Institute of Broadband Communication (Univ.-Prof. Dipl.-Ing. Dr.techn. Gernot Kubin)
Faculty of Electrical and Information Engineering
Graz University of Technology

Theory, Implementation and Evaluation of the Digital Phase Vocoder in the Context of Audio Effects

Bachelor Thesis, Telematics

Author
Johannes Grünwald

Advisor
Dipl.-Ing. Dr.techn. Werner Magnes

Graz, July 2010.

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, July 8, 2010

Johannes Grünwald

ABSTRACT

The fast technological progress of recent decades has brought a wide range of new possibilities to the field of electronic music. Besides analog filters, the theory of digital signal processing motivated the development of special-purpose processors in order to execute such sample-based algorithms on time-discrete signals. One representative of a time-discrete signal processing concept is the *digital phase vocoder*, which permits to observe and manipulate digital signals in both time- *and* frequency domain simultaneously.

In this Bachelor Thesis, a comprehensive analysis of the digital phase vocoder in musical context was carried out. At first, the necessary theory was delineated in order to gain a basic understanding of underlying concepts. Secondly, some popular audio effects utilizing the digital phase vocoder were described, such as *time stretching* (modification of the temporal evolution of a signal but keeping its pitch the same) and *pitch shifting* (modification of the signal's pitch but preserving its temporal evolution). Several issues arise from the conventional implementation of these two effects, so they are discussed more in detail than the others.

An exemplary phase vocoder was realized using the mathematical development environment *MathWorks MATLAB*® which facilitated the implementation and evaluation perfectly well.

ZUSAMMENFASSUNG

Der technologische Fortschritt der letzten Jahrzehnte ermöglichte insbesondere im Bereich der elektronischen Musik die Erschließung grundlegender neuer Gestaltungsmöglichkeiten. Zusätzlich zu etablierten analogen Filterschaltungen konnte nun – motiviert durch die theoretischen Grundlagen der digitalen Signalverarbeitung – auf speziell angefertigen Prozessoren die zeitdiskrete, digitale Verarbeitung von Signalen realisiert werden. Ein populärer Repräsentant solcher digitaler Verarbeitungskonzepte ist der *Digital Phase Vocoder*. Dieser ermöglicht die simultane Betrachtung und Modifikation von digitalen Signalen im Zeit- *und* Frequenzbereich.

In dieser Bakkalaureatsarbeit wurde dieser Besonderheit des Phase Vocoders Rechnung getragen und eine eingehende Analyse in musikalischem Kontext durchgeführt. Zunächst erfolgte die Darlegung der theoretischen Grundlagen, um der Leserin und dem Leser ein Grundverständnis für die Arbeitsweise des Phase Vocoders zu vermitteln. Anschließend wurden einige ausgewählte Audioeffekte präsentiert. Dabei lag besonderer Fokus auf den größten Herausforderungen, dem *time stretching* (Veränderung der Dauer des Signals bei gleichbleibender Tonhöhe) sowie dem *pitch shifting* (Veränderung der Tonhöhe eines Signals bei gleichbleibender Dauer), inklusive einer Analyse der zahlreichen Probleme, die gängige Verfahren mit sich bringen.

Die Realisierung eines beispielhaften Phase Vocoders wurde in der mathematischen Entwicklungsumgebung *MathWorks MATLAB*® durchgeführt, was einer zielführenden Umsetzung zuträglich war und flexible, umfassende Evaluierungsmöglichkeiten offerierte.

Contents

1 Introduction

1.1 History of the Digital Phase Vocoder

When the phase vocoder[1] was first described in 1966 by Flanagan and Golden, it was intended to compress speech signals for communication purposes rather than to perform audio effects. Actually, the word *vocoder* is a contraction of the words *voice coder*. In the 20 years that followed, a huge amount of research was done on the phase vocoder in order to get a better understanding of the underlying technique and to facilitate the advantages it comes with[2]. It was Mark Dolson in 1986 who wrote a tutorial of the phase vocoder [Dol86] with its application in musical context in mind. From this time up to now, the phase vocoder has been developed and improved excessively for certain musical applications and is now widely used in the field of electronic music.

1.2 The Digital Phase Vocoder and Music

The phase vocoder utilizes the parallel modification of spectral and temporal components of a signal. Put in other words, this means that operations in the frequency domain can be carried out *online*, i.e. the input signal is processed as it arrives at the effect device. This way of real-time spectral modification is indeed a powerful tool and can, applied to audio signals, result in impressive audio effects. Several typical phase vocoder audio effects are elucidated in this thesis from a theoretical *and* practical point of view.

1.3 Motivation and Objectives for this Thesis

Audio effect devices are *hot spots* where music and art meets signal theory and mathematics. Being highly interested in both the artistical and technical approach, it was a logical consequence to choose a subject in this domain for my Bachelor Thesis.

The aim of this work is to give a comprehensive overview of the phase vocoder. This approach addresses theoretical and practical aspects, so *MathWorks MATLAB®* was chosen as an implementation and evaluation framework for the realization of an examplary phase vocoder.

Potential future work of this thesis might be the realtime implementation of the phase vocoder on a digital signal processor (DSP) or the investigation of new audio effects using the provided implementation as a basis.

1.4 Structure of this Thesis

This thesis comprizes two main parts that cover the *theoretical* and *practical* view on the phase vocoder, being divided into the *general theory* behind of the phase vocoder and the *theory of audio effects* as well as the *implementation* of the phase vocoder and the *evaluation* of its performance.

This thesis is mainly based on [Zoe02], if not otherwise stated. In such cases, the according literature is cited.

[1]In this Thesis, the terms *digital phase vocoder* and *phase vocoder* are used interchangeably.
[2]The interested reader may be referred to [BA70] [GR67] [CF87] [Bag78] [Loo97] [Gol80] [Fel82].

2 Theory of the Phase Vocoder

In this chapter, the phase vocoder is investigated from a theoretical point of view. In Section 2.1, a short overview is given, sketching the the basic idea of time-frequency processing and where the phase vocoder fits into this pattern. Additionally, the two main models, *the filter bank summation model* and the *block by block analysis / synthesis model* of the phase vocoder are delineated.

A formal mathematical description of the phase vocoder is deduced in Section 2.2, comprising the fundamental components – the *analysis stage*, *magnitude and phase processing stage* and *synthesis stage*.

2.1 Overview

2.1.1 Time-Frequency Processing

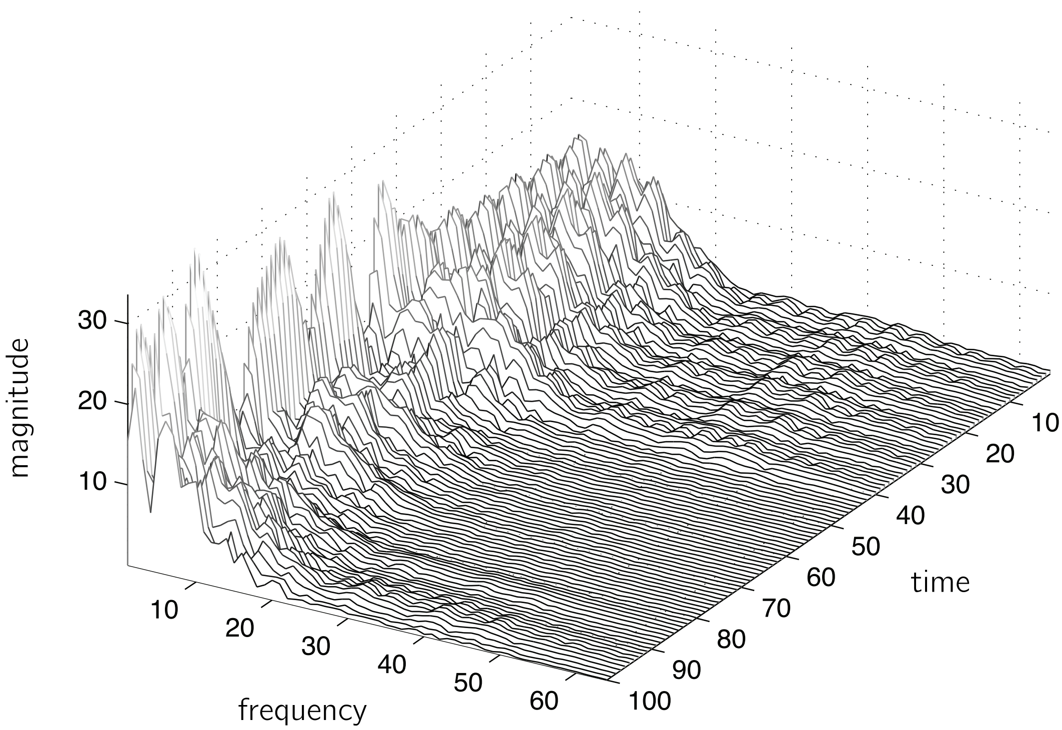


Figure 1: An exemplary waterfall plot of an acoustic signal in its time-frequency representation.

The basic idea of the phase vocoder is to edit a signal both in *time* and *frequency*. In order to achieve this, the signal is modeled as a sum of complex exponentials with time-varying amplitude and frequency. These attributes (commonly referred to as *magnitude* and *phase*) can then be processed over time in any desired way. This way of alternating the characteristics of a signal in time and frequency (modifications in both domains not necessarily being dependent on each other) is termed *time-frequency processing*, as visualized in Fig. 1.

2.1.2 Phase Vocoder Models

Without being familiar to further details, the phase vocoder and its mode of operation can be interpreted in two ways, the *filter bank summation model* and the *block-by-bock analysis/synthesis model* [Dol86]. These two approaches are shortly explained below.

Filter Bank Summation Model. One quite obvious possibility of modeling the phase vocoder is to employ a bank of identical bandpass filters which are centered around equally spaced frequencies. The output of each bandpass filter is then a magnitude and frequency representation of the expected complex exponential within that band. After manipulating these values as desired, one oscillator per band is driven with the results, contributing to the final time-domain signal, which is gained as a sum of all oscillator signals.

Summing up, there are three major constraints that have to be imposed on the filter bank summation model in order to accomplish decent results:

1. The frequency response characteristics of all bandpass filters must be identical except of their center frequencies.
2. These center frequencies must be equally spaced across the entire spectrum, ranging from 0 Hz to $f_s/2$ where f_s is the sampling frequency.
3. The combined frequency responses of all bandpass filters must be constant over the whole spectrum.

Block by Block Analysis / Synthesis Model. A similar approach to describing the phase vocoder is to represent the signal via succeeding *Discrete Fourier Transform* (DFT) frames of length N. These frames are first multiplied by an appropriate window (such as Hamming, Hanning, Kaiser, Blackman etc.) and then Fourier-transformed into the frequency domain. At this stage, any prudent modifcation of the spectrum can be made, before transforming it back to the time domain with the *Inverse Discrete Fourier Transform* (IDFT), where the delayed and optionally windowed parts are *overlap-add*ed together, yielding the final result.

Thus, for a given sample value n, each value (*frequency bin*) of the DFT-representative $X[k, n]$ corresponds to the output magnitude and phase of a bandpass filter with center frequency kf_s/N of the model above (f_s again denoting the sample frequency). The difference in this approach is though, that the filter bank summation model emphasizes the temporal evolution of the bandpass channels on their own, whereas the block by block analysis model rather concentrates on the whole spectrum at a given time. Nevertheless, both models are mathematically equivalent and the reason why both are pointed out in this thesis is that for different applications one model suits better to understand underlying ideas.

The reader may note that if the frequency of a complex exponential and the according bin of the DFT don't match exactly, the phase value will evolve over time, referring to the real frequency (also called *instantaneous frequency*) of the captured exponential. This fact must be taken into account if exact frequency estimation is a demand. One standard procedure that determines the instantaneous frequency is *phase unwrapping* which is elucidated in chapter 3.1.

Concerning the constraints that are enforced upon the filter bank summation model, it is clear that the block by block model complies with condition 1 since there are no real bandpass filters implemented. Furthermore, condition 2 is fulfilled by the inherent property of the DFT that it corresponds to the *Discrete-Time Fourier Transform* (DTFT), which is sampled at equidistant points [OS09a]. Requirement 3 is obsolete as well due to the fact that the DFT and IDFT are exactly inverse to each other, hence an identity operation in the frequency domain yields an output signal exactly identical to the input signal.

One issue arises from employing this model. As it is known, spectral components that don't exactly correspond to a frequency bin of the DFT, spread their energy across several adjoining bins. This effect is called *smearing* or *leakage* [Lyo96] and can be greatly reduced by windowing the input frames appropriately [OS09b].

Conclusion. The two models described above offer different possibilities how the phase vocoder can be interpreted. The latter of both provides a somewhat more practical point of view since the DFT and IDFT can be efficiently implemented by the *Fast Fourier Transform* (FFT) and *Inverse Fast Fourier Transform* (IFFT), respectively. In the context of digital signal processing this is very considerable since FFT algorithms reduce the complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. This is one of the reasons why a lot of research was done on implementing the phase vocoder via the FFT [Por76] [AR77].

2.2 Mathematical Description

In this subsection, the phase vocoder will be presented from its mathematical point of view. To relate theoretical and practical aspects as close as possible, the definition bases on a both a discrete time and discrete frequency domain[3]. This way differs from most approaches in the literature but seems suitable in this context.

The starting point of defining the phase vocoder are the DFT and IDFT, both defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\jmath 2\pi \frac{k}{N} n} \qquad \text{DFT Analysis Equation,} \tag{1}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \mathrm{e}^{\jmath 2\pi \frac{k}{N} n} \qquad \text{DFT Synthesis Equation.} \tag{2}$$

This transform is the main part of putting the input signal into the frequency domain and the modified spectrum back into the time domain again. The corresponding stages are called *analysis stage* and *synthesis stage*, surrounding the *processing stage* where the spectral alterations are carried out. These three main steps are described below [Cro80] [LD99a]. The reader may be referred to Fig. 2 on page 12 as an unliteral representation of the explanations below.

2.2.1 Analysis Stage

At the analysis stage, successive DFT frames are taken from the input signal $x[n]$ at the positions $n_a^u = uR_a$, where R_a is termed *input hop size* or *analysis hop size* and u being integer-valued. Given the length of the DFT as N, the input hop size is most likely (if not necessary at all) a submultiple of it. Common values are $N/2$, $N/4$ and $N/8$, depending on the analysis window and the required performance [DGBA00]. These values let the frames overlap by 50 %, 75 % and 87.5 %, respectively.

[3]Note: Arguments in this thesis are covered with braces if they are continuous (e.g. $f(t)$) and with brackets if they are discrete (e.g. $x[n]$).

Each of these input frames is then multiplied by an arbitrary analysis window $h_a[n]$. It is clear, that in order to achieve a perfect reconstruction of the signal, the analysis and synthesis windows must produce a constant sum over time if they are overlap-added by themselves. Furthermore, it is good practice to choose windows like Hamming, Hanning etc., since they provide the property that the sum of these windows, separated by a hop size of N/g, is constant for powers of them up to $g-1$ [Puc95].

The equation

$$X[n_a^u, k] = \sum_{n=0}^{N-1} \tilde{x}_u[n] e^{-\jmath 2\pi \frac{k}{N} n} \qquad (3)$$
$$\text{with} \quad x_u[n] = h_a[n] x[n - n_a^u]$$

yields a frequency representation of the windowed input frame u at position n_a^u. $X[n_a^u, k]$ is now both depending on time (via n_a^u) and frequency (via k). Successive Fourier Transforms of a signal are also called *Short-Time Fourier Transform*s (STFTs). Eq. (3) provides the basic mathematical framework for the time-frequency representation of a given input signal $x[n]$ in frames of length N at consecutive positions n_a^u.

Referring to the term $\tilde{x}_u[n]$ in Eq. (3), one important remark in terms of implementation of the phase vocoder should be explicitly pointed out: Since the analysis window is necessarily symmetric around $N/2$, this operation will incorporate a linear phase contribution of $e^{\jmath \pi k}$ to the spectral representation[4]. This becomes obvious after inspecting [OS09c, table 8.2: property 13, property 5] where it is stated that a series of even symmetry results in a real-valued DFT and a circular shift in the time domain imposes a phase shift on the Fourier Transform. To avoid this impractical phase jumps across the frequency bins, the windowed input frame $x_u[n]$ is again circularly shifted by $N/2$ samples (denoted as $\tilde{x}_u[n]$) to compensate this phase shift. In formal terms, this circular shift of a sequence $x[n]$ is defined as

$$\tilde{x}[n] = x[((n - N/2))_N], \qquad \tilde{\tilde{x}}[n] = x[n] \qquad (4)$$

where $((\cdot))_N$ denotes the modulo operation and N is the length of the sequence and must be even. This modification is common practice and has no effect on the output signal as long as the output signal frames are circularly shifted by $N/2$ samples again before performing the overlap-add procedure.

2.2.2 Processing Stage

In virtually all cases, the result from the DFT has to be converted into polar coordinates in order to permit the desired modifications in an appropriate way as magnitudes and phases:

$$r[n_a^u, k] = \left| X[n_a^u, k] \right| ,$$
$$\varphi[n_a^u, k] = \angle X[n_a^u, k] .$$

The processing stage involves individual algorithms and has basically nothing in common with the structural definition of the phase vocoder. Therefore, at this stage, those opera-

[4]This phase contribution of $e^{\jmath \pi k}$ is not immediately visible if the Fourier-Transform of $h_a[n]$, $H_a[k]$, is inspected solely; it will be perceived as a ± 1 alternation since the Fourier Transform of an even symmetric signal is entirely real. But the phase jumps will turn out to be inconvenient when the spectrum becomes complex-valued.

tions are simply denoted as the transition

$$X[n_a^u, k] \longmapsto X'[n_a^u, k]$$

and it is referred to the particular descriptions in section 3 for concrete examples.

