# Harmonic Audio Object Processing in Time Domain

**Charalampos - Christos G. Stamatopoulos**

MASTER THESIS UPF / 2009
Master in Sound and Music Computing

Master thesis supervisor:
Jordi Bonada Sanjaume
Department of Information and Communication Technologies
Universitat Pompeu Fabra, Barcelona

# Abstract

Transformation of polyphonic audio in terms of pitch shifting and time stretching is often desirable and sometimes necessary in order to correct a mistake in the performance of the musician or in order to experiment with different chords or durations, that can be produced by transforming accordingly the given audio piece. Transparency in such transformations should be a key feature that would offer persuading results.

In this Thesis we implement a framework that carries out such processes by embedding different modules and techniques such as filter-banks, pitch shifting algorithms, and harmonic gain compensation blocks. We apply transformations between different chords on several instruments of different characteristics , and we measure the quality of the transformed sound. Furthermore we mention some inherent vulnerabilities of the system and we propose ways to overcome them.

# Acknowledgements

I would like to thank my supervisor Jordi Bonada Sanjaume for his guidance and support. I would like to thank also Xavier Serra and Emilia Gomez for the opportunity they gave me to contribute to the Music Technology Group. Many thanks also to my close friends Vassileios Pantazis and Panos Giotis for their support, suggestions, and the enlightening conversations we had over the topic of this thesis.

Finally i would like to express my unending gratitude to my beloved parents Giorgos and Giorgia who were always there supporting my choices and efforts all these years.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In several applications it is desirable to have the ability of transforming an audio signal in terms of pitch shifting and time stretching. Many tools have been designed and implemented for such purposes both in frequency and in time domain. Nowadays sequencers often embed these tools in order to offer the user the ability to manipulate the audio signal.

Unfortunately these features are offered usually only for monophonic signals, hence the transform of a unique note in a polyphonic signal is not feasible, as there is no mechanism that de-mixes the signal to its different notes. Existing technology in this field include applications like Melodyne from Celemony and SonicWorx from Prosoniq.

Celemony, Melodyne, offers DNA (Direct Note Access) technology. According to this the user has the ability to manipulate the notes that form the chord separately, pitch shift or time stretch them. The function of the underlying mechanisms that carry out the process is not known though.

Prosoniq SonicWorx uses a very precise time-frequency approach along with pattern recognition algorithms in order to separate successfully the signal of interest. According to the company's site ( `http://www.prosoniq.com/editing-products/`

sonicworx/) " *See beyond the signal sonicWORX Pro utilizes an entirely new high precision time-frequency transformation which delivers resolution down to millihertz accuracy displayed on a convenient, note based scale. Not only is this representation strikingly beautiful to look at, it also helps you make out important features such as harmonics and even signals covered in noise. What's more, our transform uses statistical properties of the signal to measure noisiness as a function of frequency: everything that's noise is colored blue, while ordered (harmonic) signals are tinted yellow. No more looking at blurry FFT spectrograms that leave you wondering if what you are editing is really there, or just a discontinuity in the graph.*"

Previous work and research has been done towards source separation in music signals. Different methods can be employed to carry out this process. One approach would be to represent the signal as a weighted sum of basis functions. The reconstruction is achieved by minimizing the error between the magnitude spectrogram between the observed signal and the model[8]. Another approach followed also in [8] performs a sinusoidal modeling of the signal and minimizes the error between the magnitude spectrogram between the observed signal and the model. Other methods for source separation would employ neural networks [18], geometric methods[19], methods based on information theory[20]. Linear Predictive Coding (LPC) can be also used to model the signal which can then be separated using deconvolution[21].

In this Thesis, since we suppose that pitch information is available we use a filter-bank approach in order to perform source separation. Each filter-bank is tuned to the fundamental frequency and the harmonics of each note.

More specifically the contribution of this thesis to this field is to :

- Provide with a framework that given the source and the target MIDI score, de-mixes a polyphonic harmonic audio object to it's notes, transforms it accordingly and finally remixes it to a target audio object. This means that we

can form any chord given the audio signal, the source and the target score.

