scale, as in Equation (9.60). Figure reprinted with IEEE permission from [VZA06].

Using auto-adaptive time-warping, we can apply fine changes in duration. A first example consists of time-compressing the gaps and time-expanding the sounding parts: the time-warping ratio γ is computed from the intensity level $a(n) \in [0, 1]$ using a mapping law such as $\gamma(n) = 2^{a(n)-a_0}$, with a_0 as a threshold. A second example consists of time-compressing the voicy parts and time-expanding the noisy parts of a sound, using the mapping law $\gamma(n) = 2^{v(n)-v_0}$, with $v(n) \in [0, 1]$ the voiciness and v_0 the voiciness threshold. When used for local changes of duration, it provides modifications of timbre and expressiveness by modifying the attack, sustain and decay durations. Using cross-adaptive time-warping, time-folding of sound A is slowed down or sped up depending on the sound B content. Generally speaking, adaptive time-warping allows for a re-interpretation of recorded sounds, for modifications of expressiveness (music) and perceived emotion (speech). Unfortunately, there is a lack of knowledge of and investigation into the link between sound features and their mapping to the effect control on one side, and the modifications of expressiveness on the other side.

Adaptive time-warping that preserves signal length

When applying time-warping with adaptive control, the signal length is changed. To preserve the original signal length, we must evaluate the length of the time-warped signal according to the adaptive control curve. Then, a specific mapping function is applied to the time-warping ratio γ so that it verifies the synchronization constraint.

Synchronization constraint Time indices in Equations (9.63) and (9.64) are functions that depend on γ and m as given by

$$t_A(m) = \sum_{l=1}^{m} \gamma(t_A(l-1))R_S \tag{9.66}$$

$$t_S(m) = m\ R_S, \quad 1 \le m \le M. \tag{9.67}$$

The analysis signal length $L_A = \sum_{l=1}^{M} \gamma(t_A(l-1))\ R_S$ differs from the synthesis signal length $L_S = MR_S$. This is no more the case for $\tilde{\gamma}$ verifying the synchronization constraint

$$M = \sum_{l=1}^{M} \tilde{\gamma}(t_A(l-1)). \tag{9.68}$$

Three synchronization schemes The constraint ratio $\tilde{\gamma}(t_A(m))$ can be derived from γ by:

(1) Addition as

$$\tilde{\gamma}_1(t_A(m)) = \gamma(t_A(m)) + 1 - \frac{\sum_{l=1}^{M} \gamma(t_A(l-1))}{M} \tag{9.69}$$

(2) By multiplication as

$$\tilde{\gamma}_2(t_A(m)) = \gamma(t_A(m)) \cdot M / \sum_{l=1}^{M} \gamma(t_A(l-1)) \tag{9.70}$$

(3) By exponential weighting $\tilde{\gamma}_3(t_A(m)) = \left[\gamma(t_A(m))\right]^{\gamma_3}$, with γ_3 the iterative solution[4] of

$$\gamma_3 = \arg\min_{p} \left| \frac{\sum_{l=1}^{M} \left[\gamma(t_A(l-1))\right]^{p}}{M} - 1 \right|. \tag{9.71}$$

An example is provided in Figure 9.34. Each of these modifications imposes a specific behavior on the time-warping control. For example, the exponential weighting is the only synchronization technique that preserves the locations where the signal has to be time-compressed or expanded: $\tilde{\gamma} > 1$ when $\gamma > 1$ and $\tilde{\gamma} < 1$ when $\gamma < 1$.

Synchronization that preserves $\boldsymbol{\gamma}$ boundaries However, none of these three synchronization schemes ensure to preserve $\boldsymbol{\gamma}_{\mathbf{min}}$ and $\boldsymbol{\gamma}_{\mathbf{max}}$, the boundaries of $\boldsymbol{\gamma}$ as given by the user. A solution consists in using the clipping function

$$\overline{\boldsymbol{\gamma}}(p) = \begin{cases} \tilde{\boldsymbol{\gamma}}_i(p) & \text{if } \tilde{\boldsymbol{\gamma}}_i(p) \in [\boldsymbol{\gamma}_{\mathbf{min}}; \boldsymbol{\gamma}_{\mathbf{max}}] \\ \boldsymbol{\gamma}_{\mathbf{min}} & \text{if } \tilde{\boldsymbol{\gamma}}_i(p) < \boldsymbol{\gamma}_{\mathbf{min}} \\ \boldsymbol{\gamma}_{\mathbf{max}} & \text{if } \tilde{\boldsymbol{\gamma}}_i(p) > \boldsymbol{\gamma}_{\mathbf{max}} \end{cases}. \tag{9.72}$$

[4] There is no analytical solution, so an iterative scheme is necessary.

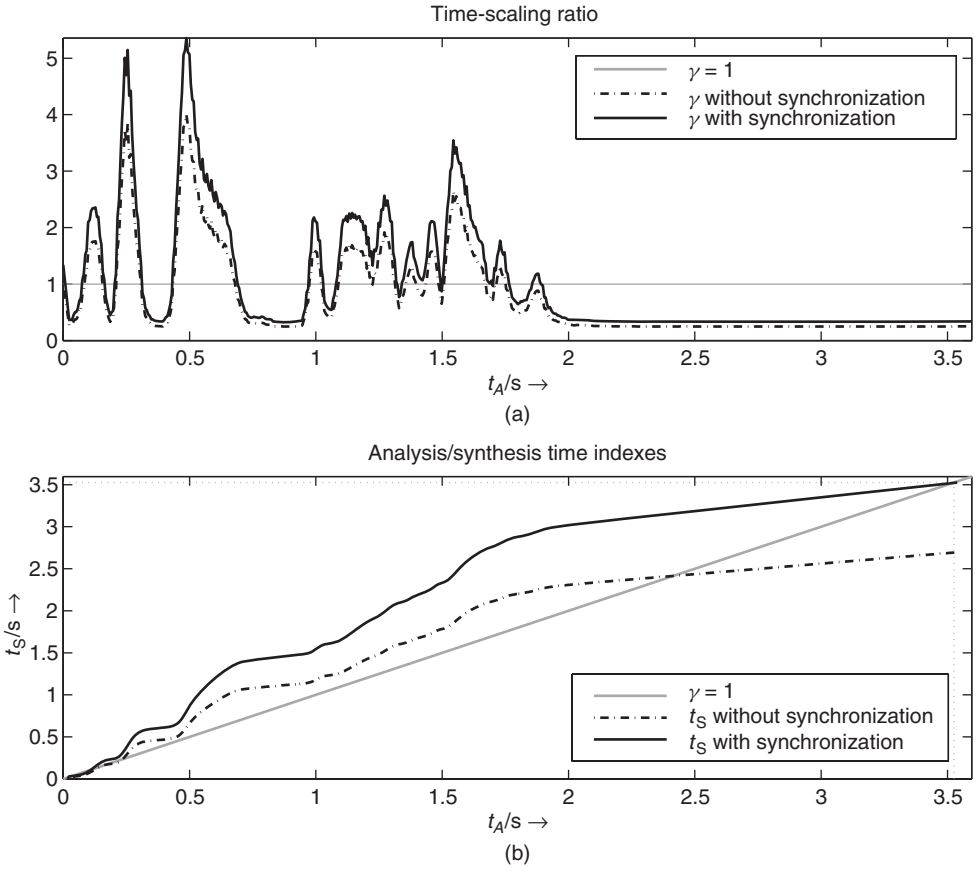

Figure 9.34 (a) The time-warping ratio γ is derived from the amplitude (RMS) as $\gamma(m) = 2^{4(a(m)-0.5)} \in [0.25, 4]$ (dashed line), and modified by the multiplication ratio $\gamma_m = 1.339$ (solid line). (b) The analysis time index $t_A(m)$ is computed according to Equation (9.63), verifying the synchronization constraint of Equation (9.68).

The iterative solution $\overline{\overline{\gamma}}_i$ that both preserves the synchronization constraint of Equation (9.68) and the initial boundaries can be derived as

$$\overline{\overline{\gamma}}_i = \arg\min_{\overline{\gamma}_i} \left| \frac{\sum_{l=1}^{M} \overline{\gamma}_i(t_A(l-1))}{M} - 1 \right|, \tag{9.73}$$

where $i = 1, 2, 3$ respectively denotes addition, multiplication, and exponential weighting. The adaptive time-warping that preserves the signal length can be used for various things, among which expressiveness change (slight modifications of a solo improvisation that still sticks to the beat), a more natural chorus (when combined with adaptive pitch-shifting), or a swing change.