2.2.3 Synthesis Stage

If the phase vocoder is utilized to perform effects that involve the inequality $R_a \neq R_s$ (where R_s is termed *output hop size* or *synthesis hop size*), it is advisable to perform some phase adjustments, as it is explained in Section 3.1 more comprehensively. These phase updates cause for their parts the transition

$$X'[n_a^u, k] \longmapsto Y[n_s^u, k]\,.$$

Finished with the optional phase update, synthesizing the STFT frames back to the time domain is performed analogically to the analysis stage. One difference is that the analysis hop size n_a^u and synthesis hop size n_s^u are potentially unequal since R_s may differ from R_a. The same applies for the synthesis window $h_s[n]$ which can differ from the analysis window as long as the windowing constraints, as explained above, are met.

The output signal $y[n]$ is then an overlap-added sum of delayed time-domain frames [Cro80].

$$y[n] = \sum_{u=0}^{\infty} h_s[n - n_s^u] y_u[n - n_s^u] \tag{5}$$

$$\text{with} \quad \tilde{y}_u[n] = \frac{1}{N} \sum_{k=0}^{N-1} Y[n_s^u, k] e^{j2\pi \frac{k}{N} n}$$

Here, the tilde again denotes that $y_u[n]$ is $\tilde{y}_u[n]$, circularly shifted by $N/2$ samples (cf. Eq. (4)).

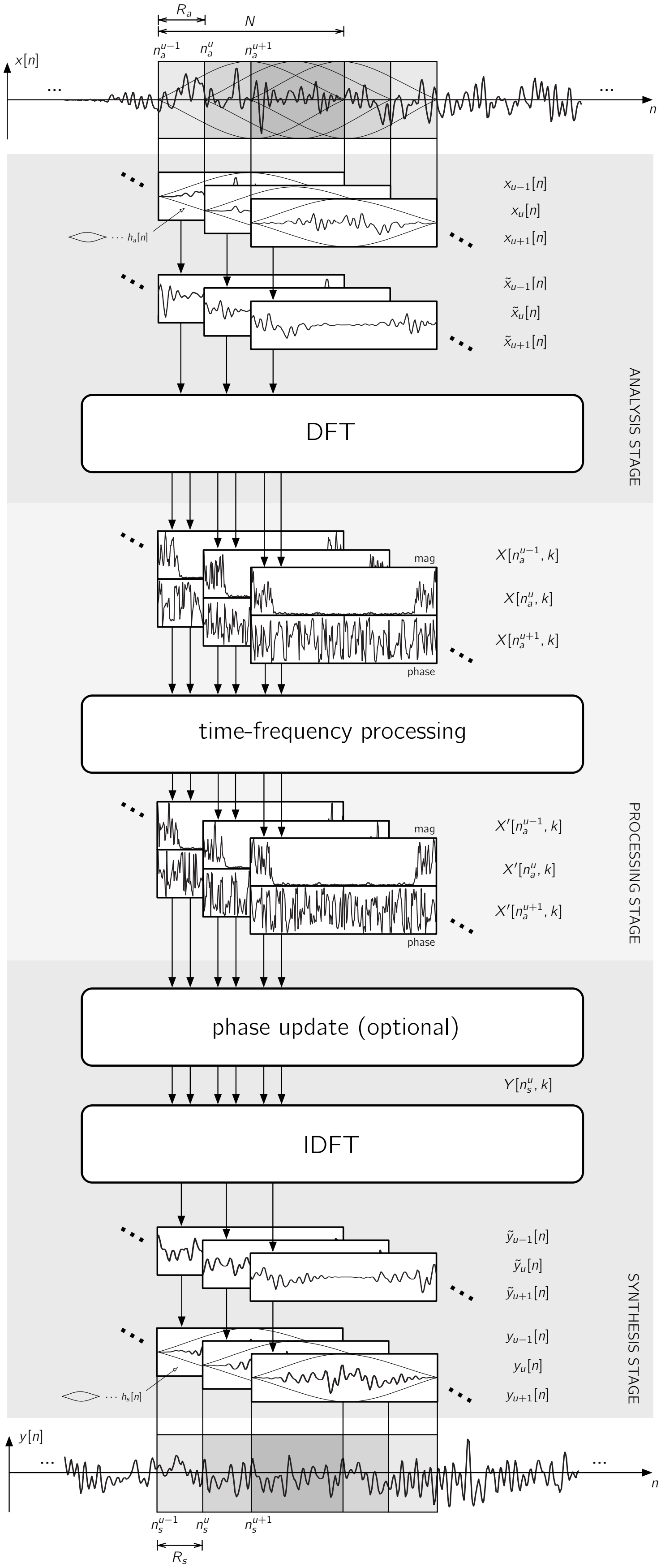


Figure 2: A sketch delineation of the different phase vocoder stages.

3 Audio Effects with the Digital Phase Vocoder

In this section, some of the most common audio effects that can be realized with the phase vocoder are presented. Strictly speaking, most of them refer to the *channel vocoder* which differs from the phase vocoder in not focusing on the phase evolution over time but merely operating on the vocoder channels (i.e. frequency bins). This is the reason why the term *vocoder* itself is often used to neglect a further specification whether the phase or channel vocoder is meant.

The most challenging effects, however, are those implemented by the phase vocoder: *time stretching* (discussed in section 3.1) and *pitch transposition*[5] (discussed in section 3.2) are explicitly dealing with the phase propagation over time. A lot of issues are introduced by manipulating temporal phase information; a discussion of uprising problems is provided in section 3.1.2.

Famous and very well known channel vocoder effects are for example the *mutation between sounds* where commonly a synthesizer sound is modulated by a voice thus locking the voice to the harmonies of the synthesizer. Audio effects gained with the channel vocoder are described in section 3.3.

Since the channel vocoder itself is merely a skeletal structure of the phase vocoder and all effects discussed can be implemented by the phase vocoder, the latter was chosen as a suitable term for this thesis.

3.1 Time Stretching

Research on the art of *time stretching* a signal has been of interest for a long time. When a signal is stretched in time, changes of the temporal evolution are made, whereas the pitch of the signal must not be altered. It is clear that this effect cannot be applied in real time, but it provides an important step towards pitch shifting, as discussed in section 3.2.

3.1.1 Underlying Model

In this section, the basic idea of how time stretching can be accomplished is deduced [LD99a]. Firstly, the *sum of sinusoids* model (cf. 2.1.2) is assumed, where the input signal is decomposed into a certain number of complex exponentials[6] at instant n

$$x[n] = \sum_{k_r=1}^{I[n]} A[n, k_r] e^{\jmath \varphi_a[n,k_r]} \quad n \geq 0$$

$$\varphi_a[n, k_r] = \varphi_a[0, k_r] + \sum_{m=1}^{n} \omega_a[m, k_r] \tag{6}$$

where the amount of sinusoids is time-variant and denoted by $I[n]$, and each signal is associated with an index k_r. The terms $\varphi_a[n, k_r]$ and $\omega_a[n, k_r]$ refer to the *instantaneous phase* and *instantaneous frequency* to be determined by this algorithm. It is important

[5] The terms *pitch transposition* and *pitch shifting* are commutable in this thesis as well as the terms *time scaling* and *time stretching*.

[6] In this thesis, the terms *complex exponential* and *sinusoid* both refer to a complex-valued exponential sequence and are contemplated as equivalent.

to notice that the magnitude and the frequency of the sinusoid can vary over time, being consistent with real-world assumptions where a spectral component of an arbitrary signal may not remain constant over time.

This sum of sinusoids shall now be expanded in matters of temporal development. A straight forward approach is to varying the synthesis hop size R_s (cf. section 2.2.3), while keeping R_a constant (cf. section 2.2.1), yielding a *time stretching factor* α

$$\alpha = \frac{R_s}{R_a} . \tag{7}$$

The perfectly stretched synthesized signal is now (with $\varphi_s[n, k_r]$ being the synthesis phase)

$$y[n] = \sum_{k_r=1}^{I[n]} A[n, k_r] e^{\jmath\varphi_s[n,k_r]} \quad n \geq 0 . \tag{8}$$

It is remarked that the magnitude values, despite of the phase values, have not to be changed. However, the phase term $\varphi_s[n_s^u, k_r]$ at a synthesis instant n_s^u can be derived by taking the analysis phase $\varphi_a[n_a^u, k_r]$ at the corresponding analysis instant and regarding that the synthesized signal lasts α times as long as the input signal, but with preserved frequencies. Since the relationship between phase and frequency and thus time is linear, the phase advances by the factor α too. This phase propagation between $n = 0$ and $n = n_s^u$ must be added to the initial synthesis phase $\varphi_s[0, k_r]$.

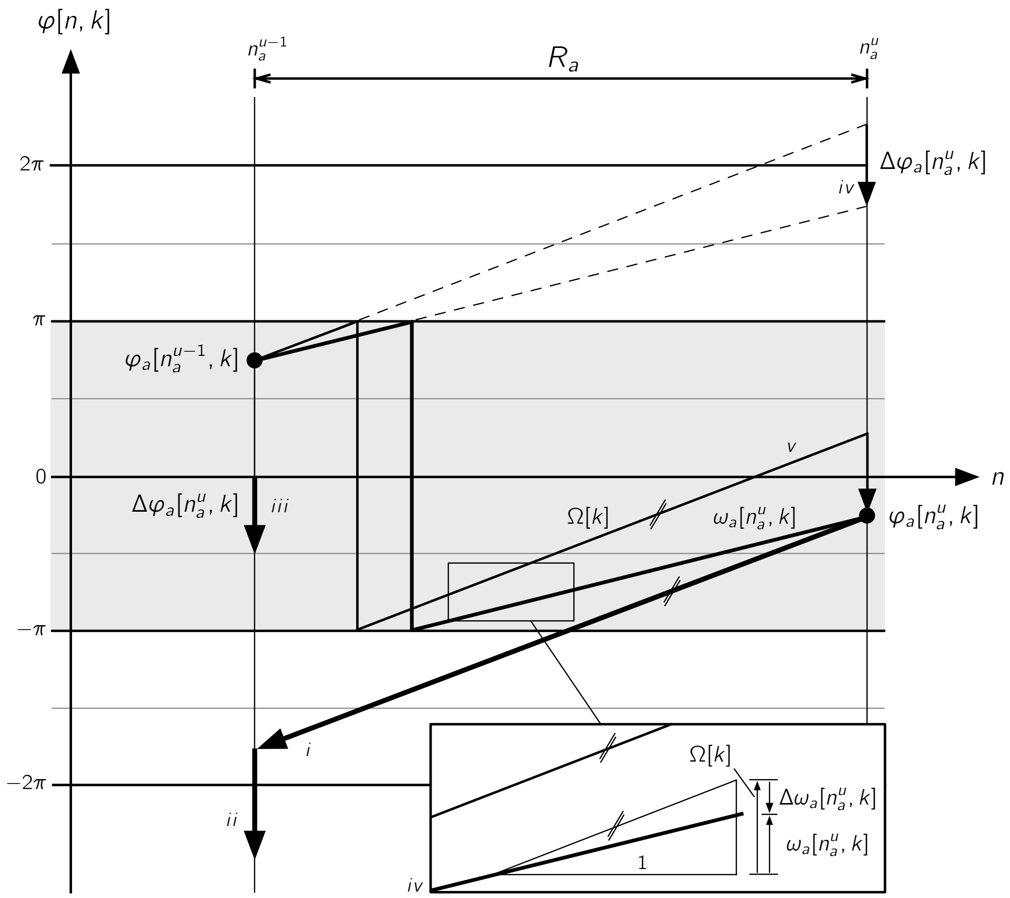
$$\begin{aligned}
\varphi_s[n_s^u, k_r] &= \varphi_s[0, k_r] + \alpha\varphi_a[n_a^u, k_r] \\
&= \varphi_s[0, k_r] + \alpha \sum_{m=1}^{n_a^u} \omega_a[m, k_r] \\
&= \varphi_s[0, k_r] + \alpha \sum_{m=1}^{n_a^u} \Big(\varphi_a[m, k_r] - \varphi_a[m-1, k_r]\Big) \\
&= \varphi_s[0, k_r] + \alpha\Big(\varphi_a[n_a^u, k_r] - \varphi_a[0, k_r]\Big)
\end{aligned} \tag{9}$$

In section 3.1.2, it will be shown that for integer values of α, the choice of $\varphi_s[0, k_r]$ is crucial to the quality of the output signal. Furthermore, it is noted that Eq. (9) is a strictly analytical statement and does not deal with phase wrapping into the principal domain around $\pm\pi$ which is implicitly introduced by the DFT/FFT.

As it might have been noticed, the synthesis phase is given in roughly quantized steps of R_s, so one may claim the phase values between these steps for a full representation. But as the synthesis is carried out in steps of size R_s, it is not necessary to define those values explicitly. It is an inherent property of the STFT that the phase values between synthesis steps are linearly interpolated, amounting the real frequency of the sinusoid belonging to the respective bin.

Phase Unwrapping and Instantaneous Frequency. It is now required to determine the instantaneous frequencies of the observed sinusoids to model the output signal. It should be observed that Eq. (9) looks as if this was an easy step; unfortunately it is not.

Since the phase values determined in Eq. (5) are implicitly wrapped around $\pm\pi$, the correct phase difference can not be estimated by simply subtracting two successive phase values; an additional step must be incorporated – *phase unwrapping*.



i ... $\varphi_a[n_a^u,k] - R_a\Omega[k]$

ii ... $\varphi_a[n_a^u,k] - R_a\Omega[k] - \varphi_a[n_a^{u-1},k]$

iii ... $\Delta\varphi[n_a^u,k] = \arg_\mathrm{p}\left(\varphi_a[n_a^u,k] - R_a\Omega[k] - \varphi_a[n_a^{u-1},k]\right)$

iv ... instantaneous frequency decomposition:

$$\omega_a[n_a^u,k] = \Omega[k] + \Delta\omega_a[n_a^u,k]$$

with $\Delta\omega_a[n_a^u,k] = \Delta\varphi[n_a^u,k]/R_a$

v ... graphical verification:

$$\begin{aligned}\varphi_a[n_a^u,k] &= \arg_\mathrm{p}\left(\varphi_a[n_a^{u-1},k] + R_a\Omega[k] + \Delta\varphi_a[n_a^u,k]\right)\\ &= \arg_\mathrm{p}\left(\varphi_a[n_a^{u-1},k] + R_a\omega_a[n_a^u,k]\right)\end{aligned}$$

Figure 3: The phase unwrapping procedure.

With Fig. 3 as a graphical sketch in mind, the phase unwrapping algorithm for estimating the instantaneous frequency can be deduced. After the analysis stage and processing stage, for two successive STFTs at instants n_a^{u-1} and n_a^u two different phases values $\varphi_a[n_a^{u-1},k] = \angle X'[n_a^{u-1},k]$ and $\varphi_a[n_a^u,k] = \angle X'[n_a^u,k]$ are passed to the synthesis stage[7]. As mentioned earlier, this phase difference contributes to the *real* frequency of the sinusoid

[7] It should be remarked that now the *channels* k of the phase vocoder are considered – no longer the real sinusoids k_r of the signal. This is justified because the real sinusoids are modeled by a linear combination of the vocoder channels, as the Fourier Transform proposes.

in channel k of the phase vocoder.

The *real* phase propagation $R_a\omega_a[n_a^u, k]$ of a sinusoid in channel k of the N-point DFT between the instants $n_a^u - n_a^{u-1} = R_a$ is now divided into two parts, namely the *nominal* phase propagation $R_a\Omega[k] = R_a 2\pi k/N$ and the *additional* phase propagation $\Delta\varphi_a[n_a^u, k]$:

$$R_a\omega_a[n_a^u, k] = R_a\Omega[k] + \Delta\varphi_a[n_a^u, k] \tag{10}$$

or, in frequency notation

$$\omega_a[n_a^u, k] = \Omega[k] + \Delta\omega_a[n_a^u, k]\,. \tag{11}$$

As stated earlier, $\omega_a[n_a^u, k]$ is termed *instantaneous frequency* and has to be determined. In order to accomplish that, the additional phase difference $\Delta\varphi_a[n_a^u, k]$ between the two instants n_a^{u-1} and n_a^u is computed:

$$\Delta\varphi_a[n_a^u, k] = \arg_\mathrm{p}\Big(\varphi_a[n_a^u, k] - R_a\Omega[k] - \varphi_a[n_a^{u-1}, k]\Big)\,. \tag{12}$$

Here, the phase at instant n_a^u, $\varphi_a[n_a^u, k]$, is *rolled back* R_a samples at the nominal frequency $\Omega[k]$ of the vocoder channel k (cf. step *i* in Fig. 3). After this, the original phase at instant n_a^{u-1}, $\varphi_a[n_a^{u-1}, k]$, is subtracted (cf. step *ii* in Fig. 3). The result must be taken into the principal domain within $\pm\pi$ to get a valid result, indicated by the operator $\arg_\mathrm{p}(\cdot)$

$$\arg_\mathrm{p}(\vartheta) = \vartheta - \left\lfloor \frac{\vartheta}{\pi} \right\rfloor \pi \qquad\qquad \lfloor x \rfloor = \{\, \check{x} \;\big|\; \check{x} \in \mathbb{Z},\; x - 1 < \check{x} \leq x \,\}\,. \tag{13}$$

The necessity of "backwrapping" to the principal domain can be observed in Fig. 3, step *iii*. As the nominal phase propagation $\Omega[k]$ does only cancel (i.e. the phase propagation amounts to an integer multiple of 2π) in channels with the property

$$\begin{aligned} k\frac{2\pi}{N}R_a &= 2\pi m \qquad m \in \mathbb{Z} \\ k &= m\frac{N}{R_a}\,, \end{aligned} \tag{14}$$

the phase values between successive STFT frames most likely vary per se, eliminiating the possibility of a straight forward instantaneous frequency determination. Only for values $R_a = N$ (which is nonsense for the phase vocoder), all nominal phase propagation values are equal to integer multiples of 2π, which is perspicuous in knowledge of the fact that the bins of the DFT are located at frequencies of $k2\pi/N$.

Finally, the additional frequency contribution $\Delta\omega_a[n_a^u, k]$ is computed as (cf. step *iv* in Fig. 3)

$$\Delta\omega_a[n_a^u, k] = \frac{\Delta\varphi_a[n_a^u, k]}{R_a}\,. \tag{15}$$

Now, the instantaneous frequency can be calculated according to Eq. (11). This is essential for estimating the phase update of the output signal, especially for varying synthesis hop sizes. A graphical proof of the presented algorithm can be found in step *v* in Fig. 3.