- Provide analysis and review on the different stages that consist the framework.
- Evaluate the results obtained by performing both objective and informal (listening) tests.
- Highlight upcoming issues, like overlapping harmonics and partially inharmonic sounds, and propose solutions on these.

We chose to work in time domain, as there are several advantages over time - frequency domain, like the lack of the window (as the whole process can be implemented per sample) that in some cases would smooth the fast transients, less complexity in terms of cpu cycles, as we avoid the transition from time domain to time - frequency domain and vice-versa. A significant parameter is also the high frequency accuracy that can be obtained easily with the use of filters, without adding more "weight" to the cpu by using bigger size for the FFT transform to get larger detail in the frequency domain. Finally it would be desirable to compare the quality results obtained (both objective and subjective) to the relevant work of Stefan Huber, who implements the same framework in time frequency domain.

The targets we set were

- Transparent de-mixing, mixing by maintaining the same pitch in all notes.
- High quality of the transformed signal. Persuading results.
- Discard phasing problems that come up by using IIR filters.
- Tune and select carefully the components of the framework and propose different implementations that could be used as a module of a more "blind", or general system, that can transform audio without a source score, by initially estimating the notes that consist the polyphonic audio signal.

In the following chapters we will present the structure of the framework, the state of the art algorithms it embodies, and will explain the way each separate component works. In addition, since there are several stages within the framework that acquire extensive and deep research, we will also submit proposals for further work mostly regarding the de-mixing of the signal, and ways to deal with inharmonic sounds.

# Chapter 2

# System Overview

## 2.1 System Structure

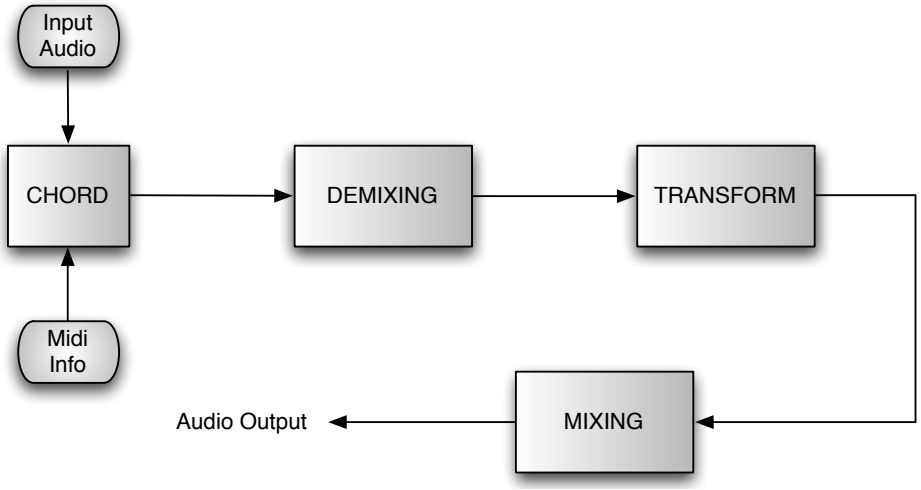



Figure 2.1: System Overview

In figure 2.1 we show the general system structure. The system consists of

- **The Chord Forming Block**. This block given the input midi information loads the separate instrument notes and forms the chord accordingly.
- **The Demixing Block**. This block is responsible for signal de-mixing to

separate notes. More specifically it demixes the signal into separate channels from which each one contains a note and all it's harmonics. In order to achieve this for each note we use a parallel bank of IIR filters tuned to the fundamental and it's harmonics Furthermore, assuming that the vibrato of the instrument is available as a MIDI data we need to adapt the function our system in order to take this information in account and tune the filters correctly over time.

Here we must note that in order to achieve a successful de-mixing we need to compensate the amplitudes of the overlapping harmonics. To achieve this we have to know or approximate the timbre of each instrument, and then apply an algorithm that fixes the filters' gains.

- **The Transformation Block**. In this block the pitch shifting transformation takes place. Having in our hands the source and the target midi score we decide which channel is about to get transformed. The pitch transformation can be done with Resampling and SOLA (Synchronous Overlap-Add), or PSOLA (Pitch Synchronous Overlap-Add).

- **The Mixing Block**. In this block we sum the transformed and the intact channels to form the output chord.