Swing change

Adaptive time-warping that preserves the signal length provides groove change when giving several synchronization points [VZA06]. When time-scaling is controlled by an analysis from a beat

tracker, synchronization points can be beat-dependent and then the swing can be changed [GFB03] by displacing one every two beats either to make them even (binary rhythm) of uneven with the rule of thumb 2/3 and 1/3 (ternary rhythm). This effect modifies time and rhythm.

Time-scaling that preserves the vibrato and tremolo

Adaptive time-scaling also allows for time-scaling sounds with vibrato, for instance when combined with adaptive pitch-shifting controlled by a vibrato estimator: vibrato is removed, the sound is time-scaled, and vibrato with same frequency and depth is applied [AD98]. Using the special sinusoidal model [MR04], both vibrato (often modeled and reduced as frequency modulation) and tremolo (amplitude modulation) can be preserved by applying the resampling of control parameters on the partials of partials. This allows one to time-scale the portion of the sound which would be considered as 'sustained,' while leaving untouched the attack and release.

9.4.3 Adaptive effects on pitch

Adaptive pitch-shifting

Using any of the techniques proposed for the usual pitch-shifting with formant preservation (PSOLA, phase vocoder combined with source-filter separation, and additive model), adaptive pitch-shifting can be performed in real time, with its pitch-shift ratio $\rho(m)$ defined in the middle of the block as

$$\rho(m) = \frac{F_{0,\text{out}}(m)}{F_{0,\text{in}}(m)} \tag{9.74}$$

where $F_{0,\text{in}}(m)$ (resp. $F_{0,\text{out}}(m)$) denotes the fundamental frequency of the input (resp. the output) signal. The additive model allows for varying pitch-shift ratios, since the synthesis can be made sample by sample in the time domain [MQ86]. The pitch-shifting ratio is then interpolated sample by sample between two blocks. PSOLA allows for varying pitch-shifting ratios as long as one performs at the block level and performs energy normalization during the overlap-add technique. The phase vocoder technique has to be modified in order to permit that two overlap-added blocks have the same pitch-shifting ratio for all the samples they share, thus avoiding phase cancellation of overlap-added blocks. First, the control curve must be low pass filtered to limit the pitch-shifting ratio variations. In doing so, we can consider that the spectral envelope does not vary inside a block, and then use the source-filter decomposition to resample only the source. Second, the variable sampling rate $F_A(m)$ implies a variable length of the synthesis block $N_S(m) = N_A/\rho(m)$ and so a variable energy of the overlap-added synthesis signal. The solution we chose consists in imposing a constant synthesis block size $N_c = N_U/\max \rho(m)$, either by using a variable analysis block size $N_A(m) = \rho(m)N_U$ and then $N_S = N_U$, or by using a constant analysis block size $N_A = N_U$ and post-correcting the synthesis block x_{ps} according to

$$y(n) = x_{ps}(n) \cdot \frac{h_{N_c}(n)}{w_{N(m)}(n)}, \tag{9.75}$$

where h is the Hanning window, $N(m) = 1 + \sum_{l=2}^{N_A} \rho(l)$ is the number of samples of the synthesis block m, $x_{ps}(n)$ is the resampled and formant-corrected block $n = 1, ..., N(m)$, w is the warped analysis window defined for $n = 1, ..., N(m)$ as $w_{N(m)}(n) = h(\sum_{l=1}^{n} \rho_s(l))$, and $\rho_s(n)$ is the pitch-shifting ratio $\rho(m)$ resampled at the signal sampling rate F_A.

A musical application of adaptive pitch-shifting is adaptive detuning, obtained by adding to a signal its pitch-shifted version with a lower than a quarter-tone ratio (this also modifies timbre). For instance, adaptive detuning can be controlled by the amplitude as $\rho(n) = 2^{0.25 \cdot a(n)}$, where louder sounds are the most detuned. Adaptive pitch-shifting allows for melody change when controlled by

long-term features such as the pitch of each note of a musical sentence [GPAH03]. The auto-tune is a feedback-adaptive pitch-shifting effect, where the pitch is shifted so that the processed sound reaches a target pitch. Adaptive pitch-shifting is also useful for intonation change, as explained below.

Harmonizer

As it will be further explained in Chapter 10, a vocal chorus can be harmonized by adding pitch-shifted versions of a voice sound. These pitch-shifted versions are tuned to one given target degree in the current musical scale, so that chords of voices are obtained. This is very similar to the auto-tune (feedback-adaptive effect), except that the target pitch is not the pitch in the scale which is nearest the original pitch, but a pitch with a specific relation to the original voice pitch. This effect, combined with other effects such as auto-tune, reverberation, etc., has been widely used in recent products such as the Vocaloid by Yamaha, and the VoiceOne by TC Helicon.

Adaptive intonation change

Intonation is the pitch information contained in prosody of human speech. It is composed of the macro-intonation and the micro-intonation [Cri82]. To compute these two components, the fundamental frequency $F_{0,\text{in}}(m)$ is segmented over time. Its local mean $\overline{F_{0,\text{in}}^{loc}}$ is the macro-intonation structure for a given segment, and the remainder $\Delta F_{0,\text{in}}(m) = F_{0,\text{in}}(m) - \overline{F_{0,\text{in}}^{loc}}$ is the micro-intonation structure,[5] as shown in Figure 9.35. This yields the following decomposition of the input fundamental frequency

$$F_{0,\text{in}}(m) = \overline{F_{0,\text{in}}^{loc}} + \Delta F_{0,\text{in}}(m). \tag{9.76}$$

The adaptive intonation change is a non-real-time effect that modifies the fundamental frequency trends by deriving (α, β, γ) from sound features, using the decomposition

$$F_{0,\text{out}}(m) = \gamma\,\overline{F_{0,\text{in}}} + \alpha(\overline{F_{0,\text{in}}^{loc}} - \overline{F_{0,\text{in}}}) + \beta\,\Delta F_{0,\text{in}}(m), \tag{9.77}$$

where $\overline{F_{0,\text{in}}}$ is the mean of $F_{0,\text{in}}$ over the whole signal [AV03]. One can independently control the mean fundamental frequency (γ e.g., controlled by the first formant frequency), the macro-intonation structure (α e.g., controlled by the second formant frequency) and the micro-intonation structure (β e.g., controlled by the intensity level $a(m)$); as well as strengthen ($\alpha \geq 1$ and $\beta \geq 1$), flatten ($0 \leq \alpha < 1$ and $0 \leq \beta < 1$), or inverse ($\alpha < 0$ and $\beta < 0$) an intonation, thus modifying the voice ambitus, but also the whole expressiveness of speech, similarly to people from different countries speaking a foreign language with different accents.

9.4.4 Adaptive effects on timbre

The perceptual attribute of timbre offers the widest category of audio effects, among which voice morphing [DGR95, CLB+00] and voice conversion, spectral compressor (also known as Contrast [Fav01]), automatic vibrato (with both frequency and amplitude modulation [ABLS02]), martianization [VA02], adaptive spectral tremolo, adaptive equalizer, and adaptive spectral warping [VZA06].

[5] In order to avoid rapid pitch-shifting modifications at the boundaries of voiced segments, the local mean of unvoiced segments is modified as the linear interpolation between its bound values (see Figure 9.35.(b)). The same modification is applied to the remainder (micro-intonation).