Phase Propagation Formula. Recalling that time stretching means altering the duration of a signal without changing its pitch, the output phase is updated in each synthesis step according to Eq. (9):

$$\begin{aligned}\varphi_s[n_s^u, k] &= \varphi_s[n_s^{u-1}, k] + R_s\omega_a[n_a^u, k],\\ \angle Y[n_s^u, k] &= \angle Y[n_s^{u-1}, k] + R_s\omega_a[n_a^u, k]\,.\end{aligned} \tag{16}$$

However, the magnitude values are simply passed through:

$$\left|Y[n_s^u, k]\right| = \left|X'[n_a^u, k]\right| \tag{17}$$

3.1.2 Drawbacks, Issues and Solutions

There are several drawbacks of the standard time stretching algorithm which have to be taken into account properly in order to maintain a result of decent quality. A lot of research has been done on the main issues – the *horizontal* and *vertical phase coherence* – which are discussed below [LD99a] [Puc95] [LD97].

Horizontal Phase Coherence

This term denotes the requirement that the phase values of successively synthesized output frames must be consistent with each other, i.e. Eq. (16) must be fulfilled.

If the conditions arising from Eq. (16) are not met, the output signal is impaired, ranging from audible artifacts named *phasiness* and *reverberation* to complete distortion. Fortunately, horizontal phase coherence can be easily maintained since the requirements are fulfilled per se if the preceding algorithm is applied.

Vertical Phase Coherence

This issue is much more sophisticated and still a matter of research. In contrast to horizontal phase coherence, where the phase values of successive STFT frames are regarded, vertical phase coherence addresses the phase consistency *across* the frequency bins. Special difficulties arise, when fractional stretching factors and nonstationary input signals are processed. For example, if the frequency components change over time, they switch the channel they are associated with, leading to even more artifacts due to so-called *phase jumps*.

In this section, at first the theoretical background will be elucidated. Afterwards, the resulting problems are sketched and possible solutions are presented. Unfortunately, no unmitigated solution does exist to get rid of vertical phase consistency problems – artifacts such as phasiness, reverberation or modulation will always arise, even though modern algorithms suppress them very well.

Theoretical Background. Based on Eq. (16), the accumulated output phase can be written as

$$\angle Y[n_s^u, k] = \angle Y[0, k] + \sum_{\nu=1}^{u} R_s\omega_a[n_a^\nu, k] \tag{18}$$

and then, after using Eq. (11) and Eq. (15) to express the instantaneous frequency in terms of $\Omega[k]$ and $\Delta\varphi_a[n_a^\nu, k]$, it can be rewritten as

$$\angle Y[n_s^u, k] = \angle Y[0, k] + \sum_{\nu=1}^{u} \left(R_s\Omega[k] + \frac{R_s}{R_a} \Delta\varphi_a[n_a^\nu, k] \right) . \tag{19}$$

Eq. (12) is now inserted, which yields

$$\angle Y[n_s^u, k] = \angle Y[0, k] + \sum_{\nu=1}^{u} \Bigg(R_s\Omega[k] + \frac{R_s}{R_a} \operatorname{arg_p} \Big(\angle X'[n_a^u, k] - R_a\Omega[k] - \angle X'[n_a^{u-1}, k] \Big) \Bigg)$$

$$\angle Y[n_s^u, k] = \angle Y[0, k] + \sum_{\nu=1}^{u} \Bigg(R_s\Omega[k] + \frac{R_s}{R_a} \Big(\angle X'[n_a^\nu, k] - R_a\Omega[k] - \angle X'[n_a^{\nu-1}, k] + 2m[n_a^\nu, k]\pi \Big) \Bigg) \tag{20}$$

and simplifies to

$$\angle Y[n_s^u, k] = \angle Y[0, k] + \alpha \Big(\angle X'[n_a^u, k] - \angle X'[0, k] \Big) + \alpha \sum_{\nu=1}^{u} 2m[n_a^\nu, k]\pi \tag{21}$$

where α is consistent to Eq. (7). The function $\operatorname{arg_p}(\cdot)$ is resolved by incorporating an additional term $\sum_{\nu=1}^{u} 2m[n_a^\nu, k]\pi$ that adopts its business. The values of m are integer since the phase value and its pendant in the principal domain $\pm\pi$ are always distinct from each other by integer multiples of 2π.

With Eq. (21), the phase propagation algorithm is put into a form where the direct relationship between input and output phases is stated only in terms of STFT values. One interesting fact is that formally there is no error propagation possible by the STFT values themselves, since they need not to be accumulated. On the other hand, error propagation can occur when one of the unwrapping factors was wrongly determined.

For α being an integer, the term $\alpha \sum_{\nu=1}^{u} 2m[n_a^\nu, k]\pi$ vanishes. This is equivalent to the statement that the computationally expensive phase unwrapping procedure can be dropped. Thus, integer stretching factors simplify the investigation of coherence problems a lot. Unfortunately, integer values are not the general case.

A special remark regards the similarity between Eq. (21) and Eq. (9). In the latter one, the phase unwrapping property is intrinsic. Now it is also formally clear why it was not allowed to simply utilize Eq. (9) in order to calculate the phase propagation.

Concluding this observations, the following influences on vertical phase coherence can be drawn from Eq. (21):

- The initialization phase $\angle Y[0, k]$. Some possibilities to set it up properly are discussed subsequently.
- Errors in the accumulated phase unwrapping factors $\sum_{\nu=1}^{u} 2m[n_a^\nu, k]\pi$. The reason why this term contributes especially to the potential loss of *vertical* phase coherence, is that phase unwrapping errors can originate from mutual influences of adjacent channels. As it is known, sinusoidal components of a

signal most likely spread their energy over several channels. Even with proper windowing of the input signal, one sinusoid commonly influences more than one channel. Especially in complicated signals like speech and audio, it is very likely that such interferences between channels occur.

Another issue that contributes to phase unwrapping errors is that it can not be expected for a sinusoidal component to keep its frequency constant over time. As time progresses, it will certainly be associated with different channels, engendering another difficulty of estimating the true instantaneous frequency. One straight forward thought to keep this incidence small is to choose the analysis hop size R_a sufficiently short, decreasing the amount of channels the sinusoid can change from one STFT instance to another.

Solutions and Improvements. In this paragraph, several approaches to deal with the problems mentioned previously are delineated.

Choice of Initial Phase (Integer Stretching Factors only). As it was already discussed, in Eq. (21), the term $\alpha \sum_{\nu=1}^{u} 2m[n_a^\nu, k]\pi$ cancels for integer values of α:

$$\angle Y[n_s^u, k] = \angle Y[0, k] + \alpha\Big(\angle X'[n_a^u, k] - \angle X'[0, k]\Big)\,. \tag{22}$$

This expression can be rewritten as

$$\angle Y[n_s^u, k] = \alpha \angle X'[n_a^u, k] + \underbrace{\angle Y[0, k] - \alpha \angle X'[0, k]}_{\theta[k]} \tag{23}$$

and results in

$$\begin{aligned} \angle Y[n_s^u, k] &= \alpha \angle X'[n_a^u, k] + \theta[k] \\ \theta[k] &= \angle Y[0, k] - \alpha \angle X'[0, k]\,. \end{aligned} \tag{24}$$

The introduced variable $\theta[k]$ is not dependent on time. This facilitates the interpretation of what occurs if a sinusoid migrates from channel k_0 at $n_a^{u_0}$ to channel $k_0 + 1$ at $n_a^{n_0+1}$ – which is very likely. It is clear that this sinusoid will experience a phase jump of $\theta[k_0 + 1] - \theta[k_0]$ since the term $\theta[k]$ represents the constant phase offset of channel k (cf. Eq. (24)).

Based on this interpretation, for integer stretching factors the vertical phase coherence can be maintained by arrogating

$$\theta[k] = \angle Y[0, k] - \alpha \angle X'[0, k] \overset{!}{=} C \qquad C \ldots \text{constant} \tag{25}$$

and defining the initial setup rule for the output phase

$$\angle Y[0, k] = C\alpha \angle X'[0, k]\,. \tag{26}$$

Quality improvements from -10 dB to -25 dB can be reached by employing this condition [LD99a][8]. However, for noninteger stretching factors, the situation is much more complicated.

[8] A measurement of the consistency of the output signal $y[n]$, with its respective N-point STFT synthesis

Loose Phase-Locking. This approach exploits the phase relationships of adjacent channels. In respect to the assumption of an underlying sinusoid and supposing that adjacent channels are out of phase[9] by $\Delta k\pi$ due to even symmetry around $N/2$ of the windowing function (cf. 2.2.1) [Puc95], the following phase update formula is proposed:

$$\angle Y[n_s^u, k] = \angle(-X'[n_a^u, k-1] + X'[n_a^u, k] - X'[n_a^u, k+1])\,. \tag{27}$$

The very straight forward interpretation of this rule is: If the component $X'[n_a^u, k]$ prevails in terms of magnitude, the value $\angle(-X'[n_a^u, k-1]+X'[n_a^u, k]-X'[n_a^u, k+1])$ approximately amounts to its phase again. This means that if $X'[n_a^u, k]$ and $Y[n_s^u, k]$ are succeeding peaks in channel k, their phase value will be passed through. On the other hand, if $Y[n_s^u, k]$ is adjacent to a peak, it will receive the phase of the peak with an offset of π. It is referred to Fig. 4 on page 21 for a graphical sketch delineation to get a better understanding of the concept and how it works.

The algorithm of loose phase-locking is very considerable in terms of computational complexity. First, it is not necessary to circularly shift the windowing function because it is desired for the channels to have an offset of π. Secondly, only a few additional calculations per STFT channel are necessary.

Loose phase-locking can be easily implemented and shows a good performance on synthetic tests. Unfortunately, its application on speech or music signals doesn't yield a dramatic improvement of the phasiness [LD99a].

Rigid Phase-Locking: Identity Phase-Locking. This model introduces an improvement above the former one by identifying the peaks of the underlying sinusoids. A rather simple but effective and sufficiently performant peak detection would be to identify a sample as a peak if its two neighbors on each side are smaller. In the next step, the spectrum is divided into *regions of influence* of each peak. These regions can either be separated by the nearest neighbor principle (frequency bin belongs to nearest peak) or by samples of minimum value.

The idea of this phase-locking scheme was proposed independently in [Fer99] and [QDH95]: Its aim is to preserve the phase relation to the peak within its region of influence. The formula is given as

$$\begin{aligned}\angle Y[n_s^u, k] - \angle Y[n_s^u, k_p] &= \angle X'[n_a^u, k] - \angle X'[n_a^u, k_p]\\ \angle Y[n_s^u, k] &= \angle Y[n_s^u, k_p] + \angle X'[n_a^u, k] - \angle X'[n_a^u, k_p]\end{aligned} \tag{28}$$

frames $Y[n_s^u, k]$ was introduced in [LD99a], based on [GL84]:

$$D_M = \frac{\sum_{u=P}^{U-P-1}\sum_{k=0}^{N-1}\left(\left|Z[n_s^u, k]\right| - \left|Y[n_s^u, k]\right|\right)^2}{\sum_{u=P}^{U-P-1}\sum_{k=0}^{N-1}\left|Y[n_s^u, k]\right|^2}\,.$$

$Z[n_s^u, k]$ are STFT frames, taken from the time-domain output signal again. An offset P is given, limiting the amount of total STFT frames U due to the fact that errors can be introduced at the beginning and the end of the signal, regardless of its internal consistency.

This way of consistency measurement addresses the fact that the synthesis stage can produce a complex-valued rather than a real-valued signal $y[n]$. Projecting this signal onto the real axis (which is necessarily performed since audio signals are real-valued) is one reason why artifacts are introduced. Unfortunately, no measurement exists yet that directly addresses the phasiness of a signal.

[9] The circular shift proposed in 2.2.1 is not applied here, as mentioned later.

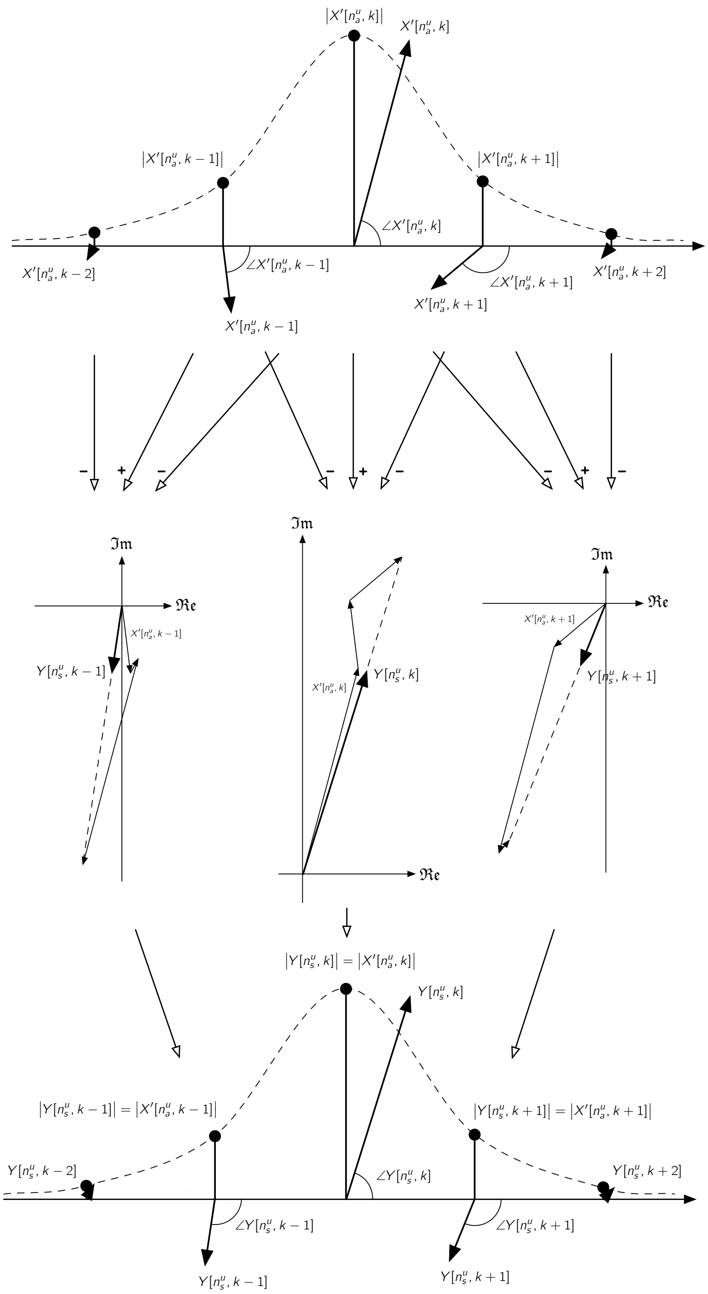


Figure 4: Figurative explanation of the loose phase-locking procedure. The complex vectors are informally plotted onto the frequency bins, being graphically added in step two. The *phase* of the resulting vector is then imposed as a final phase on the output vectors. The reader may perceive the approximate $\pm\pi$-offset of channels adjacent to $Y[n_s^u, k]$.

where the resulting angle $\angle Y[n_s^u, k]$ of channel k, associated with peak k_p by its region of influence, consists of the phase of the peak $\angle Y[n_s^u, k_p]$ – which is left to be determined – and the phase difference from the input domain $\angle X'[n_a^u, k] - \angle X'[n_a^u, k_p]$.

Only one trigonometric and phase unwrapping calculation per peak channel is necessary. The rest of the phase propagation can be computed by a complex multiplication which results directly from Eq. (28):

$$Y[n_s^u, k] = Y[n_s^u, k_p] \cdot e^{\jmath(\angle X'[n_a^u, k] - \angle X'[n_a^u, k_p])} \tag{29}$$

where the phase of the peak $Y[n_s^u, k_p]$ still needs to be computed by phase unwrapping. Further explanations and examples of this algorithm can be found in [LD97] [LD99a]. In Fig. 5, the process of identity phase-locking is graphically sketched.

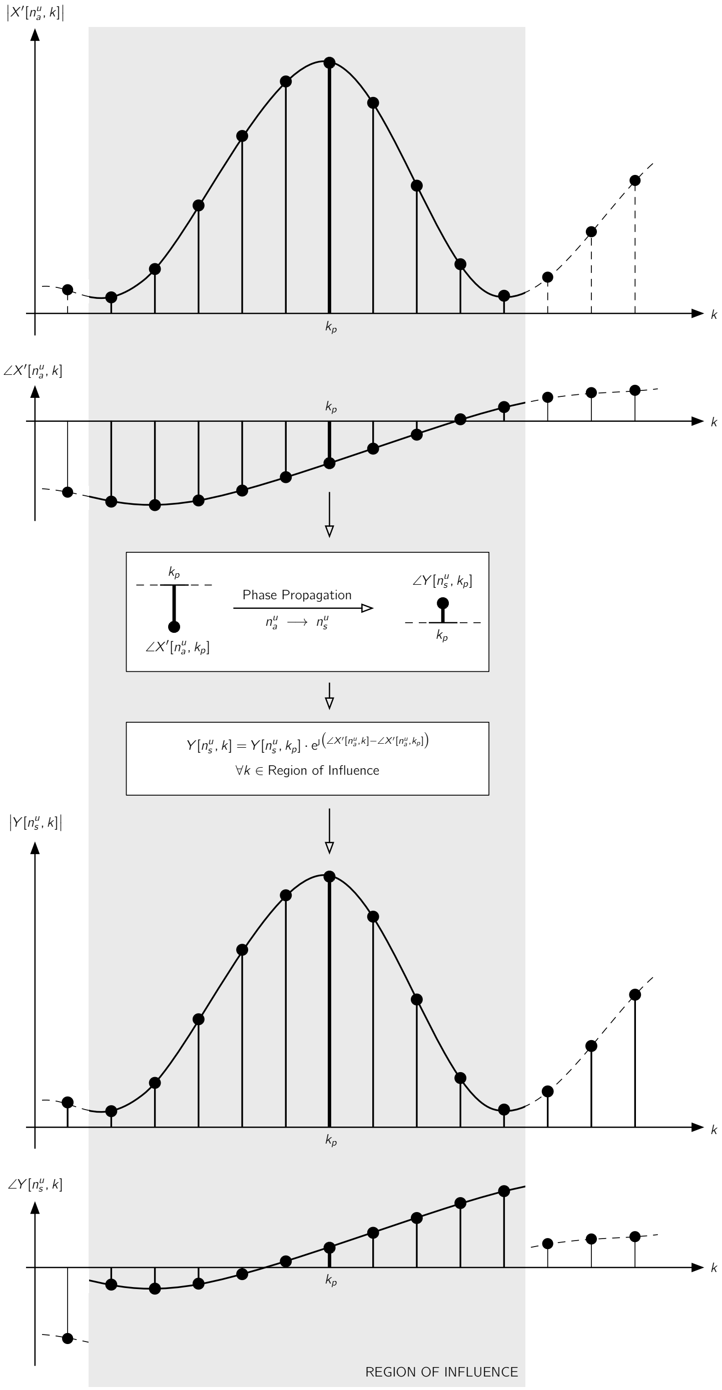


Figure 5: Schematic picture of identity phase-locking. After detecting the region of influence (shaded with gray), the phase of the peak bin at position p_k is normally propagated (upper box). All the other channels can then be related to the peak's phase due to the assumption that the phase relations across the region of influence are preserved over time.