## 2.2 Component Review

### 2.2.1 Chord Formation, Obtaining MIDI features when not exist

The chord formation was done within the program for efficiency, because the separate notes were needed in order to perform evaluation tests both for the Transformation and the De-mixing - Re-mixing stage. The notes are represented by integer numbers starting from the lower to the higher note, depending the range of the

| | |
|---|---|
| C3 | 1 |
| C#3 | 2 |
| D3 | 3 |
| D#3 | 4 |
| ... | ... |

Table 2.1: Correspodence Between Note and Indexes

available ones. In the following table this correspondence is shown.

When the chord indexes are given, the audio files that correspond to each specific note in the table are loaded and added to form the chord which will be the input to the De-mixing stage. Before that, it is necessary to calculate the frequency in Hz for each note, to keep the filters updated with the correct coefficients. The formula used for this mapping is

$$Freq_{Hz} = 6.875 * 2^{(63+Note_{Idx})/12} \tag{2.1}$$

Where $Note_{Idx}$ is the distance in indexes from $C\#3$.

In addition it is necessary to feed the system with further MIDI information such as vibrato. If this information is not available, but vibrato is apparent we can use a pitch tracking algorithm to obtain it. For this purpose we used the Yin algorithm, which will be analyzed later in this document. By applying Yin to every note's audio file, we obtain the vibrato information, which can be stored and used at the next stage.

### 2.2.2 Demixing

De-mixing a polyphonic audio signal to its monophonic components can be carried out using different approaches, depending on the nature of the audio signal, the desirable cpu load, and the quality of the de-mixing. When there is no vibrato and

the notes are discrete, we can apply this process per note duration. Otherwise it is needed to work on a smaller time frame, in order to follow the nuances and the expression of the instrument.

Here we will present the three different methods we used to de-mix the audio signal.

**Processing per Note Duration** Initially it is necessary to feed the system with the target score, or better, the target note index. After that we apply an iterative process for each harmonic of each note.

More specifically for each note we set a maximum number of harmonics. Then we form a vector which contains the fundamental and all its harmonics up to the limit we have set. We restrict if necessary the length of this vector ($L$), to contain only harmonics up to $Fs/2$ where $Fs$ is the sampling rate. In the next step we create a filter bank which features narrow IIR bandpass filters tuned to the fundamental frequency of the note and it's harmonics, and we filter the audio signal with each one of these filters. When this process has finished we sum the outputs of all the filters. The result is a de-mixed monophonic channel that contains only a note. By applying this process for every note index we obtain a matrix that contains a de-mixed channel at each row.

A key issue here is the overlapping harmonics. In some cases we have total overlap, (for example the fundamental of $C4$ and the 1st harmonic of $C3$), and in other cases the harmonics fall to close to one another. The solutions given in these cases will be discused later in this document.

**Processing per Frame Duration** When vibrato is apparent the notes and the harmonics in the audio are not stable, but modulating around their center frequency. In order to follow the vibrato, one approach would be to perform the previous

process, not per note duration, but frame wise.

For each harmonic of each note we apply a frame by frame analysis in our signal. We apply windowing and we update the filter coefficients at each frame in respect to the center frequency and the modulation range of each harmonic. Thus the filters in the filter bank are transformed into time varying filters that follow also the vibrato information.

The approach followed here was to take the value of the vibrato vector at the middle index of the frame. We tried also to take the mean value of the vibrato within the frame, but it smoothed out the vibrato information a lot.

At the overlap - add stage it is necessary to multiply with a coefficient that depends from the window length and the hop size, to avoid high gains and clipping distortion at the de-mixed channels. This coefficient is given by

$$C = \frac{H}{M} \tag{2.2}$$

where $H$ is the hop size and $M$ is the window length.

**Processing per Arbitrary time frame** This approach filters the mixed signal per sample, and updates the filter coefficients when the vibrato varies over a given threshold, which can be calculated accordingly to the filter passband range, and the filter steepness ($dB$ per octave), in order to avoid attenuating the vibrating notes.

In the figure 2 we show the time instants that the vibrato information will be sampled according to a desirable upped modulation threshold.