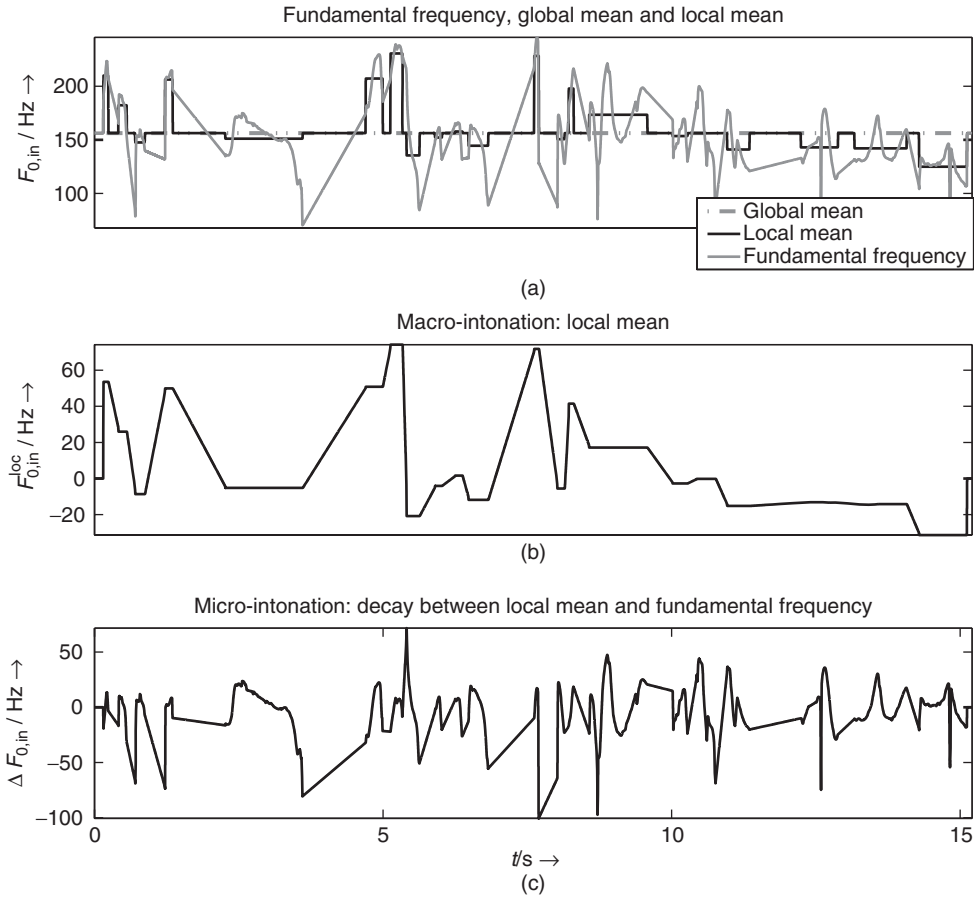

Figure 9.35 Intonation decomposition using an improved voiced/unvoiced mask. (a) Fundamental frequency $F_{0,\text{in}}(m)$, global mean $\overline{F_{0,\text{in}}}$ and local mean $\overline{F^{loc}_{0,\text{in}}}$. (b) Macro-intonation $\overline{F^{loc}_{0,\text{in}}}$ with linear interpolation between voiced segments. (c) Micro-intonation $\Delta F_{0,\text{in}}(m)$ with the same linear interpolation.

Adaptive equalizer

This effect is obtained by applying a time-varying equalizing curve $H_i(m, k)$, which is constituted of N_q filter gains of a constant-Q filter bank. In the frequency domain, we extract a feature vector of length N denoted[6] $f_i(m, \bullet)$ from the STFT $X_i(m, k)$ of each input channel i (the sound being mono or multi-channel). This vector feature $f_i(m, \bullet)$ is then mapped to $H_i(m, \bullet)$, for example by averaging its values in each of the constant-Q segments, or by taking only the N_q first values of $f_i(m, \bullet)$ as the gains of the filters. The equalizer output STFT is then

$$Y_i(m, k) = H_i(m, k) \cdot X_i(m, k). \tag{9.78}$$

If $H_i(m, k)$ varies too rapidly, the perceived effect is not varying equalizer/filtering, but ring modulation of partials, and potentially phasing. To avoid this, we low pass filter $H_i(\bullet, k)$ in time

[6] The notation $f_i(m, \bullet)$ corresponds to the frequency vector made of $f_i(m, k), k = 1, ..., N$.

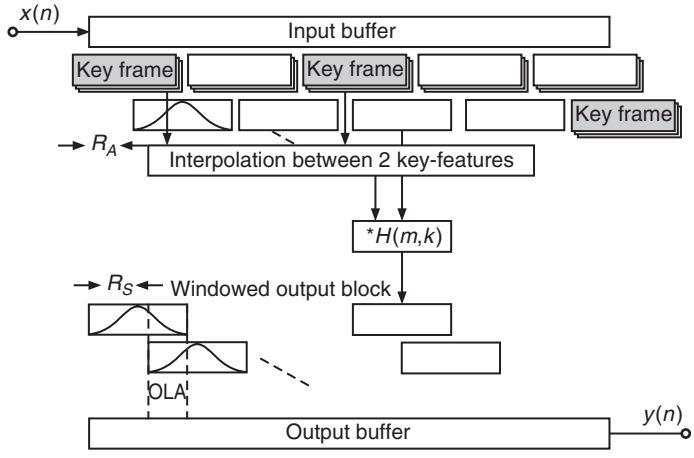

Figure 9.36 Block-by-block processing of adaptive equalizer. The equalizer curve is derived from a vector feature that is low pass filtered in time, using interpolation between key frames. Figure reprinted with IEEE permission from [VZA06].

[VD04], with L the down sampling ratio, $F_I = F_B/L$ the equalizer control sampling rate, and F_B the block sampling rate. This is obtained by linear interpolation between two key vectors denoted C_{P-1} and C_P (see Figure 9.36). For each block position m, $PL \leq m \leq (P+1)L$, the vector feature $f_i(m, \bullet)$ is given by

$$f_i(m,k) = \alpha(m) \cdot C_{P-1}(m,k) + (1-\alpha(m)) \cdot C_P(m,k), \tag{9.79}$$

with $\alpha(m) = (m - PL)/L$ the interpolation ratio. Real-time implementation requires the extraction of a fast-computing key vector $C_P(m,k)$, such as the samples buffer $C_P(m,k) = x\,(PLR_A + k)$, or the spectral envelope $C_P(m,k) = E(PL,k)$. However, non-real-time implementations allow for using more computationally expensive features, such as a harmonic comb filter, thus providing an odd/even harmonics balance modification.

Adaptive spectral warping

Harmonicity is adaptively modified when using spectral warping with an adaptive warping function $W(m,k)$. The STFT magnitude is computed by

$$|Y(m,k)| = |X\,(m, W(m,k))\,| \tag{9.80}$$

The warping function is given by

$$W\,(m,k) = C_1(m) \cdot C_2(m,k) + (1 - C_1(m)) \cdot k \tag{9.81}$$

and varies in time according to two control parameters: a vector $C_2(m,k) \in [1, N]$, $k = 1, ..., N$ (e.g., the spectral envelope E or its cumulative sum) which is the maximum warping function, and an interpolation ratio $C_1(m) \in [0, 1]$ (e.g., the energy, the voiciness), which determines the warping depth. An example is shown in Figure 9.37, with $C_2(m,k)$ derived from the spectral