Rigid Phase-Locking: Scaled Phase-Locking. This algorithm was proposed in [LD97] and is more spanningly described in [LD99a]. It takes into account that sinusoids can switch the channels they are associated with, which leads to the necessity of updating the phase propagation formula (Eq. (16)) to

$$\angle Y[n_s^u, k] = \angle Y[n_s^{u-1}, k - \Delta k] + R_s \omega_a[n_a^u, k]\,. \tag{30}$$

Eq. (30) models a transition of a sinusoid by Δk channels between the instants n_s^{u-1} and n_s^u. The maintenance of the phase relations within a region of interest can be written as a generalization of Eq. (28):

$$\angle Y[n_s^u, k] = \angle Y[n_s^u, k_p] + \beta\big(\angle X'[n_a^u, k - \Delta k] - \angle X'[n_a^u, k_p - \Delta k]\big) \tag{31}$$

Since there is little theoretical background, a formal algorithm to derive the best value for β does not exist. One basic problem is that it is not yet analytically possible to give a measurement of the phasiness.

However, informal listening tests have shown that $\beta \approx 2/3 + \alpha/3$ produces good results, far better than identity phase-locking [LD99a].

Furthermore, it should be mentioned that the need of a peak-following algorithm arises from this approach.

Reconstruction from Magnitude. Phase values of a signal spectrum can be recovered from its magnitude values, but with high computationally costs since it is an iterative procedure [GL84]. Nevertheless, a promising real time approach was proposed in [ZBW07], but this is beyond the scope of this thesis.

3.2 Pitch Transposition

When a signal is *pitch transposed*, its frequencies are multiplied by a constant factor α. This preserves the harmonies and the pitch transposed signal is perceived by the human ear as being consistent with the original signal. In respect of the original frequency ω, the individual shift amounts to

$$\Delta\omega(\omega) = \omega(\alpha - 1)\,. \tag{32}$$

In contrast to pitch shifting, *frequency shifting* imposes a constant frequency transposition $\Delta\omega = \text{const.}$ to all frequencies. Here, the harmonies are destroyed and the signal is noticed to be distorted.

When a signal is pitch or frequency shifted, its temporal evolution is forced to remain the same. This is the reason why direct *resampling* does not achieve the aimed effect[10], but with incorporating time stretching, it can do so. Indeed, the standard technique to pitch shift a signal by a factor α is to first time stretch it by α and then resample it by α. Both operations can be interchanged, leading to different computational complexity: if $\alpha < 1$, the pitch of the signal should be lowered, thus it is advisable to first time stretch it and then resample it in order to save memory. Reversely, if the pitch of a signal should be raised ($\alpha > 1$), the signal may first be resampled and afterwards be stretched. This input-sensitivity of an algorithm is generally considered as a disadvantage.

In the subsequent sections, some approaches addressing pitch transposition are deduced.

[10] A simply resampled signal is either faster and higher pitched or slower and lower pitched.

3.2.1 Standard Approach: Time Stretching and Resampling

As mentioned previously, the common approach towards pitch transposition is to first time scale the signal and then resample it by an arbitrary transposition factor α. Recalling that from Eq. (7) it follows that α must be expressable in terms of a fraction with integer numerator and denominator, resampling can be achieved by first *upsampling* by factor R_a and then *downsampling* by factor R_s.

The process of resampling is described perfectly well in literature and the reader may be referred to [OS09d] [Lyo96].

3.2.2 Alternative Approach: Selective Peak Shifting

In this section, an alternative technique to pitch shift a signal is presented [LD99b]. At first, the theoretical deduction will take place, followed by a complete description of the algorithm itself. It should be noted that this algorithm does not involve any variations of input versus output hop size and is therefore entirely carried out in the processing stage.

Theoretical Background. To get a basic understanding of how this algorithm works, a complex exponential

$$x[n] = e^{\jmath(\omega_0 n + \varphi_0)} \tag{33}$$

with frequency ω_0 and phase offset φ_0 is assumed as an input signal. Furthermore, the input frame $x^u[n]$ of the STFT at instant n_a^u – which is basically a shifted and windowed version of $x[n]$, $h_a[n]x[n + n_a^u]$ – is given as

$$\begin{aligned} x^u[n] &= h_a[n]e^{\jmath(\omega_0(n+n_a^u)+\varphi_0)} \\ &= h_a[n]e^{\jmath\omega_0 n}e^{\jmath(\omega_0 n_a^u+\varphi_0)} \end{aligned} \tag{34}$$

which yields the STFT[11] (cf. frequency shifting theorem [OS09e])

$$X(e^{\jmath\omega})[n_a^u] = H_a(e^{\jmath(\omega-\omega_0)})e^{\jmath(\omega_0 n_a^u+\varphi_0)} \tag{35}$$

with $H_a(e^{\jmath\omega})$ being the frequency response of $h_a[n]$.

It should be observed that the term $e^{\jmath(\omega_0 n_a^u+\varphi_0)}$ is passed through the STFT as a constant because it is not dependent on n.

As a next step, the *frequency shift* by $\Delta\omega(\omega_0) = \Delta\omega_0$ can be performed on the sinusoid, resulting in a substitution of $\omega_0 \to \omega_0 + \Delta\omega_0$:

$$\begin{aligned} Y(e^{\jmath\omega})[n_s^u] &= H_a(e^{\jmath(\omega-(\omega_0+\Delta\omega_0))})e^{\jmath((\omega_0+\Delta\omega_0)n_a^u+\varphi_0)} \\ &= H_a(e^{\jmath(\omega-\omega_0-\Delta\omega_0)})e^{\jmath(\omega_0 n_a^u+\varphi_0)}e^{\jmath\Delta\omega_0 n_a^u} \\ &= X(e^{\jmath(\omega-\Delta\omega_0)})[n_a^u]e^{\jmath\Delta\omega_0 n_a^u} \end{aligned} \tag{36}$$

where Eq. (35) was inserted in the last step[12]. The synthesized parts of the STFT frames, $y^u[n]$, can be represented in time domain as the input frame $x^u[n]$ modulated with $e^{\jmath\Delta\omega_0 n}$, which is consistent with the expression $X(e^{\jmath(\omega-\Delta\omega_0)})[n_a^u]$ in Eq. (36).

[11] Actually, at this stage the DTFT must be considered since continuous pitch shifts of $\Delta\omega$ are performed.

[12] It is remarked, that in this context no additional phase update (as in the preceding sections) is considered and therefore the term Y(·) is used instead of X'(·). Additionally, the terminology of distinguishing between n_a^u and n_s^u is kept – although these terms are equal – to be consistent with preceding sections. Formally, $Y[\cdot, k]$ is always associated with instant n_s^u as $X[\cdot, k]$ and $X'[\cdot, k]$ are always associated with n_a^u, regardless if R_a and R_s are equivalent or not.

Next, the STFT frame $Y(e^{\jmath\omega})[n_s^u]$ is transformed back into the time domain, which results directly from Eq. (36):

$$\begin{aligned} y^u[n] &= x^u[n]e^{\jmath\Delta\omega_0 n}e^{\jmath\Delta\omega_0 n_a^u} \\ &= h_a[n]e^{\jmath\omega_0 n}e^{\jmath(\omega_0 n_a^u+\varphi_0)}e^{\jmath\Delta\omega_0 n}e^{\jmath\Delta\omega_0 n_a^u} \\ &= h_a[n]e^{\jmath((\omega_0+\Delta\omega_0)n+\varphi_0)}e^{\jmath(\omega_0+\Delta\omega_0)n_a^u} \end{aligned} \tag{37}$$

where Eq. (34) was used to resolve $x^u[n]$.

Finally, the whole output signal $y[n]$ is gained by windowing with a synthesis window $h_s[n]$ and overlap-adding the frames $y^u[n]$:

$$\begin{aligned} y[n] &= \sum_u h_s[n-n_s^u]y^u[n-n_s^u] \\ y[n] &= \sum_u h_s[n-n_s^u]h_a[n-n_s^u]e^{\jmath((\omega_0+\Delta\omega_0)(n-n_s^u)+\varphi_0)}e^{\jmath(\omega_0+\Delta\omega_0)n_a^u} \\ y[n] &= \sum_u h_s[n-n_s^u]h_a[n-n_s^u]e^{\jmath((\omega_0+\Delta\omega_0)n+\varphi_0)}e^{-\jmath(\omega_0+\Delta\omega_0)n_s^u}e^{\jmath(\omega_0+\Delta\omega_0)n_a^u} \\ y[n] &= \sum_u h_s[n-n_s^u]h_a[n-n_s^u]e^{\jmath((\omega_0+\Delta\omega_0)n+\varphi_0)} . \end{aligned} \tag{38}$$

Here, $y^u[n-n_s^u]$ was expressed by Eq. (37) and the last two terms were cancelled, recalling that $n_a^u \equiv n_s^u$.

If the windowing constraints

$$\sum_u h_s[n-n_s^u]h_a[n-n_s^u] = 1 \quad \forall n \tag{39}$$

are met, it becomes clear that the output sinusoid is a perfect frequency shift[13] of the input:

$$y[n] = e^{\jmath((\omega_0+\Delta\omega_0)n+\varphi_0)} . \tag{40}$$

Description of the Algorithm. In this paragraph, the necessary steps are shortly sketched to show the operations that have to be performed in order to achieve a pitch shift in respect to the concepts presented above.

The algorithm consists of:

1. Peak detection.
   As mentioned already, an examplary peak detection could be to identify one sample as a peak if its value is bigger than those of its four neighbors.
2. Region of influence estimation.
   The regions of influence can be separated either by the samples in the middle of two peaks or the samples with the lowest value between two peaks.
3. Frequency estimation.
   The algorithm was presented in the continuous sprectrum, but it has to be implemented using the DFT/FFT, which results in a discrete spectrum. Therefore, the real frequencies of the underlying sinusoids have to be estimated.

[13]For single sinusoids, the terms frequency shift and pitch shift are equal. Nevertheless, this proof holds for all signals since every signal can be expressed as a superposition of sinusoids.

One possibility to determine the exact frequency of a sinusoid is to use a Gauss window. If the spectrum is given in decibel (dB), quadratic interpolation can be used. This observation originates from the fact that the Fourier Transform of a Gauss window is a Gauss window again.

More sophisticated techniques for frequency estimation can be found in [PB98].

4. Calculating the frequency shift.
   If a uniform pitch shifting should be performed, then

$$\Delta\omega(\omega) = \omega(\alpha - 1) \tag{41}$$

   has to be set, where α is the pitch shifting factor.

   It is obviously clear, that the frequency shifts of different region of interest need not necessarily correspond to the same shifting factor α.

5. Peak shifting.
   In the general case (noninteger shifts), interpolation techniques must be applied. Fractional time delay algorithms can be used, which have been widely investigated [KJ09] [VL93] [LVKL96]. The simplest technique is linear interpolation.

   Overlapping the STFT frames by 75 % (hop size $N/4$) reduces the artifacts to the borders of perceptibility (-51 dBA) whereas overlapping by 50 % (hop size $N/2$) seems not to be considerable [LD99b]. However, integer shifts are especially simple to handle, so 50 % overlap is sufficient here.

6. Phase adjusting.
   From Eq. (36) it follows that the phase update must involve

$$\angle Y(e^{\jmath\omega})[n_s^u] = \angle X(e^{\jmath(\omega - \Delta\omega_{p_k})})[n_a^u] + \Delta\omega_{p_k} n_a^u \tag{42}$$

   for each peak and its region of influence. The term $\Delta\omega_{p_k}$ denotes the frequency shift which is applied to peak p_k.

   Eq. (42) points out clearly that no trigonometric calculations for the phase update must be performed.

   Another special remark addresses integer shifts (by n bins):

$$\begin{aligned} \Delta\omega_{p_k} n_a^u &= 2\pi\frac{n}{N}\cdot uR_a && n \in \mathbb{N}, \quad N \in \mathbb{N}\ldots\text{DFT size} \\ &= 2\pi\frac{n}{N}\cdot u\frac{N}{m} && u \in \mathbb{Z}, \quad m \in \mathbb{N} \\ &= 2\pi\frac{nu}{m}\,. \end{aligned} \tag{43}$$

   For 50 % overlap ($m = 2$), the phase update simplifies to integer multiples of π, thus making this process trivial.

3.3 The Channel Vocoder

In this section, the channel vocoder itself is investigated more in detail. As it has been pointed out before, the phase vocoder and the channel vocoder differ in the fact that the phase vocoder concentrates more on the phase evolution over time – introducing effects like time stretching and pitch shifting – whereas the channel vocoder focuses more on modifications across the channels.

Some of the uncountable possibilities of introducing audio effects via the channel vocoder are described shortly below, such as the *mutation between sounds*, *dispersion*, *robotization*, *whisperization* and *denoising*.

3.3.1 Mutation between Sounds

The basic principle of mutating sounds via the phase vocoder is to combine two or more input sounds, each contributing a specific part to the amplitude and the phase of the output signal.

This effect, driven with voice and synthesizer, is very popular in electronic music. High-quality and cheap effect devices are available on the market.

Being $X_1[n_a^u, k]$ and $X_2[n_a^u, k]$ the input spectra, the output $X'[n_a^u, k]$ can be combined by these common choices:

Amplitude:

- $\left|X'[n_a^u, k]\right| = \left|X_1[n_a^u, k]\right| \cdot \left|X_2[n_a^u, k]\right|$
  With this setting, the operation on the magnitudes corresponds to a logical *AND*, thus only letting components pass through with non-zero amplitude values of both sounds.
- $|X'[n_a^u, k]| = |X_1[n_a^u, k]| + |X_2[n_a^u, k]|$
  This setup refers to a logical *OR* and lets components pass through if one of both channels is non-zero.
- $|X'[n_a^u, k]| = |X_{\{1,2\}}[n_a^u, k]|$
  This setting assigns the magnitude of either input signal 1 or 2 to the output signal.

Phase:

- $\angle X'[n_a^u, k] = \angle X_{\{1,2\}}[n_a^u, k]$
  Since the phase values contain the temporal structure of a sound, the result will be a signal that earns the characteristics of one of both sounds.
- $\angle X'[n_a^u, k] = \angle X_1[n_a^u, k] + \angle X_2[n_a^u, k]$
  This setting lets the mean phase rotate with double speed.

3.3.2 Dispersion

The origin of this effect lies in an issue of telecommunicational nature – that some frequency bands arrive delayed when a signal is transmitted. This property can be imitated via *group delay*, which is defined as

$$\mathrm{grd}\left[X(e^{\jmath\omega})\right] = -\frac{\mathrm{d}}{\mathrm{d}\omega}\left\{\angle X(e^{\jmath\omega})\right\} \tag{44}$$

and describes the *delay in respect of the frequency* [OS09f].

If linear group delay should be introduced, a quadratic phase term must be imposed on the signal, which is a simple addition in frequency domain. In time domain, this refers to the convolution of the input signal with a chirp signal (sinusoid with constant amplitude and linearly increasing frequency). Therefore, time aliasing effects have to be considered in frequency domain, being crucial to the choice of the window size.

3.3.3 Robotization

This effect results from setting the phase of each STFT to zero. Depending on the STFT size, this adds a robotic flavour to the sound.

3.3.4 Whisperization

This effect is achieved by setting either the phase or the magnitude of the STFT to a random value, which leads especially for small STFT sizes to a whispering effect.

3.3.5 Denoising

Denoising is achieved by applying a nonlinear transfer function to the amplitude spectrum, keeping amplitudes with sufficient high values as they are, while lowering small amplitudes. This can be interpreted as a bank of noise gates, each related to one frequency bin. A basic transfer function may be

$$f(x) = \frac{x^2}{x + c} \tag{45}$$

where c has to be arbitrarily chosen.

One popular noise reduction process is to first calibrate the filter with "silence", where the spectral components of the noise are extracted and the noise gates are properly configured. Further information is available in literature [Cap94] [Vas06].

3.4 Conclusion and Discussion

Regarding the phase vocoder and in particular the presented algorithms for time stretching and pitch shifting, it seems to be needful to draw some conclusions in order to provide a better overview.

The basic algorithm of **phase propagation** was proposed in Section 3.1 as the standard procedure of time stretching utilizing the phase vocoder. Unfortunately, in this basic configuration, it works only well for constant-frequency sinusoids, but not for signals like music and speech.

Furthermore, the described algorithm of phase unwrapping needs a four-quadrant arc tangent function to transform the Cartesian coordinates into polar coordinates which is a computational disadvantage. There exists an approach which utilizes another Fourier Transform instead of trigonometric calculations and phase unwrapping, as proposed in [Puc95].

The drawbacks of potential loss of horizontal and vertical phase coherence were discussed in Section 3.1.2 and some solutions were presented. The simplest approach towards an improvement of this issue was the derivation of a rule for choosing the **initial synthesis phase**. Unfortunately, this rule shows only moderate betterments and only on integer scaling factors. Nevertheless, since the choice of the initial phase is free, it may not be bad to set it according to Eq. (26). For standard analysis windows (Hamming, Hanning etc.), the analysis frames must overlap by at least 75 % in order to yield good results for constant-frequency sinusoids [LD99a].