Again the result is a time varying filter that adapts to the modulation frequency of the note. The filtering implementation for this approach should be done per sample, and not per vector using the matlab's filtering functions, to avoid destroying the signal due to filters' transition effects.

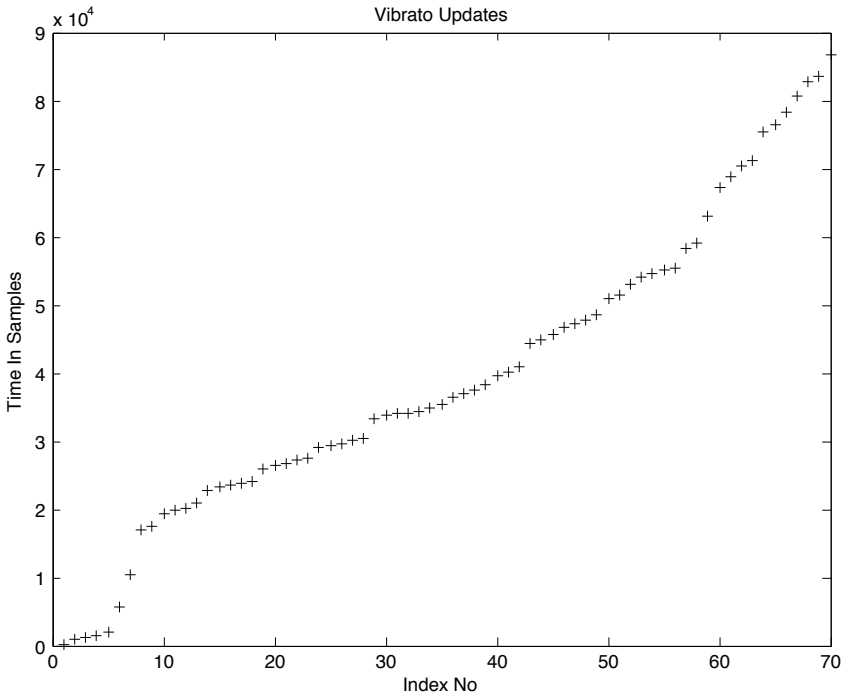



Figure 2.2: Vibrato Sampling

### 2.2.3 Transformation

This stage carries out the transformation of the audio object. Given the matrix with the de-mixed channels and by taking in account the target midi score, it applies pitch shifting to these target indexes that differ from the source ones. The result is a matrix that contains the transformed audio channels, along with the intact ones.

## 2.3 Overview and Analysis of the Algorithms Used

This system embeds some state of the art algorithms, like Yin, & PSOLA, in order to carry out the process of audio transformation. These algorithms lie within the different modules of the system. In this section we will make a review of these algorithms.

### 2.3.1 Yin

As mentioned before, when there is not any MIDI information about the vibrato of the input notes, it is necessary to estimate it in order to tune correct the filters over time. For this estimation we use the Yin [5] algorithm.

The Yin algorithm is based on the autocorrelation function of the signal, which is the first step for $F_0$ calculation, and it's goal is to decrease the error rate of this method.

By taking the autocorrelation function the result we obtain will be a set of peaks whose locations indicate a candidate for periodicity. The higher peak is chosen to measure the fundamental frequency $F_0$. This is not always correct, because in many signals the peak that corresponds to the true value of $F_0$ could be higher or lower than the higher power - chosen peak.

In the 2nd step the Yin algorithm models the signal as a periodic function with period $T$, by definition invariant for a time shift of T

$$x_t - x_{t+T} = 0. \tag{2.3}$$

By taking the square and averaging over a period we obtain

$$\sum_{j=t+1}^{t+W} (x_j - x_{j+T})^2 = 0; \tag{2.4}$$

An unknown period may be found by forming the difference function

$$d_i(\tau) = \sum_{j=1}^{W} (x_j - x_{j+\tau})^2 \tag{2.5}$$

and searching for the values of $\tau$ that make the function equal to zero.

The squared sum may be expanded in order to express the function in terms of the

ACF

$$d_t(\tau) = t_t(0) + r_{t+\tau}(0) - 2r_t(\tau) \tag{2.6}$$

The use of this difference function makes the ACF more immune to input amplitude changes, thus the error rate decreases significantly.