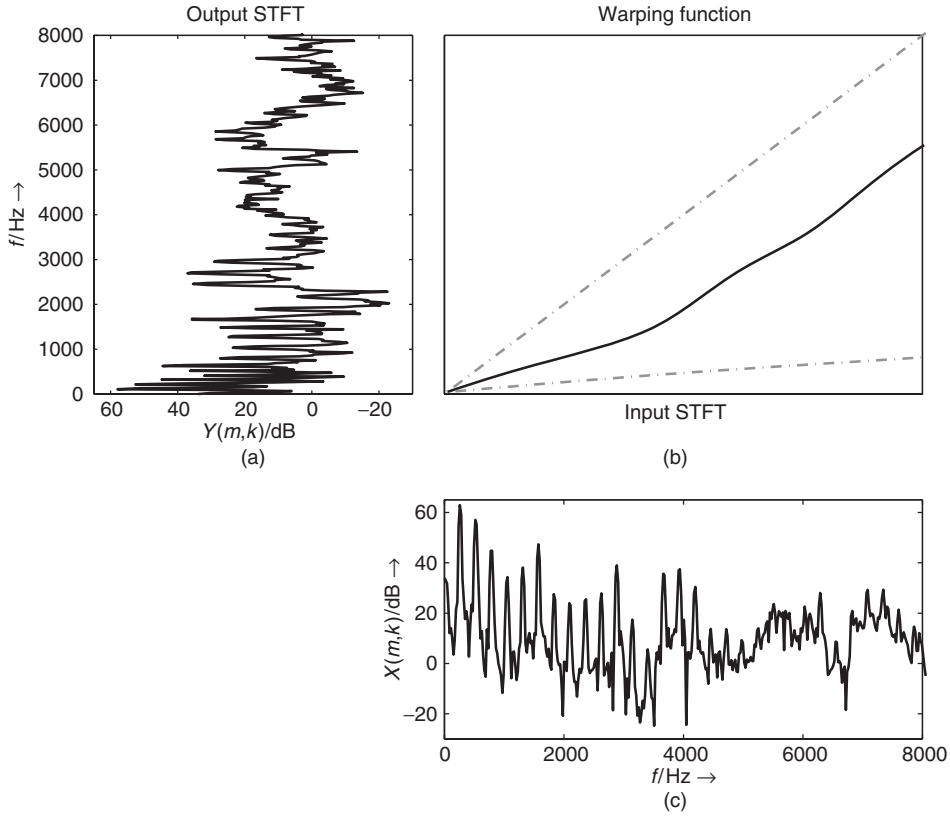

Figure 9.37 A-spectral warping: (a) output STFT, (b) warping function derived from the cumulative sum of the spectral envelope, (c) Input STFT. The warping function gives to any frequency bin the corresponding output magnitude. The spectrum is then non-linearly scaled according to the warping-function slope p: compressed for $p < 1$ and expanded for $p > 1$. The dashed lines represent $W(m, k) = C_2(m, k)$ and $W(m, k) = k$. Figure reprinted with IEEE permission from [VZA06].

envelope $E(m, k)$ as

$$C_2(m, k) = k - (N - 1) \cdot \frac{\sum_{l=2}^{k} E(m, l)}{\sum_{l=2}^{N} E(m, l)}. \tag{9.82}$$

This mapping provides a monotonous curve, and prevents the spectrum folding over. Adaptive spectral warping allows for dynamically changing the harmonicity of a sound. When applied only to the source, it allows for better in-harmonizing of voice or a musical instrument since formants are preserved.

9.4.5 Adaptive effects on spatial perception

Adaptive panning

Usually, panning requires the use of both a modification of left and right intensity levels and delays. In order to avoid the Doppler effect, delays are not taken into account in this example. With adaptive

control, the azimuth angle $\theta(n) \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ varies in time according to sound features. A constant power panning with the Blumlein law [Bla83] gives the following gains

$$L_l(n) = \frac{\sqrt{2}}{2}(\cos\theta(n) + \sin\theta(n)) \tag{9.83}$$

$$L_r(n) = \frac{\sqrt{2}}{2}(\cos\theta(n) - \sin\theta(n)). \tag{9.84}$$

A sinusoidal control $\theta(n) = \sin(2\pi f_{pan} n/F_A)$ with $f_{pan} > 20$ Hz is not heard anymore as a motion, but as a ring modulation (with a phase decay of $\pi/2$ between the two channels). With more complex motions obtained from sound feature control, this effect does not appear because the motion is not sinusoidal and varies most of the time under 20 Hz. The fast motions cause a stream segregation effect [Bre90], and the coherence in time between the sound motion and the sound content gives the illusion of splitting a monophonic sound into several sources. An example consists of panning synthetic trumpet sounds (obtained by frequency-modulation techniques [Cho71]) with an adaptive control derived from brightness, which is a strong perceptual indicator of brass timbre [Ris65], given by

$$\theta(n) = \pi \cdot \frac{cgs(n)}{f_S} - \frac{\pi}{4}. \tag{9.85}$$

Low-brightness sounds are left panned, whereas high brightness sounds are right panned. Brightness of trumpet sounds evolves differently during notes, attack and decay, implying that the sound attack moves fast from left to right, whereas the sound decay moves slowly from right to left. This adaptive control then provides a spatial spreading effect.

Adaptive spectral panning

Panning in the spectral domain allows for intensity panning by modifying the left and right spectrum magnitudes as well as for time delays by modifying the left and right spectrum phases. Using the phase vocoder, we only use intensity panning in order to avoid the Doppler effect, thus requiring slow motions of sound. To each frequency bin of the input STFT $X(m, k)$ we attribute a position given by the panning angle $\theta(m, k)$ derived from sound features. The resulting gains for left and right channels are then

$$L_l(m, k) = \frac{\sqrt{2}}{2}(\cos\theta(m, k) + \sin\theta(m, k)) \tag{9.86}$$

$$L_r(m, k) = \frac{\sqrt{2}}{2}(\cos\theta(m, k) - \sin\theta(m, k)). \tag{9.87}$$

In this way, each frequency bin of the input STFT is panned separately from its neighbors (see Figure 9.38): the original spectrum is then split across the space between two loudspeakers. To avoid a phasiness effect due to the lack of continuity of the control curve between neighboring frequency bins, a smooth control curve is needed, such as the spectral envelope. In order to control the variation speed of the spectral panning, $\theta(m, k)$ is computed from a time-interpolated value of a control vector (see the adaptive equalizer, Section 9.4.4). This effect adds envelopment to the sound when the panning curve is smoothed. Otherwise, the signal is split into virtual sources having more or less independent motions and speeds. In the case the panning vector $\theta(m, \bullet)$ is derived from the magnitude spectrum with a multi-pitch tracking technique, it allows for source separation. When derived from the voiciness $v(m)$ as $\theta(m, k) = \frac{\pi v(m) \cdot (2k - N - 1)}{4(N-1)}$, the sound localization varies between a point during attacks and a wide spatial spread during steady state, simulating width variations of the sound source.

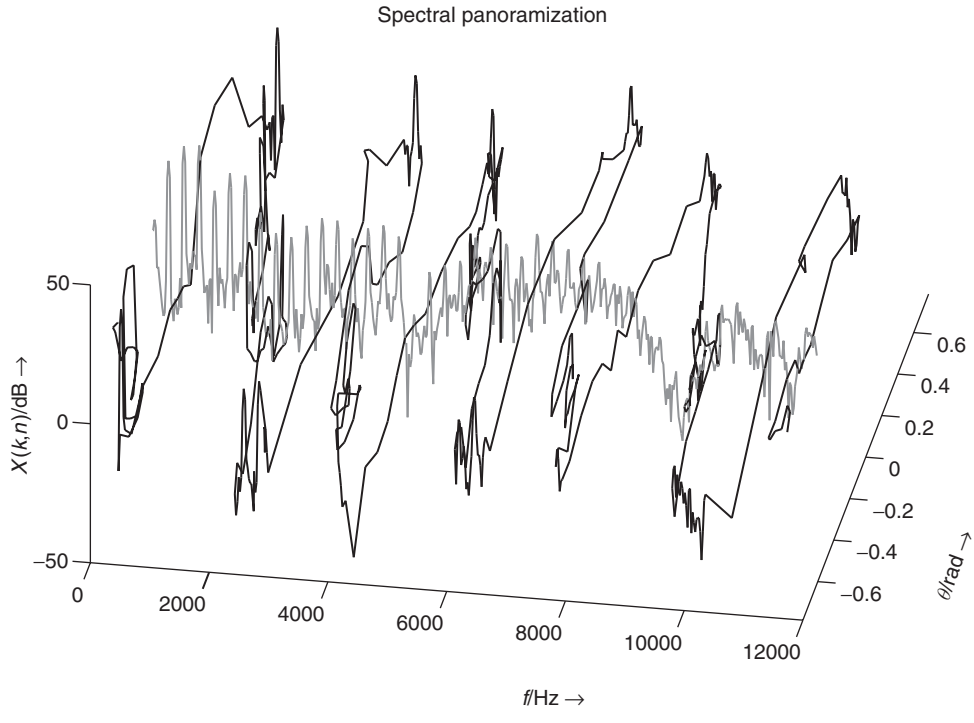

Figure 9.38 Frequency-space domain for the adaptive spectral panning (in black). Each frequency bin of the original STFT $X(m,k)$ (centered with $\theta = 0$, in gray) is panned with constant power. The azimuth angles are derived from sound features as $\theta(m,k) = x(mR_A - N/2 + k) \cdot \pi/4$. Figure reprinted with IEEE permission from [VZA06].