Another set of possible solutions to the vertical phase coherence problem were presented, such as loose phase-locking and rigid phase-locking, comprizing identity and scaled phase-locking. These approaches utilize intrinsic relationships across the frequency bins around spectral peaks.

Phase-locking in its simplest version, **loose phase-locking**, relates all adjacent channels in general (after the original phase propagation was applied). The advantage is that the circular shift in the analysis and synthesis stage can be dropped since it is appreciated to handle channels with an offset of $\pm\pi$ in respect to each other (cf. Eq. (27)). Another positive side of loose phase-locking is its simplicity – and synchronously its drawback, unfortunately. It shows only moderate improvements on speech or music signals over the standard phase propagation algorithm. At least, it performs well on synthetic signals, such as pure sinusoids with steady or varying frequency [LD99a].

Identity phase-locking on the other side, expands this approach by involving a peak detection stage to apply the phase-locking scheme explicitly on peaks and their surroundings. The advantages of this approach are as follows [LD99a]:

- The performance on a synthetical chirp signal was improved from -6.5 dB without phase-locking to -37 dB, which is quite impressive.
- It is possible to set the analysis hop size to $N/2$, which halves the computational costs comparing to usual hop sizes of $N/4$.
- The regular phase propagation needs to be performed only for peaks. The phase values of other samples within the associated regions of influence can be updated by a single complex multiplication (cf. Eq. (29)).

The last phase-locking scheme, **scaled phase-locking**, extends identity phase-locking in a way that peaks are not only detected, but also followed as time advances. The phase update formula then changes according to Eq. (30). This approach of peak following over time incorporates even more computational costs, but theoretically yields better results. This could be confirmed by informal listening tests, as shown in Section 5.

A time stretched signal, resampled by the same factor yields a pitch shifted signal of the original. Besides this standard technique an alternative approach, **selective peak shifting**, was proposed. As it was already sparsely pointed out, this method offers several advantages towards the standard approach:

- In contrast to the standard pitch shifting technique, the performance of this approach in matters of execution time is not dependent on the shifting factor α.
- Different peaks can be shifted to different locations. This is not possible with the standard technique either.
- No trigonometric calculations need to be performed during the phase update.
- The algorithm is simpler, but nonetheless it incorporates the identity phase-locking scheme, which is considered to yield results of higher quality than algorithms that don't take phase-locking into account.

Besides these improvements over the standard technique, no clear theoretical disadvantages can be found. One requirement, however, is the recognizability of peaks in the spectrum and a clear region of influence. If, for instance, the STFT size is too small, the frequency modulated windows (cf. Eq. (36)) can partially merge, thus impairing magnitude and phase information.

4 MATLAB® Implementation

In this section, the structure of the *MATLAB®* implementation, which was developed during this thesis, is provided. Some details are picked out, and the full source code can be found in Appendix A.

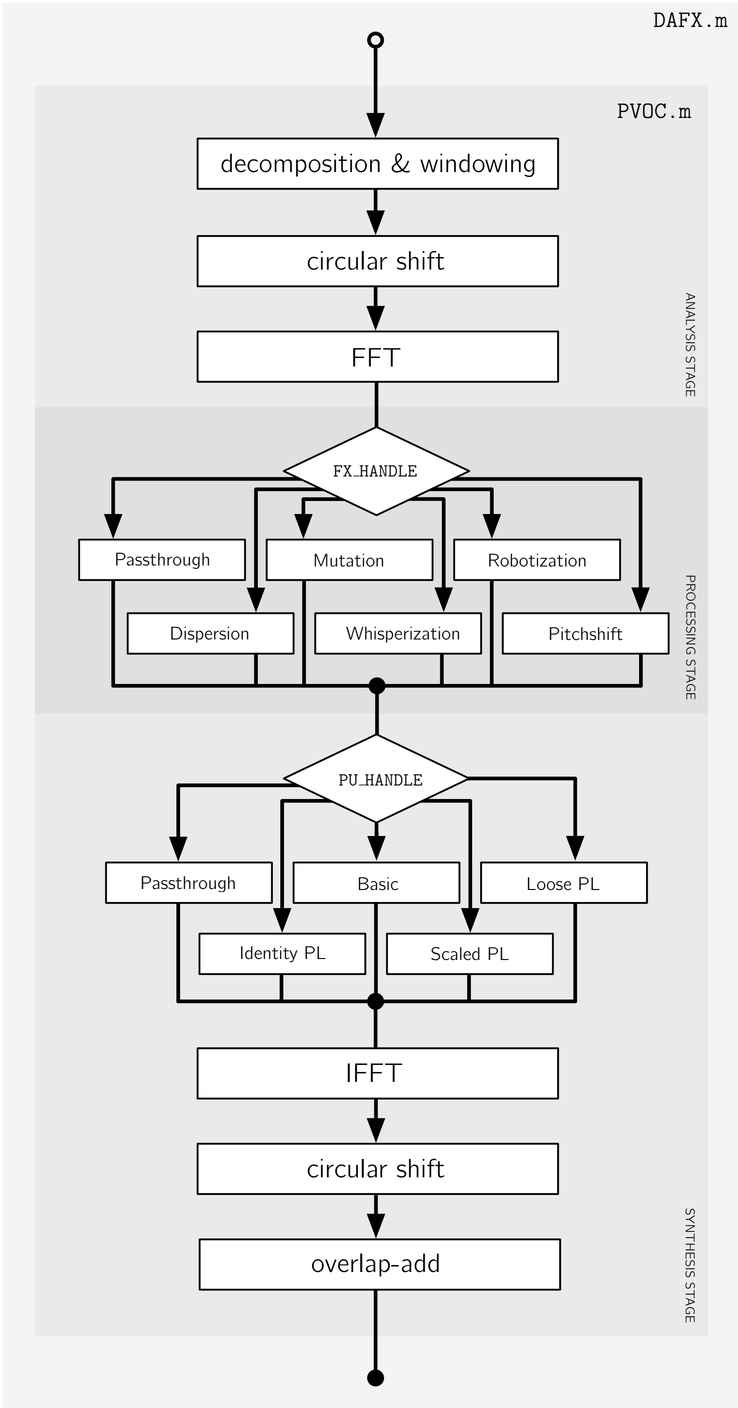


Figure 6: Schematic representation of the implementation in *MATLAB®* .

4.1 Design

To get an overview about the structure of this implementation, a short flow diagram is plotted in Fig. 6. As it is visible, the main script, wrapped around all the other components, is termed `DAFX` and located in the file `DAFX.m`. The full content of this script is listed in Appendix A.1.1. After configuring the phase vocoder (i.e. setting the effect and phase update function handles and defining the parameters), the phase vocoder function `PVOC()` is called, which is listed in Appendix A.1.2.

The configuration towards a particular effect is realized via submission of function handles to the phase vocoder script. This way facilitates a modular and extendable design.

4.2 Time Stretching and Pitch Shifting

Time stretching consists of different phase update algorithms, of which the implementation source codes are provided in Appendix A.2.

The functions that achieve the pitch shifting effect are listed in Appendix A.3. For the standard pitch transposition technique, the necessity of resampling was implemented in the function `PVOC()`, as given in Appendix A.1.2. Furthermore, the source code of the alternative approach of selective peak shifting is provided in Appendix A.3.1. The reader may note that this function is not carried out as a phase update procedure but as an audio effect on its own.

4.3 Channel Vocoder Effects

Finally, the implementation of channel vocoder effects is presented in Appendix A.4. It is remarked that the effect of *Denoising* was left out since a proper setup with decent results would have turned into a quite complex implementation, and as this thesis focuses rather on *phase* vocoder effects, this seemed to be beyond the scope of this thesis. However, programming the other channel vocoder effects appeared to be simple, as shown in Appendix A.4.1, A.4.2, A.4.3 and A.4.4.

4.4 Additional Utilities

In Appendix A.5, additional functions that were written during the realization of the phase vocoder are listed. In order to decimate redundant parts of the code, some parts of the algorithms had to be outsourced. This involved, most importantly, the detection of regions of influence, provided by the procedure `getRegions()` (file `getRegions.m`). The code of this function is listed in Appendix A.5.1.

The other functions are only of small size, but nevertheless important; so the source code of them is shown in Appendix A.5.2, A.5.3, A.5.4 A.5.5 and A.5.6.

One final note regards the fractional delay filter coefficients, gained via `lagrangeFIR3()` (see A.5.2). It was chosen to implement an interpolation filter of third order, for which the coefficients were set according to [LVKL96].

5 Evaluation and Conclusion

After implementing the phase vocoder, it may be of interest to ascertain its performance and draw conclusions from that – all of which is done in the current section.

Since a comprehensive, analytical investigation – besides the fact that no completely reliable quality measurement method exists – would be far beyond the scope of this thesis, the evaluation tests were informal and subjective and should only provide a basic feeling of "sweet spots" regarding different effect settings.

In most cases, audio effect devices are driven by music or speech signals, which can reveal quite different characteristics. Therefore, the tests were performed with both signal types, sometimes resulting in different optimal configuration settings.

5.1 Determination of Optimal Settings

In this section, the configuration parameters that achieved the best evaluation results are provided. A rather tabular than textual presentation may give a structural overview of the values.

5.1.1 Default settings

If it is not differently stated in the tables below, the default phase vocoder configuration values are set according to Table 1.

Parameter Name	Default Value
blending factor (`ALPHA`)	1.0
window size (`W_SIZE`)	2^{10}
window type (`W_TYPE`)	`@hanningz`
window extension (`W_EXTENSION`)	1
overlap (`OVERLAP`)	0.750
ratio (`RATIO`)	1.000
pitch shift flag (`PITCHSHIFT`)	`true`
effect function handle (`FX_HANDLE`)	`@FX_PASSTHRU`
phase update function handle (`PU_HANDLE`)	`@PU_PASSTHRU`

Table 1: Default values of the phase vocoder.

5.1.2 Time Stretching / Pitch Shifting via Resampling

Several test stretching factors α were applied, residing between $0.5 \leq \alpha \leq 2.0$. Then, with inspection of the influences of different parameters, the listening tests on speech and music signals were performed. If for equal values the same quality was observerd, the one with better behaviour in terms of computational complexity was chosen.

Basic Phase Propagation

To apply this phase update algorithm, the function handle `PU_HANDLE` must be set to `@PU_PASSTHRU`. In Table 2 the results are listed.

It was perceived that the basic phase propagation algorithm – as expected – does not show sufficient performance on voice signals, not to mention on music signals. Nevertheless, with values of $\alpha \approx 1.00 \pm 0.05$ the result was at least acceptable. In combination with a blending factor (field `ALPHA`) of about 0.5 and pitch shifting activated (field `PITCHSHIFT = true`), a decent chorus effect can even be modeled (a window size of `W_SIZE` $= 2^{10}$ being necessary).

Parameter Name	Speech Signals	Music Signals
window size (`W_SIZE`)	2^{10}	2^{12}
overlap (`OVERLAP`)	0.750	0.750

Table 2: Optimum values for time stretching / pitch shifting via basic phase propagation.

Loose Phase-Locking

This phase update algorithm can be involved by setting `PU_HANDLE = @PU_LOOSEPL`. The evaluation results are presented in Table 3.

The observations revealed that this algorithm performs best on ambient music signals, but not so well on speech signals. Since loose phase-locking is the only algorithm that applies a general phase relation between all channels – regardless of peaks in the spectrum –, this might be the reason why in this particular case it overtops the rigid phase locking algorithms, which need clear and well defined peaks to work well.

Parameter Name	Speech Signals	Music Signals
window size (`W_SIZE`)	2^{10}	2^{12}
overlap (`OVERLAP`)	0.750	0.750

Table 3: Optimum values for time stretching / pitch shifting algorithm when loose phase-locking is applied.

Identity Phase-Locking

Parameter Name	Speech Signals	Music Signals
window size (`W_SIZE`)	2^{10}	2^{12}
overlap (`OVERLAP`)	0.750	0.875

Table 4: Optimum values for time stretching / pitch shifting algorithm when identity phase-locking is applied.

In order to activate this algorithm, the parameter PU_HANDLE = @PU_IDENTITYPL must be set. The results are given in Table 4.

This algorithm shows bad performance on complex music signals where no clear peaks can be detected. Far better performance was achieved on speech signals.

Scaled Phase-Locking

The scaled phase-locking algorithm can be applied to the phase vocoder synthesis stage by setting the function handle PU_HANDLE to @PU_SCALEDPL. Table 5 shows the results.

A clear improvement compared to identity phase-locking could not be perceived, but a small improvement of the signal clarity was observed. Similar to identity phase-locking, the performance on sophisticated music signals was absolutely poor, but despite of this, excellent on speech signals.

Parameter Name	Speech Signals	Music Signals
window size (W_SIZE)	2^{10}	2^{12}
overlap (OVERLAP)	0.750	0.750

Table 5: Optimum values for time stretching / pitch shifting algorithm when scaled phase-locking is applied.

5.1.3 Selective Pitch Shifting

Selective pitch shifting is activated by setting FX_HANDLE = @FX_PITCHSHIFT and all other parameters to their standards, except those of Table 6.

Unfortunately, this algorithm completely failed on music signals. On speech signals, there were clearly perceivable artifacts introduced.

Parameter Name	Speech Signals	Music Signals
window size (W_SIZE)	2^{12}	-
overlap (OVERLAP)	0.875	-

Table 6: Optimum values for selective pitch shifting.

5.1.4 Mutation between Sounds

This effect can be achieved by applying the values listed Table 7 to the configuration parameters and setting the effect function handle to FX_HANDLE = @FX_MORPH.

Here, one popular example – amongst countless others – is picked out. A speech signal is mutated with an ambient signal with strong harmonies but little temporal change, such as the sound of a synthesizer or similar. The result is a quite impressive audio effect in which the voice assimilates the spectral characteristics whereas remaining to be understandable.

A remarkable fact is that even if the magnitude is taken from the voice signal, the spectral characteristics of the underlying ambient signal are intensively perceivable. On the other hand, even if the phase values of the ambient signal are passed through, the voice signal loses nothing of its perspicuity.

Parameter Name	Best Value
input file 1 (INPUT_FILE1)	ambient signal
input file 2 (INPUT_FILE2)	voice signal
blending factor (ALPHA)	0.9
magnitude combination (MORPHTYPE_R)	'R2'
phase combination (MORPHTYPE_P)	'P1'
window size (W_SIZE)	2^9
overlap (OVERLAP)	0.750

Table 7: Optimum values for selective pitch shifting.

5.1.5 Dispersion

For this effect, the handle FX_HANDLE must be set to @FX_DISPERSION. If the values from Table 8 are inserted, the dispersion effect can be produced.

As a subjective interpretation of the resulting sound characteristics, a kind of reverberation on speech signals was heard. The typical effect of dispersion – incorporating different delay for different frequency bands – was best experienced on percussive signals.

Parameter Name	Best Value
window size (W_SIZE)	2^{11}
window extension (W_EXTENSION)	2
overlap (OVERLAP)	0.750
dispersion factor (DISPFACTOR)	2.000

Table 8: Optimum values for the dispersion effect.

5.1.6 Robotization

The application of the robotization effect is done by setting the effect function handle FX_HANDLE to @FX_ROBOT. In Table 9, the best configuration values are listed.

Parameter Name	Best Value
input file 1 (INPUT_FILE1)	voice signal
window size (W_SIZE)	2^{12}
overlap (OVERLAP)	0.875

Table 9: Optimum values for the robotization effect.

Since in this effect, the phase values are set to zero for all STFT frames, the resulting output signal spectrum is in a way *locked* to a frequency similar to the reciprocal of the window size. Hence, for small window sizes, this frequency increases, whereas for bigger window sizes it decreases.

5.1.7 Whisperization

If the field `FX_HANDLE` is set to `@FX_WHISPER` and the values from Table 10 are applied, the whispering effect can be produced, performing best on speech signals.

Parameter Name	Best Value
input file 1 (`INPUT_FILE1`)	voice signal
window size (`W_SIZE`)	2^9
overlap (`OVERLAP`)	0.750
whispering component (`WHISP_COMP`)	`'PHASE'`

Table 10: Optimum values for the robotization effect.

5.2 Conclusion

As it might have been expected, music signals turned out to be much more complicated to handle than ordinary speech signals. Of course, no comprehense test library was of disposal, so only a limited bandwith of characteristical sounds could be tested. Somehow, the bad performance on music signals (especially *phase* vocoder effects) was disappointing due to the fact that promising algorithms were implemented.

One special remark addresses the failure of selective peak shifting on audio signals and the average performance of identity and scaled phase-locking on music signals. As all three of them utilize a peak detection stage, this peak detection algorithm may be improved or adapted in future work.

The algorithm of reconstructing the phase values from magnitude mentioned in Section 3.1.2 could bring some advantages of the output quality of time stretched or pitch shifted audio signals as well.

A Appendix: MATLAB® Source Code

In this appendix, the full source code of the *MATLAB®* implementation is provided.