The difference function is zero at zero lag but close to it in period multiples, because of imperfect signal periodicity. Thus instead of a zero value we must set a lower decision limit, or else the algorithm will chose the zero lag time index and fail. The setting of a lower limit is not the best idea because a strong resonance at the first formant ($F_1$) might produce a set of secondary dips something that can confuse the process of choosing the right index. Instead of that it is possible to use the cumulative mean normalized difference function instead of just the difference function.

$$d_t^{'}(\tau) = \begin{cases} 1 & \text{if } \tau = 0 \\ d_i(\tau)/[(1/\tau)\sum_{j=1}^{\tau} d_t(j)] & \text{otherwise} \end{cases} \tag{2.7}$$

This function starts from 1 rather than 0, tends to remain large at low lags and drops below 1 only where $d(\tau)$ falls below average.

Sometimes one of the higher order dips are deeper than the period dip, something that can produce octave errors. The solution to this is to set an absolute threshold and choose the smallest value of $\tau$ that gives the minimum of $d'$ deeper than the threshold. If none is found the global minimum is used instead. This reduces the error rate down to 0.78%

If the $F_0$ is not a multiple of the sampling period then some estimation errors might occur. To solve this problem parabolic interpolation is used to extract the correct location of the minimum using it's neighbor values.

### 2.3.2 Pitch Shifting

Pitch shifting was implemented with two different methods (Resampling & SOLA, PSOLA), in order to chose the best out of them.

Following the first approach, in order to pitch shift the signal we examine the ratio $r$ between the source and the target frequency ($r = \frac{f_{in}}{f_{out}}$). The next step is to time stretch the signal by this ratio. The result will be a signal of the same pitch, but with length $r$ times longer or shorter depending on the values of input - output frequencies. Finally to obtain the pitch shifted signal which has the same duration as the original one we apply resampling by a factor of $\frac{1}{r}$.

The SOLA algorithm was first described by Rouscos and Wilgus [3] algorithm performs as follows. It splits the input vector in overlapping blocks of equal lengths, which are shifted according to a stretching factor a. Then it performs a calculation of the cross-corellation in the overlap area and finds it's maximum value and the discrete time interval in which occurs $\Delta t_n$. At this point, when maximum similarity occurs, the two successive blocks are multiplied by a fadeout and fadein function.

The PSOLA algorithm (proposed by Moulines et al.) [7] takes in account the pitch information of the signal in order to better synchronize the time segments to avoid pitch discontinuity. Since pitch information is available, through the MIDI information, we used this algorithm to perform pitch shifting, instead of Resampling & SOLA, as it has very good quality, it is cheaper in CPU cycles than SOLA used with some high quality Sinc Interpolation, and it preserves formants .

### 2.3.3 Filter Design and Zero Phase Filtering

The filters we used for the de-mixing stage were IIR bandpass filters. The use of IIR filters offers significantly less calculations than FIR filters, as for low orders we can

get a very steep frequency response. The filters were designed to be quite narrow and precise in frequency, so to filter out as much as possible frequencies that lie close to the one of our interest.

More specifically we used 6th order IIR bandpass filters, with 14 Hz pass band. These were the narrowest bandpass filters we could obtain using direct-1 form of implementation without introducing significant arithmetic error in our calculations. In order to use more steep filters we should augment their order, and implement a direct-2 form to avoid numerical errors.

We tested two different kinds of bandpass filters. The first one is the Butterworth filter. The second one is based on all-pass filters that follow a parallel topology. The filter design starts from the construction of a second-order, all-pass based, bandpass filter. The second order band-pass filter us described by the following transfer function

$$H(z) = \frac{1}{2}[1 + A(z)] \tag{2.8}$$

$$A(z) = \frac{-c + d(1-c)z^{-1} + z^{-2}}{1 + d(1-c)z^{-1} - cx^{-2}} \tag{2.9}$$

$$c = \frac{tan(\pi f_b/f_s) - 1}{tan(2\pi f_b/f_s) + 1} \tag{2.10}$$

$$d = -cos(2\pi f_c/f_s) \tag{2.11}$$

We can obtain higher order filters by cascading the 2 order BP filter described above.