Spatialization

Both adaptive panning and adaptive spectral panning can be extended to 3-D sound projection, using techniques such as VBAP techniques, and so on. For instance, the user defines a trajectory (for example an ellipse), onto which the sound moves, with adaptive control on the position, the speed, or the acceleration. Concerning the position control, the azimuth can depend on the chroma $a(n) = \frac{\pi \log_2 F_0(n)}{6}$, splitting the sounds into a spatial chromatic scale. The speed control adaptively depending on voiciness as $\dot{x}(n) = (1 - v(n))$ allows for the sound to move only during attacks and silences; conversely, an adaptive control of speed given as $\dot{x}(n) = v(n)$ allows for the sound to move only during steady states, and not during attacks and silences.

9.4.6 Multi-dimensional adaptive effects

Various adaptive effects affect several perceptual attributes simultaneously: adaptive resampling modifies time, pitch and timbre; adaptive ring modulation modifies only harmonicity when combined to formant preservation, and harmonicity and timbre when combined with formants modifications [VD04]; gender change combines pitch-shifting and adaptive formant-shifting [ABLS01, ABLS02] to transform a female voice into a male voice, and vice versa. We now present two other multi-dimensional adaptive effects: adaptive robotization, which modifies pitch and timbre, and adaptive granular delay, which modifies spatial perception and timbre.

Adaptive robotization

Adaptive robotization changes expressiveness on two perceptual attributes, namely intonation (pitch) and roughness (timbre), and allows the transformation a human voice into an expressive robot voice [VA01]. This consists of zeroing the phase $\varphi(m,k)$ of the grain STFT $X(m,k)$ at a time index given by sound features: $Y(m,k) = |X(m,k)|$, and zeroing the signal between two blocks [VA01, AKZ02b]. The synthesis time index $t_S(m) = t_A(m)$ is recursively given as

$$t_S(m) = t_S(m-1) + R_S(m). \tag{9.88}$$

The step increment $R_S(m) = \frac{F_A}{F_0(m)}$ is also the period of the robot voice, i.e., the inverse of the robot fundamental frequency to which sound features are mapped (e.g., the spectral centroid as $F_0(m) = 0.01 \cdot cgs(m)$, in Figure 9.39). Its real-time implementation implies the careful use of a circular buffer, in order to allow for varying window and step increments. Both the harmonic and the noisy part of the sound are processed, and formants are locally preserved for each block. However, the energy of the signal is not preserved, due to the zero-phasing, the varying step increment, and the zeroing process between two blocks, resulting in a pitch and modifying the

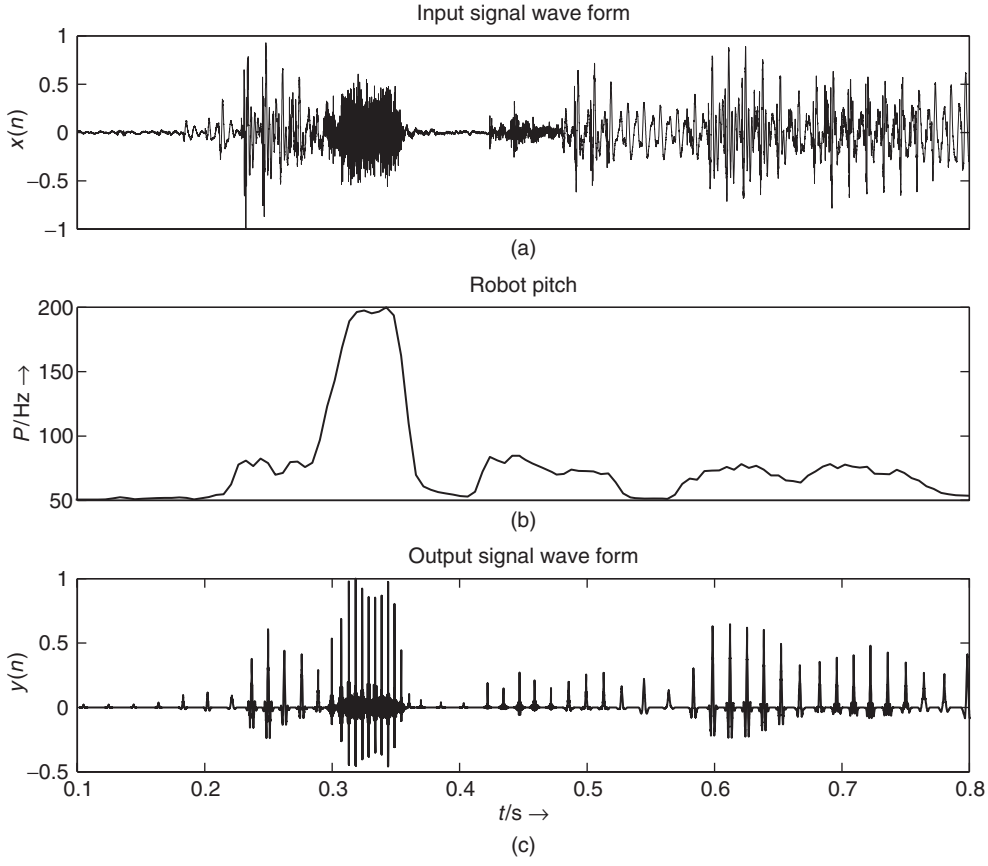

Figure 9.39 Robotization with a 512 samples block. (a) Input signal wave form. (b) $F_0 \in [50, 200]$ Hz derived from the spectral centroid as $F_0(m) = 0.01 \cdot cgs(m)$. (c) A-robotized signal wave form before amplitude correction. Figure reprinted with IEEE permission from [VZA06].

loudness of noisy content. An annoying buzz sound is then perceived, and can be easily removed by reducing the loudness modification. After zeroing the phases, the synthesis grain is multiplied by the ratio of analysis to synthesis intensity level computed on the current block m given by

$$y_{norm}(n) = y(n) \cdot \frac{a_x(m)}{a_y(m)}, n \in [m - N/2, m + N/2]. \tag{9.89}$$

A second adaptive control is given by the block size $N(m)$ and allows for changing the robot roughness: the lower the block length, the higher the roughness. At the same time, it allows preservation of the original pitch (e.g., $N \geq 1024$) or removal (e.g., $N \leq 256$), with an ambiguity in between. This is due to the fact that zero phasing a small block creates a main peak in the middle of the block and implies amplitude modulation (and then roughness). Inversely, zero phasing a large block creates several additional peaks in the window, the periodicity of the equally spaced secondary peaks being responsible for the original pitch.

Adaptive granular delay

This effect consists of applying delays to sound grains, with constant grain size N and step increment R_A [VA01], and varying delay gain $g(m)$ and/or delay time $\tau(m)$ derived from sound features (see Figure 9.40). In non-real-time applications, any delay time is possible, even fractional delay times [LVKL96], since each grain repetition is overlapped and added into a buffer. However, real-time implementations require limiting the number of delay lines, and so forth, to quantize delay time and delay gain control curves to a limited number of values. In our experience, 10 values for the delay gain and 30 for the delay time is a good minimum configuration, yielding 300 delay lines.

In the case where only $g(m)$ varies, the effect is a combination between delay and timbre morphing (spatial perception and timbre). For example, when applying this effect to a plucked string sound and controlling the gain with a voiciness feature as $g(m) = 0.5 \cdot (1 + \cos(-\pi v(m)))$, the attacks are repeated a much longer time than the sustain part. With the complementary mapping $g(m) = 0.5 \cdot (1 + \cos(\pi(1 - v(m))))$, the attacks rapidly disappear from the delayed version, whereas the sustain part is still repeated.

In the case where only $\tau(m)$ varies, the effect is a kind of granular synthesis with adaptive control, where grains collapse in time, thus implying modifications of time, timbre, and loudness. With a delay time derived from voiciness $\tau(m) = v(m)$ (in seconds), attack and sustain parts of a plucked string sound have different delay times, so sustain parts may be repeated before the attack with repetitions going on, as depicted Figure 9.40: not only are time and timbre are modified, but also loudness, since the grain superposition is uneven.

Adaptive granular delay is a perfect example of how the creative modification of an effect with adaptive control offers new sound transformation possibilities. It also shows how the frontiers between the perceptual attributes modified by the effect may be blurred.

9.4.7 Concatenative synthesis

The concept of concatenative synthesis is already in use for speech synthesis. An overview of this technique for audio processing can be found in [Sch07]. From an audio effects point of view it is an adaptive audio effect, which is controlled by low-level and high-level features extracted from the incoming audio signal and then uses sound snippets from a database to resynthesize a similar input audio. Figure 9.41 shows the functional units of such a signal-processing scheme. Sound features are derived from a block-by-block analysis scheme, which are then used for the selection of sound units from a sound database which possess similar sound features. These selected sound units are then used in a concatenative synthesis to reconstruct an audio signal. The concatenative synthesis can be performed by all time- and frequency-domain techniques discussed inside the DAFx book.