A.1 The Basic Framework

A.1.1 Main Script

DAFX.m

```
% ----------------------------------------------------------------------
% DAFX MAIN SCRIPT
% The phase vocoder is configured and executed, assumed to be integra-
% ted into the framework properly.
% ----------------------------------------------------------------------
% Bachelor Thesis Telematics                Graz University of Technology
% Johannes Gruenwald               johannes.gruenwald@student.tugraz.at
%                                                              June2010
% ----------------------------------------------------------------------


% ----------------------------------------------------------------------
% CLEANING UP
% ----------------------------------------------------------------------

clear all;
close all;
clc;


% ----------------------------------------------------------------------
% DEFINITION OF INPUT FILES
% ----------------------------------------------------------------------

input_dir = 'sound/';
input_files = {[input_dir, 'sample1.wav'], ...
               [input_dir, 'sample2.wav'], ...
               [input_dir, 'sample3.wav']};

SAMPLE1 = 1;
SAMPLE2 = 2;
SAMPLE3 = 3;


% ----------------------------------------------------------------------
% CONFIGURATION: OBLIGATORY VALUES
% ----------------------------------------------------------------------

% Input file 1
pv_in.INPUT_FILE1 = input_files{SAMPLE1};

% Blending: 0 (dry) <= alpha <= 1 (wet)
pv_in.ALPHA      = 1.0;

% Window size
pv_in.W_SIZE     = 2^10;

% Window type function handle. Currently supported: @hanningz, @gaussz
pv_in.W_TYPE     = @hanningz;
```

```

% Zero-padding extension factor. Value 1 means no change of window size,
% value 2 means zero padding of W_SIZE samples etc.
pv_in.W_EXTENSION = 1;

% Window overlap: 0.000 (no overlap) < 1.000 (total overlap).
% Input hop size: Automatically calculated by W_SIZE and W_OVERLAP.
OVERLAP             = 0.75;
pv_in.HOP_IN        = pv_in.W_SIZE*(1-OVERLAP);    % No adjustments needed!

% Stretch factor and resulting output hop size
ratio               = 1.000;
pv_in.HOP_OUT       = round(pv_in.HOP_IN*ratio);   % No adjustments needed!

% Indicator if resampling should be performed in order to achieve pitch
% shifting (only relevant if HOP_IN ~= HOP_OUT)
pv_in.PITCHSHIFT    = true;

% Effects function handle. Currently supported effects are:
%    FX_PASSTHRU . . . . . . . . . . Passthrough
%    FX_DISPERSION . . . . . . . . . Dispersion
%    FX_ROBOT  . . . . . . . . . . . Robotization
%    FX_MORPH  . . . . . . . . . . . Mutation between sounds
%    FX_WHISPER  . . . . . . . . . . Whisperization
pv_in.FX_HANDLE     = @FX_PASSTHRU;

% Phase update algorithm. Currently supported algorithms are:
%    PU_PASSTHROUGH  . . . . . . . . No phase update is performed
%    PU_BASIC  . . . . . . . . . . . Basic phase propagation is applied
%    PU_LOOSEPL  . . . . . . . . . . Loose phase-locking is applied
%    PU_IDENTITYPL . . . . . . . . . Rigid phase-locking: Identity
%                                    phase-locking is applied
%    PU_SCALEDPL   . . . . . . . . . Rigid phase-locking: Scaled phase-
%                                    locking is applied
pv_in.PU_HANDLE     = @PU_PASSTHRU;


% ---------------------------------------------------------------------
% CONFIGURATION: OPTIONAL VALUES
% Depending on phase vocoder effect and phase update algorithm
% ---------------------------------------------------------------------

% Optional time limit in seconds for faster processing
%pv_in.LIMIT = 5;

% Pitch shifting factor (only for FX_PITCHSHIFT)
%pv_in.PSFACTOR = 1.000;

% Phase scaling factor beta (only for PU_SCALEDPL), ratio <= beta <= 1
% Note that as the hop sizes are quantized values, the correct ratio is
% recomputed by relating the hop sizes and not taking the (potentially
% wrong) value directly from <ratio>.
%pv_in.SPL_BETA = 2/3 + (pv_in.HOP_OUT/pv_in.HOP_IN)/3;

% Input file 2 (only for FX_MORPH)
%pv_in.INPUT_FILE2 = input_files{SAMPLE2};

% Magnitude combination (only for FX_MORPH)
%pv_in.MORPHTYPE_R = 'R1';
%pv_in.MORPHTYPE_R = 'R2';
%pv_in.MORPHTYPE_R = 'R1*R2';
```

```
%pv_in.MORPHTYPE_R = 'R1+R2';

% Phase combination (only for FX_MORPH)
%pv_in.MORPHTYPE_P = 'P1';
%pv_in.MORPHTYPE_P = 'P2';
%pv_in.MORPHTYPE_P = 'P1+P2';

% Dispersion factor (only for FX_DISPERSION)
%pv_in.DISPFACTOR = 1;

% Component that should be randomized (only for FX_WHISPER)
%pv_in.WHISP_COMP = 'MAG';
%pv_in.WHISP_COMP = 'PHASE';


% ----------------------------------------------------------------------
% DAFX EXECUTION
% ----------------------------------------------------------------------

% Execute phase vocoder
pv_out = PVOC(pv_in);

% Normalize and play output
pv_out.y = pv_out.y ./ max(abs(pv_out.y));
sound(pv_out.y, pv_out.fs);

% Define output file string
opt = [];
if (strcmp(func2str(pv_in.FX_HANDLE), 'FX_PITCHSHIFT'))
  opt = ['_factor', num2str(pv_in.PSFACTOR)];
end
if (strcmp(func2str(pv_in.FX_HANDLE), 'FX_MORPH'))
  opt = ['_mode', pv_in.MORPHTYPE_R, pv_in.MORPHTYPE_P];
end
if (strcmp(func2str(pv_in.PU_HANDLE), 'PU_SCALEDPL'))
  opt = [opt, '_beta', num2str(pv_in.SPL_BETA)];
end

% Write file to disk
wavwrite(pv_out.y, pv_out.fs,                               ...
  [pv_in.INPUT_FILE1(1:length(input_dir)), 'output/', ...
   pv_in.INPUT_FILE1(length(input_dir)+1:end-4),      ...
   '_ratio',   num2str(pv_in.HOP_IN/pv_in.HOP_OUT),   ...
   '_alpha',   num2str(pv_in.ALPHA),                  ...
   '_wsize',   num2str(pv_in.W_SIZE),                 ...
   '_wext',    num2str(pv_in.W_EXTENSION),            ...
   '_overlap', num2str(OVERLAP),                      ...
   '_lock',    func2str(pv_in.PU_HANDLE),             ...
   '_fx',      func2str(pv_in.FX_HANDLE),             ...
   opt,                                               ...
   '.wav']);

% Plot time-domain signal (optional)
%plot(pv_out.y, 'k');
%xlabel('Time (Discrete Samples)')
%ylabel('Amplitude');
%axis tight;
```

A.1.2 Phase Vocoder Basic Script

PVOC.m

```
function pv_out = PVOC(pv_in)
  % SYNTAX
  %   pv_out = PVOC(pv_in)
  %
  % DESCRIPTION
  %   Performs a digital audio effect within the phase vocoder. The
  %   device is fully configured by parameters stored in the input
  %   struct <pv_in>.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                       data; necessary fields are
  %                                       listed below.
  %
  %   The input struct <pv_in> MUST contain the following fields:
  %
  %   pv_in.INPUT_FILE1 . . . . . . . . . Path to input file 1
  %   pv_in.ALPHA . . . . . . . . . . . . Effect blending:
  %                                       0 (dry) <= ALPHA <= 1 (wet)
  %   pv_in.W_SIZE  . . . . . . . . . . . Phase vocoder window size
  %   pv_in.W_TYPE  . . . . . . . . . . . Window type function handle.
  %                                       Currently supported by this
  %                                       framework: @hanningz, @gaussz
  %   pv_in.W_EXTENSION . . . . . . . . . Zero-padding extension factor.
  %                                       Value 1 means no change of
  %                                       window size, value 2 means zero
  %                                       padding of W_SIZE samples etc.
  %   pv_in.HOP_IN  . . . . . . . . . . . Input hop size
  %   pv_in.HOP_OUT . . . . . . . . . . . Output hop size
  %   pv_in.PITCHSHIFT  . . . . . . . . . Boolean indicator if resampling
  %                                       should be performed in order to
  %                                       achieve pitch shifting (only
  %                                       relevant if HOP_IN ~= HOP_OUT)
  %   pv_in.FX_HANDLE . . . . . . . . . . Effects function handle. Cur-
  %                                       rently supported effects are:
  %                                       FX_PASSTHRU .... Passthrough
  %                                       FX_DISPERSION .. Dispersion
  %                                       FX_ROBOT ...... Robotization
  %                                       FX_MORPH ...... Mutation
  %                                       FX_WHISPER ..... Whisperization
  %   pv_in.SC_HANDLE . . . . . . . . . . Phase update algorithm. Sup-
  %                                       ported algorithms are:
  %                                       PU_PASSTHROUGH . No phase up-
  %                                          date is performed
  %                                       PU_BASIC ...... Basic phase
  %                                          propagation is applied
  %                                       PU_LOOSEPL ..... Loose phase-
  %                                          locking is applied
  %                                       PU_IDENTITYPL .. Rigid phase-
  %                                          locking: Identity phase-
  %                                          locking is applied
  %                                       PU_SCALEDPL .... Rigid phase-
  %                                          locking: Scaled phase-
  %                                          locking is applied
  %
  %   The input struct <pv_in> MAY contain the following fields (depen-
  %   ding on phase vocoder effect and phase update algorithm):
```

```
%
%   pv_in.LIMIT . . . . . . . . . . . . Optional time limit for faster
%                                       processing
%   pv_in.INPUT_FILE2 . . . . . . . . . Input file 2 (only for effect
%                                       FX_MORPH)
%
%   Additional fields may be required too in respect of a chosen audio
%   effect. It is referred to the actual implementation of this effect
%   to see which parameters are necessary; the values here presented
%   are only for the phase vocoder in its basic configuration.
%
% RETURN VALUES
%   The struct <pv_out> contains the following output values of the
%   phase vocoder:
%
%   pv_out.y  . . . . . . . . . . . . . Time-domain output signal
%   pv_out.fs . . . . . . . . . . . . . Sample rate of audio file for
%                                       optional playback
%
% ---------------------------------------------------------------------
% Bachelor Thesis Telematics              Graz University of Technology
% Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
%                                                             June 2010
% ---------------------------------------------------------------------


% ---------------------------------------------------------------------
% Read input
try
  % Required values
  INPUT_FILE1 = pv_in.INPUT_FILE1;
  ALPHA       = pv_in.ALPHA;
  W_SIZE      = pv_in.W_SIZE;
  W_TYPE      = pv_in.W_TYPE;
  W_EXTENSION = pv_in.W_EXTENSION;
  HOP_IN      = pv_in.HOP_IN;
  HOP_OUT     = pv_in.HOP_OUT;
  PITCHSHIFT  = pv_in.PITCHSHIFT;
  FX_HANDLE   = pv_in.FX_HANDLE;
  PU_HANDLE   = pv_in.PU_HANDLE;

  % Optional values
  if isfield(pv_in, 'LIMIT')
    LIMIT = pv_in.LIMIT;
  else
    LIMIT = inf;
  end
  if strcmp(func2str(FX_HANDLE), 'FX_MORPH')
    INPUT_FILE2 = pv_in.INPUT_FILE2;
  end
  % Note: The other values need not to be extracted since they
  %       are used within other functions.
catch ME
  error(['Input could not be read properly (', ME.message, ')']);
end

% ---------------------------------------------------------------------
% Load input data and extend it properly

[x1, fs1] = wavread(INPUT_FILE1);
x1        = [x1(:, 1); ...
```

```
            zeros(HOP_IN - mod(length(x1), HOP_IN), 1)] ...
                                                      ./ max(abs(x1(:, 1)));
X1_LENGTH = length(x1);

if exist('INPUT_FILE2', 'var')
  [x2, fs2] = wavread(INPUT_FILE2);
  x2        = [x2(:, 1); ...
               zeros(HOP_IN - mod(length(x2), HOP_IN), 1)] ...
                                                     ./ max(abs(x2(:, 1)));
  X2_LENGTH = length(x2);
else
  X2_LENGTH = inf;
end

% ----------------------------------------------------------------------
% Initializations

% Compute time stretch ratio as a fraction of the hop sizes.
% Additionally, set up the factor <pratio> which indicates that
% for pitch shifting, the temporal evolution will remain the same
% after all. This factor is used when computing the length of se-
% quences.
pv_in.ratio = HOP_OUT/HOP_IN;
if (pv_in.PITCHSHIFT)
  pratio = 1;
  tratio = pv_in.ratio;
else
  tratio = 1;
  pratio = pv_in.ratio;
end

% Define input start indices and number of input blocks
sample_end = min(min(X1_LENGTH, X2_LENGTH), ...
                     LIMIT*fs1/pratio)-W_SIZE;
in_start   = 0:HOP_IN:sample_end-1;
IN_BLOCKS  = length(in_start);

% Define window
W      = W_TYPE(W_SIZE);                        % Compute window
W      = W(:);                                  % Ensure column vector
W      = [W; zeros(W_SIZE*(W_EXTENSION-1), 1)]; % Extend it properly
W      = repmat(W, 1, IN_BLOCKS);               % Repeat for all blocks
W_SIZE = size(W, 1);
W_n    = 1:W_SIZE;
[x, y] = meshgrid(in_start, W_n);

% Define input slice matrix of size [W_SIZE x IN_BLOCKS]
%
% Note that based on this definition, the data passed to through the
% phase vocoder has the following structure (examplary input hop size
% of 16):
%
%            STFT frame #  --->
%            ---------------------
%       n/k  |00   16   32  . . .
%            |01   17   33  . . .
%        |   |02   18   34  . . .
%        |   |03   19   35  . . .
%        |   |04   20   36  . . .
%        v   |..   ..   ..  .
%            |..   ..   ..    .
```

```
%             |..    ..    ..       .
%             |
%
in_slice = x+y;
clear x;
clear y;

% Generate empty output signal of proper length
pv_out.y = zeros(W_SIZE + ceil(min(min(X1_LENGTH,               ...
                               X2_LENGTH)*pratio*max(tratio,1), ...
                                   LIMIT*fs1*max(tratio,1))), 1);
% Nominal frequency
k = (0:W_SIZE-1)';
pv_in.omega = k*2*pi/W_SIZE;

% Start temporal performance measurement of chosen algorithm
tic;

% ---------------------------------------------------------------------
% ANALYSIS STAGE
% Load windowed frames, circularly shift them and transform them into
% the frequency domain.
% ---------------------------------------------------------------------

% Load input frames and window them
x1_block = x1(in_slice).*W;

% Perform circular phase shift via the command <fftshift> except
% for loose phase locking as phase update algorithm
if ~strcmp(func2str(PU_HANDLE), 'PU_LOOSEPL')
  x1_block = fftshift(x1_block, 1);
end

% Execute FFT
fft_in.fft1 = fft(x1_block);

% If a second input file is necessary, load and transform it
% analogously to the sequence above
if exist('INPUT_FILE2', 'var')
  x2_block = x2(in_slice).*W;
  if ~strcmp(func2str(PU_HANDLE), 'PU_LOOSEPL')
    x2_block = fftshift(x2_block, 1);
  end
  fft_in.fft2 = fft(x2_block);
end

% Save memory; in_slice is quite a big matrix, so free it
clear in_slice;

% ---------------------------------------------------------------------
% PROCESSING STAGE
% Perform audio effect defined by FX_HANDLE.
% ---------------------------------------------------------------------

% Perform channel vocoder algorithm
[fft_out] = FX_HANDLE(pv_in, fft_in);

% ---------------------------------------------------------------------
% SYNTHESIS STAGE
% Apply optional phase update algorithm and transform frames back into
% the time domain and overlapp-add them to yield the final result.
```

```
% ------------------------------------------------------------------------

% Apply scaling algorithm
[y_fft] = PU_HANDLE(pv_in, fft_out);

% Perform IFFT
y_block = real(ifft(y_fft));

% Circularly shift the blocks if necessary (cf. analysis stage)
if ~strcmp(func2str(PU_HANDLE), 'PU_LOOSEPL')
  y_block  = fftshift(y_block, 1);
  x1_block = fftshift(x1_block, 1);    % If pv_in.ALPHA < 1, x1_block
                                       % contributes to the output
                                       % signal as well; thus it has to
                                       % be shifted back.
end

% Window output frames
y_block = y_block.*W;

% Define indices of output blocks
sample_end = min(min(X1_LENGTH*pv_in.ratio, ...
                     X2_LENGTH), LIMIT*fs1*tratio)-W_SIZE;
out_start  = 1:HOP_OUT:sample_end;
OUT_BLOCKS = min(length(out_start), IN_BLOCKS);

% Blend y_block (by ALPHA) to the output signal pv_out.y
for i=1:OUT_BLOCKS
  out_slice = (out_start(i):out_start(i)+W_SIZE-1).';
  pv_out.y(out_slice) = pv_out.y(out_slice) + y_block(:,i)*ALPHA;
end

% ------------------------------------------------------------------------
% RESAMPLING (optional)
% If time scaling should be converted into pitch shifting, the output
% signal needs to be interpolated (at the moment linearly)
% ------------------------------------------------------------------------
if (pv_in.ratio ~= 1 && PITCHSHIFT)

  % Length of the interpolated signal
  l = min(length(pv_out.y)/pv_in.ratio, ...
          length(pv_out.y));

  % Compute indices of neighbors of the interpolated samples
  n      = 0:l-1;
  yfloor = floor(pv_in.ratio*n);
  yceil  = yfloor+1;

  % Compute interpolation factor, which is the distance of the current
  % point to its lower (left) neighbor
  g      = n*pv_in.ratio-yfloor;

  % Increment indices, since MATLAB starts indexing at value 1
  yfloor = yfloor+1;
  yceil  = yceil+1;

  % Assemble interpolated signal
  pv_out.y = pv_out.y(yfloor).*(1-g).' + pv_out.y(yceil).*g.';

end

```

```
  % ------------------------------------------------------------------
  % ADDING DRY COMPONENTS
  % After optional resampling, the original signal can be added in res-
  % pect to ALPHA.
  % ------------------------------------------------------------------
  for i=1:OUT_BLOCKS*min(pv_in.ratio, 1)
    out_slice = (in_start(i):in_start(i)+W_SIZE-1).' + 1;
    pv_out.y(out_slice) = pv_out.y(out_slice) + x1_block(:, i)*(1-ALPHA);
  end

  % Evaluate temporal performance of chosen algorithm
  toc

  % Write sample rate to output
  pv_out.fs    = fs1;

end
```