Although IIR filters distort the phase of the signal, since the filtering process is applied offline, we can overcome this by using zero phase filtering. The cancellation

of the phase distortion effect makes it possible to use any of the aforementioned types of band pass IIR filters, since their magnitude response is almost identical.

### 2.3.4 Dealing with Overlapping Harmonics

In a chord it is quite common to have notes that their harmonics overlap. Up to this stage we have dealt with this issue when the harmonics overlap 100% (when they occur in the exact same frequency). Knowing the notes that the audio signal consists of, and by assuming that the signal is harmonic, it is easy to calculate the frequency indexes that harmonic overlap occurs. A necessary feature is the timbre of the instrument, which informs us about the spectral envelope of the signal. We explain the function of this algorithm with a simple example.

Let us imagine that the chord has two notes, $C\#3$ and $C\#4$. This first harmonic of $C\#3$ will overlap 100% with the fundamental of $C\#4$, and so on. Knowing the amplitude of each harmonic(by timbre information) it is easy to derive the filters' gain coefficients, in order to de-mix the signal correctly without artifacts in the reconstruction. So if the gain of the fundamental frequency is $g_0$ and this of the first harmonic $g_1$ the filter that filters the first harmonic of $C\#3$ will be multiplied by a coefficient given by

$$c_f = \frac{g_1}{g_0 + g_1}$$

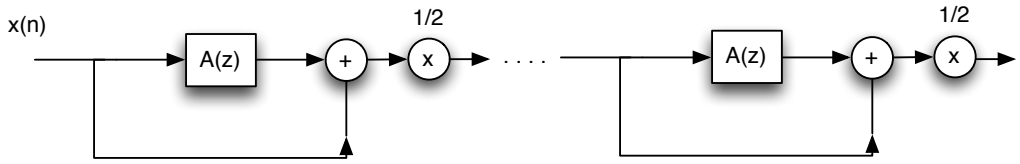



Figure 2.3: IIR filter derived by cascaded second order Band Pass filters.

.

Respectively the filter that filters the fundamental of $C\#4$ will be multiplied by

$$c_f = \frac{g_0}{g_0 + g_1}$$

.

When there is no harmonic overlap these coefficients are equal to 1.

In order to obtain correct values for the harmonics' amplitudes, a frequency analysis is being performed over the notes of the signal. More specifically we employ a peak detection algorithm that process the magnitude spectrum of the signal and detects the fundamental and all it's harmonics. This is quite trivial as the fundamental frequency of the signal is given. Then we form a vector that contains the magnitudes of these harmonics, and we normalize it in respect to the magnitude of the fundamental.

We used also a more blind approach which is based on the approximation of the spectral envelope. According to this we considered exponentially decayed spectral envelopes described by the following relation.

$$E[i] = \frac{1}{1 + a \cdot i} \tag{2.12}$$

where $i$ the number of the harmonic.

By changing the value of $a$ we can approximate spectral envelopes that decay faster or slower. This approach performs well in most cases, where the spectrum of the instrument decays exponentially like in Rhodes or Guitar. To achieve better approximation in instruments like violin and piano that do not follow strictly an exponential pattern, the use of a higher order spline would give significantly better results. The design of such curves is instrument targeted, or better, instrument

- family targeted, and is omitted here as it requires a lot of research, that would result to a detailed capture of all it's nuances, and includes significant work on the physical modeling area.

Here we analyze only one note, since we make the assumption that the ratio between the harmonics remains stable at different notes. In reality this ratio changes according to the note and also according to the vibrato. To model more accurately the ratio between harmonics an extensive, instrument - targeted, research should take place.

In the following figure(2.4) we plot the direct measuring and the approximation of the spectral envelope of a Rhodes, for 25 harmonics.