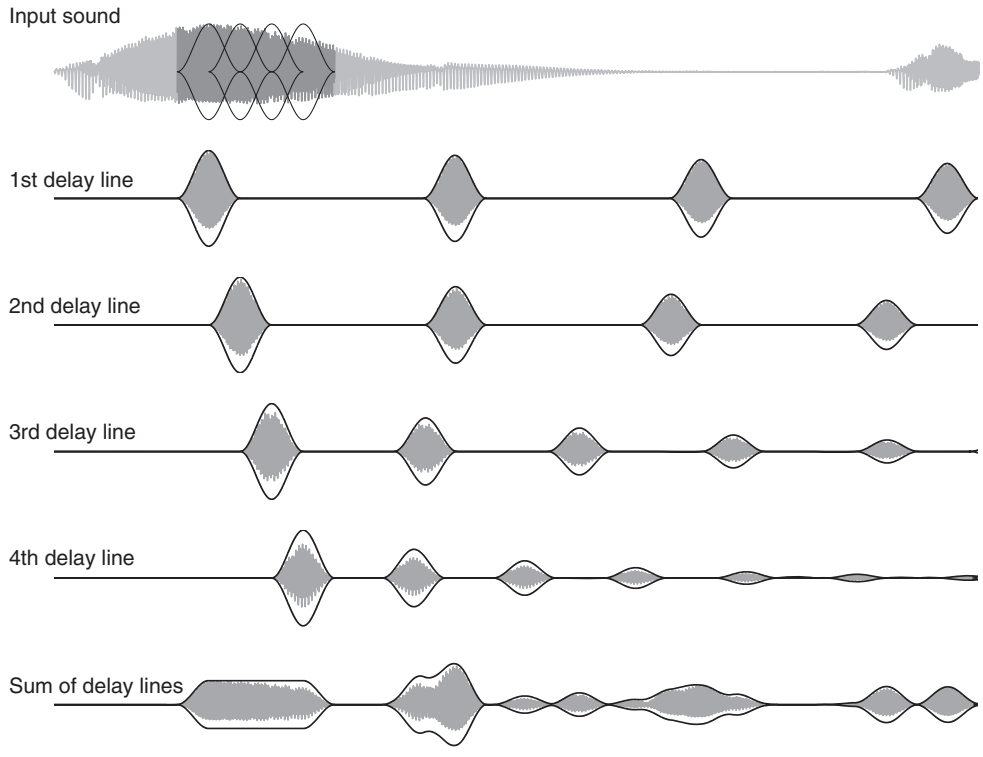

Figure 9.40 Illustration of the adaptive granular delay: each grain is delayed, with feedback gain $g(m) = a(m)$ and delay time $\tau(m) = 0.1 \cdot a(m)$ both derived from intensity level. Since the intensity level of the four first grains is going down, the gains $(g(m))^n$ and delay times $n\tau(m)$ of the repetitions are also going down with n, resulting in a granular time-collapsing effect. Figure reprinted with IEEE permission from [VZA06].

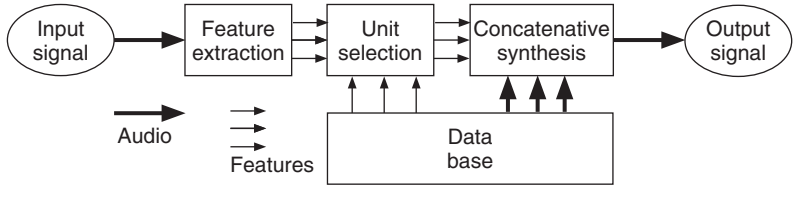

Figure 9.41 Concatenative synthesis using feature extraction, unit selection and concatenation based on a sound database.

The following M-file 9.15 shows the main processing steps for a concatenative singing resynthesis used in [FF10]. The script only presents the minimum number of features for singing resynthesis. It extracts energy, pitch, and LPC frequency response (no pitch smoothing or complex phonetic information) and searches for the best audio frame with a similar pitch. It uses the chosen frame, changes its gain to get a similar energy, and adds it to the output (no phase alignment or pitch changing). Further improvements and refinements are introduced in [FF10].

M-file 9.15 (SampleBasedResynthesis.m)

```
function out=SampleBasedResynthesis(audio)
% out = SampleBasedResynthesis(audio)
% Author: Nuno Fonseca (nuno.fonseca@ipleiria.pt)
%
% Resynthesizes the input audio, but using frames from an internal
% sound library.
%
% input:      audio signal
%
% output:     resynthesized audio signal

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Variables
    sr=44100;           % sample rate
    wsize=1024;         % window size
    hop=512;            % hop size
    window=hann(wsize); % main window function
    nFrames=GetNumberOfFrames(audio,wsize,hop); % number of frames

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Feature extraction (Energy, Pitch, Phonetic)
    disp('Feature extraction');
    % Get Features from input audio
    % Get Energy
    disp('- Energy (input)');
    energy=zeros(1,nFrames);
    for frame=1:nFrames
        energy(frame)=sumsqr(GetFrame(audio,frame,wsize,hop).*window);
    end
    % Get Pitch
    disp('- Pitch (input)');
    pitch=yinDAFX(audio,sr,160,hop);
    nFrames=min(nFrames,length(pitch));
    % Get LPC Freq Response
    disp('- Phonetic info (input)');
    LpcFreqRes=zeros(128,nFrames);
    window2=hann(wsize/4);
    audio2=resample(audio,sr/4,sr);
    for frame=1:nFrames
        temp=lpc(GetFrame(audio2,frame,wsize/4,hop/4).*window2,12);
        temp(find(isnan(temp)))=0;
        LpcFreqRes(:,frame)=20*log10(eps+abs(freqz(1,temp,128)));
    end

    % Load lib and extract information
    disp('- Loading sound library');
    lib=wavread('InternalLib.wav');
    disp('- Pitch (lib)');
    pitchLib=yinDAFX(lib,sr,160,hop);
    notesLib=round(note(pitchLib));
    disp('- Phonetic info (lib)');
    libLpcFreqRes=zeros(128,GetNumberOfFrames(lib,wsize,hop));
    audio2=resample(lib,sr/4,sr);
    for frame=1:GetNumberOfFrames(lib,wsize,hop)
        temp=lpc(GetFrame(audio2,frame,wsize/4,hop/4).*window2,12);
        temp(find(isnan(temp)))=0;
```

```
        libLpcFreqRes(:,frame)=20*log10(eps+abs(freqz(1,temp,128)));
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Unit Selection
    disp('Unit Selection');
    chosenFrames=zeros(1,nFrames);
    for frame=1:nFrames
        % From all frames with the same musical note
        % the system chooses the more similar one
        temp=LpcFreqRes(:,frame);
        n=round(note(pitch(frame)));
        indexes=find(notesLib==n);
        if(length(indexes)==0)
            n=round(note(0));
           indexes=find(notesLib==n);
        end
        [distance,index]=min(dist(temp',libLpcFreqRes(:,indexes)));
        chosenFrames(frame)=indexes(index);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Synthesis
    disp('Synthesis');
    out=zeros(length(audio),1);
    for frame=1:nFrames
        % Gets the frame from the sound lib., change its gain to have a
        % similar energy, and adds it to the output buffer.
        buffer=lib((chosenFrames(frame)-1)*hop+1:(chosenFrames(frame)-1)...
            *hop+wsize).*window;
        gain=sqrt(energy(frame)/sumsqr(buffer));
        out((frame-1)*hop+1:(frame-1)*hop+wsize)=out((frame-1)...
            *hop+1:(frame-1)*hop+wsize)+ buffer*gain;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Auxiliary functions

% Get number of frames
function out=GetNumberOfFrames(data,size,hop)
    out=max(0,1+floor((length(data)-size)/hop));
end

% Get Frame, starting at frame 1
function out=GetFrame(data,index,size,hop)
    if(index<=GetNumberOfFrames(data,size,hop))
        out=data((index-1)*hop+1:(index-1)*hop+size);
    else
        out=[];
    end
end

% Convert frequency to note (MIDI value)
function out=note(freq)
    out=12*log(abs(freq)/440)/log(2)+69;
end
```

9.5 Conclusions

From existing simple effects such as the compressor and auto-tune, we presented a general framework for adaptive DAFX using sound features as control parameters. Such adaptive control can rely on any type of sound features that are extracted from various representations of sound: samples, STFT, spectrum, source and filter. Indeed, previously described techniques such as the phase vocoder, source-filter models, and spectral models do allow for both the representation of the sound and for the extraction of further global features such as pitch or fundamental frequency, which can be estimated by the cepstrum method or auto-correlation techniques applied to the input directly or the extracted source signal. Further global features such as amplitude envelope, spectral centroid, and auto-correlation features (voiced/unvoiced detection) have been introduced, which can be estimated by simple time-domain or by advanced time-frequency techniques.