A.2 Time Stretching

A.2.1 Basic Phase Propagation

PU_BASIC.m

```
function fft_out = PU_BASIC(pv_in, fft_in)
  % SYNTAX
  %   fft_out = PU_BASIC(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies the basic phase propagation algorithm to time-frequency
  %   representation of a signal.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                        data; can be empty but needs to
  %                                        be listed as the function hand-
  %                                        les need to be interchangeably.
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                        of input signal, given in F#
  %                                        STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Processed STFT-frames; matrix
  %                                        with same dimensions as input
  %                                        parameter <fft_in>.
  %
  % ------------------------------------------------------------------
  % Bachelor Thesis Telematics               Graz University of Technology
  % Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
  %                                                             June 2010
  % ------------------------------------------------------------------

  % Transform STFT frames to polar coordinates
  r     = abs(fft_in);
  phi_a = angle(fft_in);

  % Set initial phase properly
  phi_s      = zeros(size(fft_in));
  phi_s(:,1) = pv_in.ratio*phi_a(:,1);

  % Perform phase propagation for all STFT frames
```

```
  for i=2:size(r, 2)
    delta_phi_a   = princarg(phi_a(:,i)                   ...
                           - pv_in.HOP_IN*pv_in.omega ...
                           - phi_a(:,i-1));
    delta_omega_a = delta_phi_a / pv_in.HOP_IN;
    omega_a       = pv_in.omega + delta_omega_a;
    phi_s(:,i)  = phi_s(:,i-1) + pv_in.HOP_OUT*omega_a;
  end

  % Transform result to Cartesian coordinates
  fft_out = r.*exp(1i*phi_s);

end
```

A.2.2 Loose Phase-Locking

PU_LOOSEPL.m

```
function fft_out = PU_LOOSEPL(pv_in, fft_in)
  % SYNTAX
  %   fft_out = PU_LOOSEPL(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies the basic loose phase-locking algorithm to time-frequency
  %   representation of a signal.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                       data; can be empty but needs to
  %                                       be listed as the function hand-
  %                                       les need to be interchangeably.
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                       of input signal, given in F#
  %                                       STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Processed STFT-frames; matrix
  %                                       with same dimensions as input
  %                                       parameter <fft_in>.
  %
  % ------------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald               johannes.gruenwald@student.tugraz.at
  %                                                               June 2010
  % ------------------------------------------------------------------------

  % Perform basic phase propagation
  fft_tmp = PU_BASIC(pv_in, fft_in);


  % Apply loose phase-locking
  num_ffts = size(fft_tmp, 2);
  phi_s    = angle(- [zeros(1, num_ffts); fft_tmp(1:end-1,:)] ...
                   + fft_tmp                                 ...
                   - [fft_tmp(2:end,:); zeros(1, num_ffts)]);

  fft_out = abs(fft_tmp).*exp(1i*phi_s);

end
```

A.2.3 Rigid Phase-Locking: Identity Phase-Locking

PU_IDENTITYPL.m

```
function fft_out = PU_IDENTITYPL(pv_in, fft_in)
  % SYNTAX
  %   fft_out = PU_IDENTITYPL(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies the identity phase-locking algorithm to time-frequency
  %   representation of a signal.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                        data; can be empty but needs to
  %                                        be listed as the function hand-
  %                                        les need to be interchangeably.
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                        of input signal, given in F#
  %                                        STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Processed STFT-frames; matrix
  %                                        with same dimensions as input
  %                                        parameter <fft_in>.
  %
  % -------------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald                johannes.gruenwald@student.tugraz.at
  %                                                               June 2010
  % -------------------------------------------------------------------------

  % Decimate FFT values to relevant parts (i.e. FFT-samples from N/2..N-1
  % are redundant as they are the complex conjugate of the (reversed)
  % samples N/2..1; assuming indexing from 0..N-1), hence keeping values
  % from 1..N/2+1 (cf. MATLAB addression scheme)
  fft = fft_in(1:(end/2+1),:);

  % Transform STFTs into polar coordinates
  r     = abs(fft);
  phi_a = angle(fft);

  % Initialize output phase properly
  phi_s      = zeros(size(fft));
  phi_s(:,1) = pv_in.ratio*phi_a(:,1);

  % Detect regions of influence (log domain)
  [regions_cell pks_cell pks_idx_cell] = getRegions(log(r));

  % Iterate all STFT frames
  for j=2:size(r, 2)

    % Extract peaks of current STFT
    pks_idx = pks_idx_cell{j};

    % Iterate all regions of influences
    for i=1:length(pks_idx)

      % Extract current region from cell container
      k  = regions_cell{j}{i};

```

```
      % Get current peak index
      k_p      = pks_idx(i);

      % Compute regular phase propagation for the peak
      delta_phi_a_pk   = princarg(phi_a(k_p,j)           ...
                        - pv_in.HOP_IN*pv_in.omega(k_p) ...
                        - phi_a(k_p,j-1));
      delta_omega_a_pk = delta_phi_a_pk / pv_in.HOP_IN;
      omega_a_pk       = pv_in.omega(k_p) + delta_omega_a_pk;
      phi_s_pk         = phi_s(k_p,j-1) + pv_in.HOP_OUT*omega_a_pk;

      % Receive theta as this phase difference
      theta = phi_s_pk - phi_a(k_p,j);

      % Rotate region of influence by theta
      theta_vec = repmat(theta, length(k), 1);
      Z         = exp(1i*theta_vec);
      fft(k,j)  = fft(k,j).*Z;

      % Save phi_s for next round
      phi_s(k,j) = princarg(angle(fft(k,j)));

    end
  end

  % Complete the spectrum
  fft_out = [fft(1:(end-1),:);
             conj(fft(end:-1:2,:))];

end
```

A.2.4 Rigid Phase-Locking: Scaled Phase-Locking

PU_SCALEDPL.m

```
function fft_out = PU_SCALEDPL(pv_in, fft_in)
  % SYNTAX
  %   fft_out = PU_SCALEDPL(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies the scaled phasel-locking algorithm on a time-frequency
  %   representation of a signal.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . Container of configuration
  %                                     data; necessary fields are
  %                                     listed below.
  %   pv_in.SPL_BETA  . . . . . . . . . Phase scaling factor
  %   fft_in [FL x F#]  . . . . . . . . Time-frequency representation
  %                                     of input signal, given in F#
  %                                     STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . Processed STFT-frames; matrix
  %                                     with same dimensions as input
  %                                     parameter <fft_in>.
  %
  % ------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald               johannes.gruenwald@student.tugraz.at
  %                                                             June 2010
```

```
% -----------------------------------------------------------------

% Read input
try
  SPL_BETA = pv_in.SPL_BETA;
catch ME
  error(['Input could not be read properly (', ME.message, ')']);
end

% Decimate FFT values to relevant parts (i.e. FFT-samples from N/2..N-1
% are redundant as they are the complex conjugate of the (reversed)
% samples N/2..1; assuming indexing from 0..N-1), hence keeping values
% from 1..N/2+1 (cf. MATLAB addression scheme)
fft = fft_in(1:(end/2+1),:);

% Transform to polar coordinates
r     = abs(fft);
phi_a = angle(fft);

% Initialize output phase
phi_s      = zeros(size(fft));
phi_s(:,1) = pv_in.ratio*phi_a(:,1);

% Detect regions of influence (log domain)
[regions_cell pks_cell pks_idx_cell] = getRegions(log(r));

% Empty initializiation of former peaks/indices
pks_idx0 = 0;
pks_val0 = 0;

% Iterate all STFT frames
for j=2:size(r, 2)

  % Extract peaks and indices of current frame
  pks_val = pks_cell{j}';
  pks_idx = pks_idx_cell{j}';

  % Iterate all regions of influences
  for i=1:length(pks_idx)

    % Extract region of influence k and associated peak k_p
    k   = regions_cell{j}{i};
    k_p = pks_idx(i);

    % Get corresponding peak from preceding STFT.
    % Note: Peak magnitudes are not considered yet, yielding still
    %       decent results. An improvement, however, would be to take
    %       them into account.
    kd = abs(repmat(k_p,length(pks_idx0),1)-pks_idx0); % Index distance
    [m k0] = min(kd);                                   % Min. distance
    k_f   = pks_idx0(k0);                               % Actual index

    % If peak is not in region of influence, assume the same index as
    % predecessor
    if isempty(intersect(k_f, k))
      k_f = k_p;
    end

    % Compute regular phase propagation for the peak
    delta_phi_a_pk   = princarg(phi_a(k_p,j)            ...
                         - pv_in.HOP_IN*pv_in.omega(k_p) ...
```

```
                        - phi_a(k_f,j-1));
      delta_omega_a_pk = delta_phi_a_pk / pv_in.HOP_IN;
      omega_a_pk       = pv_in.omega(k_p) + delta_omega_a_pk;
      phi_s_pk         = phi_s(k_f,j-1) + pv_in.HOP_OUT*omega_a_pk;
      phi_s(k,j)       = repmat(phi_s_pk, length(k), 1) + ...
                         SPL_BETA * ...
                         (phi_a(k,j) - repmat(phi_a(k_p,j),length(k),1));

    end

    % Save peak indices and values (not yet used) for next round
    pks_idx0 = pks_idx;
    pks_val0 = pks_val;

  end

  % Transform STFTs back to Cartesian coordinates
  fft = r.*exp(1i*phi_s);

  % Complete the spectrum
  fft_out = [fft(1:(end-1),:);
             conj(fft(end:-1:2,:))];

end
```

A.2.5 Passthrough

PU_PASSTHRU.m

```
function fft_out = PU_PASSTHRU(pv_in, fft_in)
  % SYNTAX
  %   fft_out = PU_PASSTHRU(pv_in, fft_in)
  %
  % DESCRIPTION
  %   This function does not modifiy the input signal but passes it
  %   directly through to the output.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                       data; can be empty but needs to
  %                                       be listed as the function hand-
  %                                       les need to be interchangeably.
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                       of input signal, given in F#
  %                                       STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Passed-through STFT-frames;
  %                                       matrix with same dimensions as
  %                                       input parameter <fft_in>.
  %
  % ---------------------------------------------------------------------
  % Bachelor Thesis Telematics                 Graz University of Technology
  % Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
  %                                                              June 2010
  % ---------------------------------------------------------------------

  fft_out = fft_in;

end
```

A.3 Pitch Shifting

A.3.1 Selective Peak Shifting

FX_PITCHSHIFT.m

```
function fft_out = FX_PITCHSHIFT(pv_in, fft_in)
  % SYNTAX
  %   fft_out = fft_out = FX_PITCHSHIFT(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies a pitch shift algorithm on a time-frequency representation
  %   of a signal by decomposing the signal into so-called "regions of
  %   influences" and separately moving them in the spectrum.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                       data; necessary fields are
  %                                       listed below.
  %   pv_in.PSFACTOR  . . . . . . . . . . Pitch shifting factor.
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                       of input signal, given in F#
  %                                       STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Processed STFT-frames; matrix
  %                                       with same dimensions as input
  %                                       parameter <fft_in>.
  %
  % ---------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald               johannes.gruenwald@student.tugraz.at
  %                                                             June 2010
  % ---------------------------------------------------------------------

  % Read input
  try
    PSFACTOR = pv_in.PSFACTOR;
  catch ME
    error(['Input could not be read properly (', ME.message, ')']);
  end

  % Remove redundant parts of the spectrum (i.e. FFT-samples from
  % N/2...N-1 being the complex conjugate of the (reversed) samples
  % N/2..1; assuming indexing from 0..N-1), hence keeping values from
  % 1..N/2+1 (cf. MATLAB addression scheme)
  fft      = fft_in.fft1;
  fft      = fft(1:(end/2+1),:);
  fft_size = size(fft, 1);

  % Generate empty output FFT
  fft_out = zeros(size(fft));

  % Transform into polar coordinates
  r   = abs(fft);

  % --------------------------------------------------------------------
  % 01 PEAK DETECTION
  % 02 REGION OF INFLUENCE ESTIMATION
  % --------------------------------------------------------------------

```

```
  [regions_cell pks_cell pks_idx_cell] = getRegions(log(r));

  % Iterate all STFT frames
  for j=1:size(r, 2)

    % Get peak indices and values
    pks_idx = pks_idx_cell{j};
    pks     = r(pks_idx,j);

    col = hsv(length(pks))*0.4;

    % Iterate all detected regions
    for i=1:length(regions_cell{j})

      % Assign current region index sequence and get the associated peak
      region = regions_cell{j}{i};
      pk_idx = pks_idx(i);

      % ----------------------------------------------------------------
      % 03 FREQUENCY ESTIMATION
      %    Quadratic interpolation in log domain to estimate true fre-
      %    quency (which is the exact determination if the signal was
      %    windowed by a gauss function.
      % ----------------------------------------------------------------

      % Define peak neighborhood
      pk_surr_idx = [pk_idx-1 pk_idx pk_idx+1];

      % Be sure that the peak's neighborhood is inside the spectrum
      while (pk_surr_idx(1) < 1)
        pk_surr_idx = pk_surr_idx + 1;
      end
      while (pk_surr_idx(3) > fft_size)
        pk_surr_idx = pk_surr_idx - 1;
      end

      % Get values from neighborhood samples
      pk_surr_val = r(pk_surr_idx,j)';

      % Perform quadratic interpolation
      tmp = [pk_surr_idx; ...
             pk_surr_val];
      [a b c m_x m_y] = parfit(tmp(:,1), tmp(:,2), tmp(:,3));

      % If determined maximum lies outside the spectrum, move it back
      % in (this should never have relevant consequences but is necessary
      % to guarantee faultless execution)
      if (m_x < 1)
        m_x = 1;
        m_y = r(m_x, j);
      end
      if (m_y > fft_size)
        m_x = fft_size;
        m_y = r(m_x, j);
      end

      % Compute w_0 and the necessary shift in respect of bins and
      % frequency
      w_0        = (m_x-1)*2*pi/fft_size;    % MATLAB indices start at 1

      % ----------------------------------------------------------------
```

```
    % 04 COMPUTATION OF FREQUENCY SHIFT
    % ---------------------------------------------------------------

    delta_w    = w_0*(PSFACTOR - 1);
    delta_bins = delta_w/(2*pi)*fft_size;

    % ---------------------------------------------------------------
    % 05 PERFORMING FREQUENCY SHIFT
    %    The necessary peak shift is performed in two steps; namely
    %    an integer and fractional shift.
    % ---------------------------------------------------------------

    % Decompose deta_bins into integer and fractional part
    delta_bins_int  = floor(delta_bins);
    delta_bins_frac = delta_bins - delta_bins_int; % Note that the
                                                   % fractional part
                                                   % is always posi-
                                                   % tive (i.e. refer-
                                                   % ring to a shift
                                                   % to the right),
                                                   % even for negative
                                                   % values of delta_w

    % 05.1 INTEGER SHIFT

    % Define samples to be shifted
    region_intshift = region;
    region_intshift(region_intshift + delta_bins_int > size(r, 1)) ...
                                                               = [];
    region_intshift(region_intshift + delta_bins_int < 1) = [];

    % Perform integer shift
    fft_region = zeros(size(r,1), 1);
    fft_region(region_intshift + delta_bins_int) = ...
                                          fft(region_intshift, j);

    % 05.2 FRACTIONAL SHIFT

    % Compute Lagrange interpolation FIR filter, order 3.
    h = lagrangeFIR3(delta_bins_frac);

    % Perform fractional shift via convolution
    fft_region = conv(fft_region, h);
    fft_region = fft_region(1:size(r, 1));

    % ---------------------------------------------------------------
    % 06 PHASE ADJUSTMENT
    % ---------------------------------------------------------------

    % Compute theta and apply it to the whole region
    theta      = princarg(delta_w*(j - 1)*pv_in.HOP_IN);
    fft_region = fft_region.*exp(1i*theta);

    % Accumulate result
    fft_out(:,j) = fft_out(:,j) + fft_region;

  end

end

% Complete the spectrum
```

```
  fft_out = [fft_out(1:(end-1),:);
             conj(fft_out(end:-1:2,:))];

end
```

A.4 The Channel Vocoder

A.4.1 Mutation between Sounds

FX_MORPH.m

```
function fft_out = FX_MORPH(pv_in, fft_in)
  % SYNTAX
  %   fft_out = FX_MORPH(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies the effect of mutation of two sounds on a time-frequency
  %   representation of a signal, also referred to as "morphing".
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                       data; necessary fields are
  %                                       listed below.
  %   pv_in.MORPHTYPE_R . . . . . . . . . Magnitude combination. Current-
  %                                       ly supported values are:
  %                                       'R1' ..... Magnitude of input
  %                                          signal 1 is passed through
  %                                       'R2' ..... Magnitude of input
  %                                          signal 2 is passed through
  %                                       'R1*R2' .. Magnitudes are mul-
  %                                          tiplied, referring to a lo-
  %                                          AND operation
  %                                       'R1+R2' .. Magnitudes are added
  %                                          together, referring to a lo-
  %                                          gical OR operation
  %   pv_in.MORPHTYPE_P . . . . . . . . . Phase combination. Currently
  %                                       supported values are:
  %                                       'P1' ..... Phase values of in-
  %                                          put signal 1 are passed
  %                                          through
  %                                       'P2' ..... Phase values of in-
  %                                          put signal 2 are passed
  %                                          through
  %                                       'P1+P2' .. Phase values of both
  %                                          input signals are added
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                       of input signal, given in F#
  %                                       STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Processed STFT-frames; matrix
  %                                       with same dimensions as input
  %                                       parameter <fft_in>.
  %
  % -------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald               johannes.gruenwald@student.tugraz.at
  %                                                             June 2010
  % -------------------------------------------------------------------

  % Read input
```

```
  try
    MORPHTYPE_R = pv_in.MORPHTYPE_R;
    MORPHTYPE_P = pv_in.MORPHTYPE_P;
  catch ME
    error(['Input could not be read properly (', ME.message, ')']);
  end

  % Transform input signals to polar coordinates
  r1   = abs(fft_in.fft1);
  phi1 = angle(fft_in.fft1);
  r2   = abs(fft_in.fft2);
  phi2 = angle(fft_in.fft2);

  % Apply morphing effect, depending on configuration
  switch MORPHTYPE_R

    case 'R1'
      r = r1;
    case 'R2'
      r = r2;
    case 'R1*R2'
      r = r1.*r2;
    case 'R1+R2'
      r = r1+r2;
    otherwise
      error(['FX_MORPH was not properly configured ',              ...
             '(pv_in.MORPHTYPE_R = ', MORPHTYPE_R, ', which was ', ...
             'not recognized).']);

  end

  switch MORPHTYPE_P

    case 'P1'
      phi = phi1;
    case 'P2'
      phi = phi2;
    case 'P1+P2'
      phi = phi1+phi2;
    otherwise
      error(['FX_MORPH was not properly configured ',              ...
             '(pv_in.MORPHTYPE_P = ', MORPHTYPE_P, ', which was ', ...
             'not recognized).']);

  end

  % Transform output back to Cartesian coordinates
  fft_out = r.*exp(1i*phi);

end
```