References

[ABL+03] X. Amatriain, J. Bonada, A. Loscos, J. L. Arcos, and V. Verfaille. Content-based Transformations. *J. New Music Res*., 32(1): 95–114, 2003.

[ABLS01] X. Amatriain, J. Bonada, A. Loscos, and X. Serra. Spectral modeling for higher-level sound transformations. In *MOSART Workshop Curr. Res. Dir. Comp. Music, IUA-UPF,* 2001.

[ABLS02] X. Amatriain, J. Bonada, A. Loscos, and X. Serra. Spectral processing. In U. Zölzer (ed.), *DAFX - Digital Audio Effects*, pp. 373–438. J. Wiley & Sons, Ltd, 2002.

[ACKV02] D. Arfib, J.-M. Couturier, L. Kessous, and V. Verfaille. Strategies of mapping between gesture parameters and synthesis model parameters using perceptual spaces. *Org. Sound*, 7(2): 135–52, 2002.

[AD98] D. Arfib and N. Delprat. Selective transformations of sound using time-frequency representations: An application to the vibrato modification. In *104th Conv. Audio Eng. Soc*., 1998.

[AKZ99] R. Althoff, F. Keiler, and U. Zölzer. Extracting sinusoids from harmonic signals. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 97–100, 1999.

[AKZ02a] D. Arfib, F. Keiler, and U. Zölzer. Source-filter processing. In U. Zölzer (ed.), *DAFX - Digital Audio Effects*, pp. 299–372. J. Wiley & Sons, Ltd, 2002.

[AKZ02b] D. Arfib, F. Keiler, and U. Zölzer. Time-frequency processing. In U. Zölzer (ed.), *DAFX - Digital Audio Effects*, pp. 237–97. J. Wiley & Sons, Ltd, 2002.

[Arf79] D. Arfib. Digital synthesis of complex spectra by means of multiplication of non linear distorted sine waves. *J. Audio Eng. Soc*., 27: 757–768, 1979.

[Arf98] D. Arfib. Des Courbes et des Sons. In *Recherches et Applications en Informatique Musicale*, pp. 277–86. Hermès, 1998.

[AV03] D. Arfib and V. Verfaille. Driving pitch-shifting and time-scaling algorithms with adaptive and gestural techniques. In *Proc. Int. Conf. on Digital Audio Effects (DAFx-03)*, pp. 106–11, 2003.

[Bea82] J. W. Beauchamp. Synthesis by spectral amplitude and "brightness" matching of analyzed musical instrument tones. *J. Audio Eng. Soc*., 30(6): 396–406, 1982.

[BFC05] D. Barry, D. FitzGerald, and E. Coyle. Drum source separation using percussive feature detection and spectral modulation. In *Proc. IEE Irish Signals Syst. Conf*., pp. 217–220, 2005.

[Bla83] J. Blauert. *Spatial Hearing: the Psychophysics of Human Sound Localization*. MIT Press, 1983.

[Bon00] J. Bonada. Automatic technique in frequency domain for near-lossless time-scale modification of audio. In *Proc. Int. Comp. Music Conf. (ICMC'00)*, 2000.

[BP89] J. C. Brown and M S. Puckette. Calculation of a narrowed autocorrelation function. *J. Ac. Soc. Am*., 85: 1595–601, 1989.

[Bre90] A. Bregman. *Auditory Scene Analysis*. MIT Press, 1990.

[Bru79] M. Le Brun. Digital waveshaping synthesis. *J. Audio Eng. Soc*., 27(4): 250–265, April 1979.

[Cab99] D. Cabrera. PsySound: a computer program for psychoacoustical analysis. In *Proc. Australian Ac. Soc. Conf*., pp. 47–53, 1999.

[Cab00] D. Cabrera. PsySound 2: Psychoacoustical software for Macintosh PPC. Technical report, 2000.

[Cad99] C. Cadoz. Musique, geste, technologie. In H. Genevois and R. de Vivo, (eds), *Les Nouveaux Gestes de la Musique*, pp. 47–92. Parenthèses, 1999.

[Can98] P. Cano. Fundamental frequency estimation in the SMS analysis. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 99–102, 1998.

[Cho71] J. Chowning. The synthesis of complex audio spectra by means of frequency modulation. *J. Audio Eng. Soc*., 21: 526–34, 1971.

[CLB+00] P. Cano, A. Loscos, J. Bonada, M. de Boer, and X. Serra. Voice morphing system for impersonating in karaoke applications. In *Proc. Int. Comp. Music Conf. (ICMC'00)*, pp. 109–12, 2000.

[Cri82] A. Di Cristo. *Prolégomènes à L'étude de L'intonation*. Editions du CNRS, 1982.

[dCK02] A. de Cheveigné and H. Kawahara. Yin, a fundamental frequency estimator for speech and music. *J. Acoust. Soc. Am*., 111(4): 1917–1930, 2002.

[DGR95] P. Depalle, G. Garcia, and X. Rodet. Reconstruction of a castrato voice: Farinelli's voice. In *Proc. IEEE Workshop Appl. Digital Signal Proces. Audio Acoust*., 1995.

[DH00] M. Desainte-Catherine and P. Hanna. Statistical approach for sounds modeling. In *Proc. DAFX-00 Conf. Digital Audio Effects*, pp. 91–96, 2000.

[DM00] M. Desainte-Catherine and S. Marchand. High-precision Fourier analysis of sounds using signal derivatives. *J. Audio Eng. Soc*., 48(7/8): 654–667, 2000.

[DT96a] S. Dubnov and N. Tishby. Testing for gaussianity and non linearity in the sustained portion of musical sounds. In *Proc. Journées Informatique Musicale (JIM'96)*, 1996.

[DT96b] S. Dubnov and N. Tishby. Testing for gaussianity and non-linearity in the sustained portion of musical sounds. In *Proc. Journées Informatique Musicale*, 1996.

[DZ02] P. Dutilleux and U. Zölzer. Nonlinear processing. In U. Zölzer (ed), *DAFX - Digital Audio Effects*, pp. 93–135. J. Wiley & Sons, Ltd, 2002.

[Fav01] E. Favreau. Phase vocoder applications in GRM tools environment. In *Proc. COST-G6 Workshop on Digital Audio Effects (DAFx-01)*, pp. 134–7, 2001.

[FF10] N. Fonseca and A. Ferreira. Singing voice resynthesis using vocal sound libraries. In *Proc. of the 13th Int. Conf. on Digital Audio Effects (DAFx-10)*, pp. 322–325, 2010.

[Fit04] D. FitzGerald. *Automatic drum transcription and source separation*. PhD thesis, Dublin Institute of Technology, 2004.

[Fit10] D. FitzGerald. Harmonic/percussive separation using median filtering. In *Proc. of the 13th Int. Conf. on Digital Audio Effects (DAFx-10)*, pp. 217–220, Graz, Austria, 2010.

[GFB03] F. Gouyon, L. Fabig, and J. Bonada. Rhythmic expressiveness transformations of audio recordings: swing modifications. In *Proc. Int. Conf. on Digital Audio Effects (DAFx-03)*, pp. 94–99, 2003.

[GPAH03] E. Gómez, G. Peterschmitt, X. Amatriain, and P. Herrera. Content-based melodic transformations of audio material for a music processing application. In *Proc. Int. Conf. on Digital Audio Effects (DAFx-03)*, 2003.

[HB98] P. Herrera and J. Bonada. Vibrato extraction and parameterization in the spectral modeling synthesis framework. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 107–110, 1998.

[Hes83] W. Hess. *Pitch Determination of Speech Signals*. Springer-Verlag, 1983.

[INA03] INA-GRM. GRM Tools, 2003.

[JN84] N. S. Jayant and P. Noll. *Digital Coding of Waveforms*. Prentice-Hall, 1984.

[KA90] P. Kroon and B. S. Atal. Pitch predictors with high temporal resolution. In *Proc. ICASSP*, pp. 661–664, 1990.

[KZ10] A. von dem Knesebeck and U. Zölzer. Comparison of pitch trackers for real-time guitar effects. In *Proc. 13th Int. Conf. on Digital Audio Effects (DAFx-10)*, pp. 266–269, 2010.

[LVKL96] T. I. Laakso, V. Välimäki, M. Karjalainen, and U. K. Laine. Splitting the unit delay. *IEEE Signal Proces. Mag*., 13: 30–60, 1996.

[Mar72] J. D. Markel. The SIFT algorithm for fundamental frequency estimation. *IEEE Trans. Audio Electroacoust*., 20(5): 367–377, 1972.

[Mar98] S. Marchand. Improving spectral analysis precision with enhanced phase vocoder using signal derivatives. In *Proc. DAFX-98 Digital Audio Effects Workshop*, pp. 114–118, 1998.

[Mar00] S. Marchand. *Sound models for computer music*. PhD thesis, University of Bordeaux, 2000.

[MB96] P. Masri and A. Bateman. Improved modelling of attack transients in music analysis-resynthesis. In *Proc. Int. Comp. Music Conf. (ICMC'96)*, pp. 100–3, 1996.

[MG96] B. C. J. Moore and B. R. Glasberg. A revision of Zwicker's loudness model. *Acta Acust*. United AC, 82: 3335–45, 1996.

[MGB97] B. C. J. Moore, B. R. Glasberg, and T. Baer. A model for the prediction of thresholds, loudness, and partial loudness. *J. Audio Eng. Soc*., 45(4): 224–40, 1997.

[Moo65] R. Moog. A voltage-controlled low-pass, high-pass filter for audio signal processing. In *17th Annual AES Meeting*, Preprint 413, 1965.

[MQ86] R. J. McAulay and T. F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Trans. Acoust. Speech, Signal Proces*., 34(4): 744–54, 1986.

[MR04] S. Marchand and M. Raspaud. Enhanced time-stretching using order-2 sinusoidal modeling. In *Proc. Int. Conf. Digital Audio Effects (DAFx-04)*, pp. 76–82, 2004.

[MWdSK95] S. McAdams, S. Winsberg, G. de Soete, and J. Krimphoff. Perceptual scaling of synthesized musical timbres: common dimensions, specificities, and latent subject classes. *Psychol. Res*., 58: 177–92, 1995.

[Nol64] A. M. Noll. Short-time spectrum and "cepstrum" techniques for vocal-pitch detection. *J. Acoust. Soc. Am*., 36(2): 296–302, 1964.

[NZ08] F. X. Nsabimana and U. Zölzer. Audio signal decomposition for pitch and time scaling. In *3rd Int. Symp. Commun. Control Signal Proces*., pp. 1285–1290, 2008.

[OS75] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Prentice-Hall, 1975.

[O′S00] D. O'Shaugnessy. *Speech Communication*, 2nd edition. Addison-Wesley, 2000.

[Pal03] G. Pallone. *Dilatation et transposition sous contraintes perceptives des signaux audio: application au transfert cinéma-vidéo*. PhD thesis, University of Aix-Marseille III, 2003.

[PJ82] H. Pollard and E. Jansson. A tristimulus method for the specification of musical timbre. *Acta Acust*. United AC, 51: 162–71, 1982.

[PMH00] G. Peeters, S. McAdams, and P. Herrera. Instrument sound description in the context of MPEG-7. In *Proc. Int. Comp. Music Conf. (ICMC'00)*, 2000.

[RDS+99] S. Rossignol, P. Depalle, J. Soumagne, X. Rodet, and J.-L. Colette. Vibrato: detection, estimation, extraction, modification. In *Proc. DAFX-99 Digital Audio Effects Workshop*, pp. 175–179, 1999.

[Ris65] J.-C. Risset. Computer study of trumpet tones. *J. Ac. Soc. Am*., 33: 912, 1965.

[Ros98] S. Rossignol. Feature extraction and temporal segmentation of acoustic signals. In *Proc. Int. Comp. Music Conf*., 1998.

[RS78] L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, 1978.

[RSC+74] M. Ross, H. Shaffer, A. Cohen, R. Freudberg, and H. Manley. Average magnitude difference function pitch extractor. *IEEE Trans. Acoust. Speech Signal Proces*., 22(5): 353–362, 1974.

[Sch66] P. Schaeffer. *Traité des Objets Musicaux*. Seuil, 1966.

[Sch68] M. R. Schroeder. Period histogram and product spectrum: New methods for fundamental-frequency measurement. *J. Acoust. Soc. Am*., 43(4): 829–834, 1968.

[Sch99] M. R. Schroeder. *Computer Speech*. Springer-Verlag, 1999.

[Sch07] D. Schwarz. Corpus-Based Concatenative Synthesis. *IEEE Signal Proces. Mag*., 24(2): 92–104, 2007.

[Son68] M. M. Sondhi. New methods of pitch extraction. *IEEE Trans. on Audio Electroacoust*., 16(2): 262–266, 1968.

[Sun87] J. Sundberg. *The Science of the Singing Voice*. Northern Illinois University Press, 1987.

[SZST10] A. Spich, M. Zanoni, A. Sarti, and S. Tubaro. Drum music transcription using prior subspace analysis and pattern recognition. In *Proc. 13th Int. Conf. Digital Audio Effects (DAFx-10)*, pp. 233–237, 2010.

[UH03] C. Uhle and J. Herre. Estimation of tempo, micro time and time signature from percussive music. In *Proc. 6th Int. Conf. Digital Audio Effects (DAFx-03)*, pp. 84–89, 2003.

[vA85] W. von Aures. Der sensorische wohlklang als funktion psychoakustischer empfindungsgröfsen. *Acustica*, 58: 282–90, 1985.

[VA01] V. Verfaille and D. Arfib. ADAFx: Adaptive digital audio effects. In *Proc. COST-G6 Workshop on Digital Audio Effects (DAFx-01)*, pp. 10–4, 2001.

[VA02] V. Verfaille and D. Arfib. Implementation strategies for adaptive digital audio effects. In *Proc. Int. Conf. Digital Audio Effects (DAFx-02)*, pp. 21–6, 2002.

[VD04] V. Verfaille and Ph. Depalle. Adaptive effects based on STFT, using a source-filter model. In *Proc. Int. Conf. Digital Audio Effects (DAFx-04)*, pp. 296–301, 2004.

[VM90] E. Vidal and A. Marzal. A review and new approaches for automatic segmentation of speech signals. In L. Torres, E. Masgrau, and M. A. Lagunas (eds), *Signal Processing V: Theories and Applications*, pp. 43–53. Elsevier Science Publishers, 1990.

[VWD06] V. Verfaille, M. M. Wanderley, and Ph. Depalle. Mapping Strategies for Gestural Control of Adaptive Digital Audio Effects. *J. New Music Res*., 35(1): 71–93, 2006.

[VZA06] V. Verfaille, U. Zölzer, and D. Arfib. Adaptive digital audio effects (A-DAFx): A new class of sound transformations. *IEEE Trans. Audio Speech Lang. Proc*., 14(5): 1817–1831, 2006.

[Wan02] M. M. Wanderley. Mapping strategies in real-time computer music. *Org. Sound*, 7(2), 2002.

[WD00] M. M. Wanderley and P. Depalle. Gesturally controlled digital audio effects. In *Proc. COST-G6 Workshop Digital Audio Effects (DAFx-00)*, pp. 165–9, 2000.

[ZF99] E. Zwicker and H. Fastl. *Psychoacoustics: Facts and Models*. Springer-Verlag, Berlin, 1999.

[ZS65] E. Zwicker and B. Scharf. A model of loudness summation. *Psychol. Rev*., 72: 3–26, 1965.

[Zwi77] E. Zwicker. Procedure for calculating loudness of temporally variable sounds. *J. Ac. Soc. Am*., 62(3): 675–82, 1977.