A.4.2 Dispersion

FX_DISPERSION.m

```
function fft_out = FX_DISPERSION(pv_in, fft_in)
  % SYNTAX
  %   fft_out = FX_DISPERSION(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies the dispersion effect on a time-frequency representation
```

```
  %   of a signal, simulating different delay for separate frequency
  %   bands.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                       data; necessary fields are
  %                                       listed below.
  %   pv_in.DISPFACTOR  . . . . . . . . . Dispersion factor, which is
  %                                       multiplied with the quadratic
  %                                       phase offset
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                       of input signal, given in F#
  %                                       STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Processed STFT-frames; matrix
  %                                       with same dimensions as input
  %                                       parameter <fft_in>.
  %
  % ---------------------------------------------------------------------
  % Bachelor Thesis Telematics               Graz University of Technology
  % Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
  %                                                             June 2010
  % ---------------------------------------------------------------------

  % Read dispersion factor
  try
    a = pv_in.DISPFACTOR;
  catch ME
    error(['Input could not be read properly (', ME.message, ')']);
  end

  % Transform to polar coordinates
  r   = abs(fft_in.fft1);
  phi = angle(fft_in.fft1);

  % Set up quadratic phase term
  qph = (0:(size(r, 1)-1))'.^2;

  % Apply quadratic phase term
  phi = phi + a*repmat(qph, 1, size(r,2));

  % Transform back to Cartesian coordinates
  fft_out = r.*exp(1i*phi);

end
```

A.4.3 Robotization

FX_ROBOT.m

```
function fft_out = FX_ROBOT(pv_in, fft_in)
  % SYNTAX
  %   fft_out = FX_ROBOT(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies the robotization effect on a time-frequency representation
  %   of a signal by setting phase values to zero.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
```

```
  %                                       data; can be empty but needs to
  %                                       be listed as the function hand-
  %                                       les need to be interchangeably.
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                       of input signal, given in F#
  %                                       STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Processed STFT-frames; matrix
  %                                       with same dimensions as input
  %                                       parameter <fft_in>.
  %
  % -------------------------------------------------------------------
  % Bachelor Thesis Telematics               Graz University of Technology
  % Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
  %                                                            June 2010
  % -------------------------------------------------------------------

  % Transform FFT to polar coordinates
  r   = abs(fft_in.fft1);
  phi = angle(fft_in.fft1);

  % Set the phase to zero
  phi = 0*phi; % This operation preserves the dimension of phi

  % Transform output signal back to Cartesian coordinates
  fft_out = r.*exp(1i*phi);

end
```

A.4.4 Whisperization

FX_WHISPER.m

```
function fft_out = FX_WHISPER(pv_in, fft_in)
  % SYNTAX
  %   fft_out = FX_WHISPER(pv_in, fft_in)
  %
  % DESCRIPTION
  %   Applies the whisperization effect on a time-frequency representa-
  %   tion of a signal by setting either the magnitude or phase values
  %   randomly.
  %
  % PARAMETERS
  %   pv_in . . . . . . . . . . . . . . . Container of configuration
  %                                       data; necessary fields are
  %                                       listed below.
  %   pv_in.WHISP_COMP  . . . . . . . . . Component that should be rando-
  %                                       mized. Currently supported
  %                                       values are:
  %                                       'MAG' .... Magnitude is rando-
  %                                                  mized
  %                                       'PHASE' .. Phase is randomized
  %   fft_in [FL x F#]  . . . . . . . . . Time-frequency representation
  %                                       of input signal, given in F#
  %                                       STFT frames of length FL
  %
  % RETURN VALUES
  %   fft_out [FL x F#] . . . . . . . . . Processed STFT-frames; matrix
  %                                       with same dimensions as input
  %                                       parameter <fft_in>.
```

```
  %
  % ---------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
  %                                                             June 2010
  % ---------------------------------------------------------------------

  % Read input
  try
    WHISP_COMP = pv_in.WHISP_COMP;
  catch ME
    error(['Input could not be read properly (', ME.message, ')']);
  end

  % Set magnitude values randomly
  if strcmp(WHISP_COMP, 'MAG')
    r = rand(size(fft_in.fft1));
    phi = angle(fft_in.fft1);
  % Set phase values randomly
  elseif strcmp(WHISP_COMP, 'PHASE')
    r   = abs(fft_in.fft1);
    phi = (rand(size(r))-0.5)*2*pi;
  else
    error(['FX_WHISPER was not properly configured ',          ...
           '(pv_in.WHISP_COMP = ', WHISP_COMP, ', which was ', ...
           'not recognized).']);
  end

  % Transform result back to Cartesian coordinates
  fft_out = r.*exp(1i*phi);

end
```

A.5 Additional Functions

A.5.1 Detection of Regions of Influence `getRegions()`

getRegions.m

```
function [regions pks pks_idx] = getRegions(mag)
  % SYNTAX
  %   [regions pks pks_idx] = getRegions(mag)
  %
  % DESCRIPTION
  %   Detects regions of influence of magnitude spectra spectrum, retur-
  %   ning indices in cell arrays. The regions are divided by minima be-
  %   tween the peaks.
  %
  % PARAMETERS
  %   fft_in [FL x F#]  . . . . . . . . . Input spectra, given in F#
  %                                       STFT frames of length FL
  %
  % RETURN VALUES
  %   regions {F#}{R#}  . . . . . . . . . Indices of detected regions,
  %                                       where the R# detected regions
  %                                       per frame are given in F# cells
  %   pks_val {F#}{R#}  . . . . . . . . . Values of detected peaks,
  %                                       where the R# detected peaks
  %                                       per frame are given in F# cells
  %   pks_idx {F#}{R#}  . . . . . . . . . Indices of detected peaks,
  %                                       where the R# detected peaks
```

```
%                                        per frame are given in F# cells
%
% ----------------------------------------------------------------------
% Bachelor Thesis Telematics               Graz University of Technology
% Johannes Gruenwald             johannes.gruenwald@student.tugraz.at
%                                                             June 2010
% ----------------------------------------------------------------------

% Threshold for the execution of <findpeaks>
MINPK = 1;

% Iterate all STFT frames
for j=1:size(mag,2)

  % Initialize empty region and empty peak list
  regions{j} = {};
  pks{j} = [];
  pks_idx{j} = [];

  % Extract current spectrum
  curr_mag = mag(:,j);

  % If threshold is reached, perform MATLABs <findpeak> algorithm.
  % Note that there is a threshold set, neglecting peaks with too
  % little magnitude.
  if (max(curr_mag) > MINPK)
    [pks{j}, pks_idx{j}] = findpeaks(curr_mag, 'minpeakheight', MINPK);
  end
  % If threshold is not reached, assume whole frame as one region
  % and continue
  if isempty(pks{j})
    [pks{j} pks_idx{j}] = max(curr_mag);
    regions{j}{end+1} = 1:size(mag,1);
    continue;
  end

  % This is just a verification that all samples are attached to a
  % region of influence
  border_check = ones(length(curr_mag),1);

  % Begin at the leftmost sample
  border_left = 1;

  % Iterate all detected peaks
  for i=1:length(pks_idx{j})

    % Detect minimum between peaks and assign it to border_right
    if (i~=length(pks_idx{j}))
      left    = pks_idx{j}(i);
      right   = pks_idx{j}(i+1);
      slice   = left:right;
      [m,idx] = min(curr_mag(slice));
      idx     = idx+left-1;
    else
      idx     = size(mag,1)+1;  % Correction by 1 to get
                                % borders right (s.b.)
    end
    border_right = idx;

    % Define and assign region
    region           = border_left:border_right-1;
```

```
      regions{j}{end+1} = region;

      % Assign region as attached
      border_check(region) = border_check(region) - 1;

      % Current right border is next left border
      border_left = border_right;

    end

    % Verify that all samples have been assigned to regions
    if sum(abs(border_check)) ~= 0
      error('getRegions(): Not all samples were assigned to regions!');
    end

  end

end
```

A.5.2 Lagrange FIR Interpolation Filter of Order 3

LagrangeFIR3.m

```
function h = lagrangeFIR3(delay)
  % SYNTAX
  %   h = lagrangeFIR3(delay)
  %
  % DESCRIPTION
  %   Creates impulse response (FIR coefficients) of a Lagrange cubic
  %   interpolator of order 3.
  %
  % PARAMETERS
  %   delay . . . . . . . . . . . . . . Fractional delay value
  %
  % RETURN VALUES
  %   h [4 x 1] . . . . . . . . . . . . Filter coefficients
  %
  % ---------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
  %                                                              June 2010
  % ---------------------------------------------------------------------

  h = zeros(4,1);
  D = delay;

  h(1) =   -(D-1)*(D-2)*(D-3)/6;
  h(2) =  D*(D-2)*(D-3)/2;
  h(3) = -D*(D-1)*(D-3)/2;
  h(4) =  D*(D-1)*(D-2)/6;

end
```

A.5.3 Modified Gaussian Window

gaussz.m

```
function w = gaussz(n)
  % SYNTAX
  %   w = gaussz(n)
  %
```

```
  % DESCRIPTION
  %   This function returns a Gauss window with corrected periodicity n.
  %
  % PARAMETERS
  %   n . . . . . . . . . . . . . . . . . . Window size
  %
  % RETURN VALUES
  %   w [n x 1] . . . . . . . . . . . . . Gauss window
  %
  % -------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald             johannes.gruenwald@student.tugraz.at
  %                                                            June 2010
  % -------------------------------------------------------------------

  w = [0; gausswin(n-1)];

end
```

A.5.4 Modified Hanning Window

hanningz.m

```
function w = hanningz(n)
  % SYNTAX
  %   w = hanningz(n)
  %
  % DESCRIPTION
  %   Returns a Hanning window with corrected periodicity n.
  %
  % PARAMETERS
  %   n . . . . . . . . . . . . . . . . . . Window size
  %
  % RETURN VALUES
  %   w [n x 1] . . . . . . . . . . . . . Hanning window
  %
  % -------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald             johannes.gruenwald@student.tugraz.at
  %                                                            June 2010
  % -------------------------------------------------------------------

  w = [0; hanning(n-1)];

end
```

A.5.5 Principal Domain Wrapping

princarg.m

```
function arg_out = princarg(arg_in)
  % SYNTAX
  %   arg_out = princarg(arg_in)
  %
  % DESCRIPTION
  %   Maps the submitted angle <arg_in> into the principal +/-pi-domain.
  %
  % PARAMETERS
  %   arg_in  . . . . . . . . . . . . . . Input argument
  %
  % RETURN VALUES
```

```
  %   arg_out . . . . . . . . . . . . . . Output argument
  %
  % ------------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
  %                                                             June 2010
  % ------------------------------------------------------------------------

  arg_out = mod(arg_in + pi, 2*pi) - pi;

end
```

A.5.6 Quadratic Interpolation

parfit.m

```
function [a b c m_x m_y] = parfit(p1, p2, p3)
  % SYNTAX
  %   [a b c m_x m_y] = parfit(p1, p2, p3)
  %
  % DESCRIPTION
  %   Fits a parabola of the form y = ax^2 + bx + c into the submitted
  %   points <p1>, <p2> and <p3>. The coefficients <a>, <b> and <c>
  %   are returned as well as the x- and y-value of the minimum/maximum.
  %
  % PARAMETERS
  %   p1, p2, p3 [2 x 1]  . . . . . . . . Input points
  %
  % RETURN VALUES
  %   a, b, c . . . . . . . . . . . . . . Coefficients of parabolic
  %                                       equation y = ax^2 + bx + c
  %   m_x, m_y  . . . . . . . . . . . . . x- and y-value of the minimum/
  %                                       maximum
  %
  % ------------------------------------------------------------------------
  % Bachelor Thesis Telematics              Graz University of Technology
  % Johannes Gruenwald              johannes.gruenwald@student.tugraz.at
  %                                                             June 2010
  % ------------------------------------------------------------------------

  x = [p1(1); p2(1); p3(1)];
  y = [p1(2); p2(2); p3(2)];
  X = repmat(x,1,3).^repmat(2:-1:0,3,1);

  C = X^(-1) * y;
  a = C(1);
  b = C(2);
  c = C(3);

  m_x = -b/(2*a);
  m_y = C'*m_x.^(2:-1:0)';

end
```

References

[AR77] J.B. Allen and L.R. Rabiner. A unified approach to short-time fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558 – 1564, nov. 1977.

[BA70] T. Bially and W. Anderson. A digital channel vocoder. *Communication Technology, IEEE Transactions on*, 18(4):435 –442, august 1970.

[Bag78] D. Baggi. Implementation of a channel vocoder synthesizer using a fast, time-multiplexed digital filter. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '78.*, volume 3, pages 167 – 170, apr 1978.

[Cap94] O. Cappe. Elimination of the musical noise phenomenon with the ephraim and malah noise suppressor. *Speech and Audio Processing, IEEE Transactions on*, 2(2):345 –349, apr 1994.

[CF87] A. Crossman and F. Fallside. Multipulse-excited channel vocoder. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '87.*, volume 12, pages 1926 – 1929, apr 1987.

[Cro80] R. Crochiere. A weighted overlap-add method of short-time fourier analysis/synthesis. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(1):99 – 102, feb 1980.

[DGBA00] A. De Götzen, N. Bernardini, and D. Arfib. Traditional (?) implementations of a phase vocoder: the tricks of the trade. In *COST-G6 Conference on Digital Audio Effects (DAFx-00)*, volume 3, pages 37–44, december 2000.

[Dol86] Mark Dolson. The phase vocoder: a tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[Fel82] J. Feldman. A compact digital channel vocoder using commercial devices. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '82.*, volume 7, pages 1960 – 1963, may 1982.

[Fer99] A.J.S. Ferreira. An odd-dft based approach to time-scale expansion of audio signals. *Speech and Audio Processing, IEEE Transactions on*, 7(4):441 –453, jul 1999.

[GL84] D. Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 32(2):236 – 243, apr 1984.

[Gol80] B. Gold. Formant representation of parameters for a channel vocoder. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '80.*, volume 5, pages 128 – 130, apr 1980.

[GR67] B. Gold and C. Rader. The channel vocoder. *Audio and Electroacoustics, IEEE Transactions on*, 15(4):148 – 161, dec 1967.

[KJ09] Hon Keung Kwan and A. Jiang. Fir, allpass, and iir variable fractional delay digital filter design. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 56(9):2064 –2074, sept. 2009.

[LD97] J. Laroche and M. Dolson. Phase-vocoder: about this phasiness business. In *Applications of Signal Processing to Audio and Acoustics, 1997. 1997 IEEE ASSP Workshop on*, page 4 pp., 19-22, 1997.

[LD99a] J. Laroche and M. Dolson. Improved phase vocoder time-scale modification of audio. *Speech and Audio Processing, IEEE Transactions on*, 7(3):323 –332, may 1999.

[LD99b] J. Laroche and M. Dolson. New phase-vocoder techniques for pitch-shifting, harmonizing and other exotic effects. In *Applications of Signal Processing to Audio and Acoustics, 1999 IEEE Workshop on*, pages 91 –94, 1999.

[Loo97] T.S. Loos. Implementation of a real-time hy-2 channel vocoder algorithm. In *MILCOM 97 Proceedings*, volume 1, pages 525 –529 vol.1, 2-5 1997.

[LVKL96] T.I. Laakso, V. Valimaki, M. Karjalainen, and U.K. Laine. Splitting the unit delay [fir/all pass filters design]. *Signal Processing Magazine, IEEE*, 13(1):30 –60, jan 1996.

[Lyo96] Richard G. Lyons. *Understanding Digital Signal Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[OS09a] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*, pages 648–741. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[OS09b] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*, pages 822–835. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[OS09c] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*, page 685. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[OS09d] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*, pages 219–221. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[OS09e] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*, page 87. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[OS09f] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*, page 305. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[PB98] M.S. Puckette and J.C. Brown. Accuracy of frequency estimates using the phase vocoder. *Speech and Audio Processing, IEEE Transactions on*, 6(2):166 –176, mar 1998.

[Por76] M. Portnoff. Implementation of the digital phase vocoder using the fast fourier transform. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 24(3):243 – 248, jun 1976.

[Puc95] M. Puckette. Phase-locked vocoder. In *Applications of Signal Processing to Audio and Acoustics, 1995., IEEE ASSP Workshop on*, pages 222 –225, 15-18 1995.

[QDH95] T.F. Quatieri, R.B. Dunn, and T.E. Hanna. A subband approach to time-scale expansion of complex acoustic signals. *Speech and Audio Processing, IEEE Transactions on*, 3(6):515 –519, nov 1995.

[Vas06] Saeed V. Vaseghi. *Advanced Digital Signal Processing and Noise Reduction*. John Wiley & Sons, 2006.

[VL93] V. Valimaki and T.I. Laakso. Fractional delay digital filters. In *Circuits and Systems, 1993., ISCAS '93, 1993 IEEE International Symposium on*, pages 355 –359 vol.1, 3-6 1993.

[ZBW07] Xinglei Zhu, G. Beauregard, and L. Wyse. Real-time signal estimation from modified short-time fourier transform magnitude spectra. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(5):1645 –1653, july 2007.

[Zoe02] Udo Zoelzer, editor. *DAFX: Digital Audio Effects*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

Nomenclature

DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
DTFT	Discrete-Time Fourier Transform
FFT	Fast Fourier Transform
IDFT	Inverse Discrete Fourier Transform
IDTFT	Inverse Discrete-Time Fourier Transform
IFFT	Inverse Fast Fourier Transform
STFT	Short-Time Fourier Transform