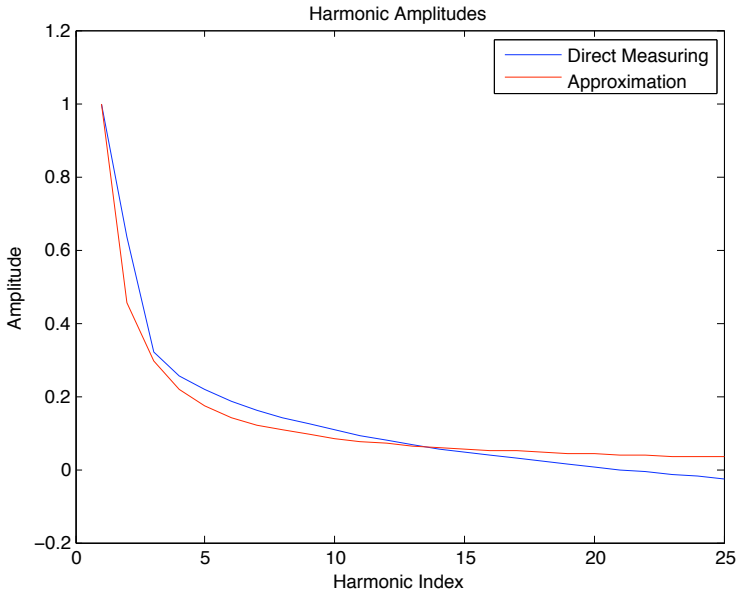



Figure 2.4: Direct Measurement and Approximation of Rhodes Spectral Envelope.

In figure 2.5 we plot different spectral envelope shapes with variable $a$ coefficient.

The result of this process is to obtain the de-mixed channels that have correct amplitudes in all their harmonics.

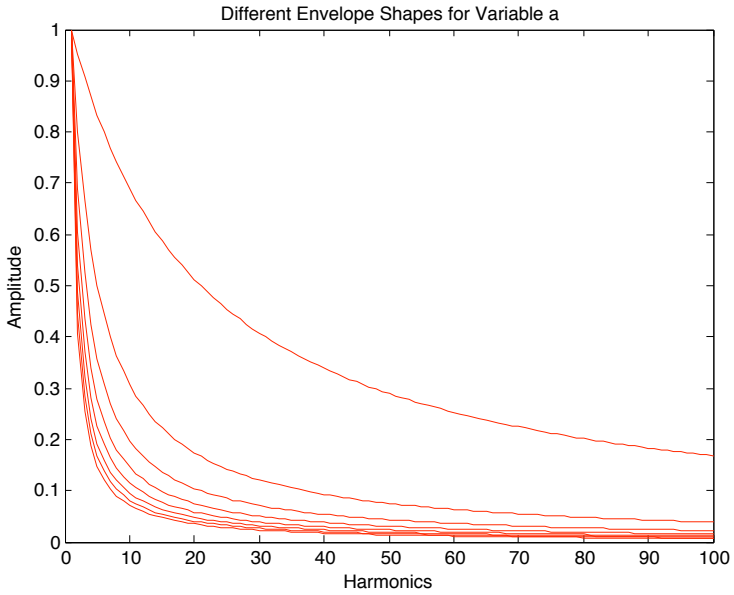



Figure 2.5: Different Spectral Envelope Shapes with Variable $a$ Coefficient.

# Chapter 3

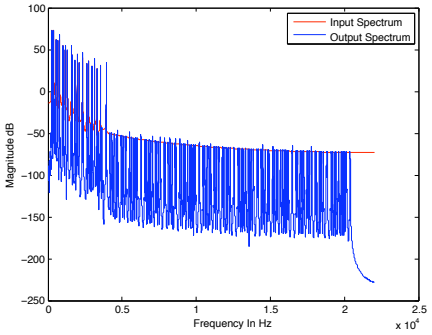
# Results

In this chapter we present the results of various tests that indicate the performance of the different stages that the whole system consists of.

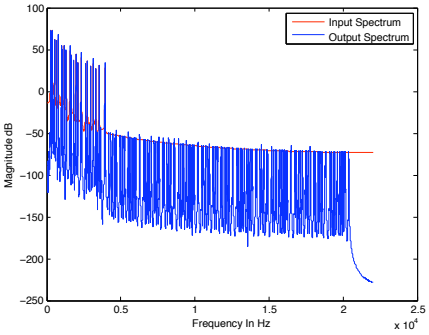
## 3.1 Filtering

To test how the behavior of the filters we fed the system with a harmonic non modulating synthesized polyphonic signal. We set the input and output note index to be the same in order not to do any pitch transformation.

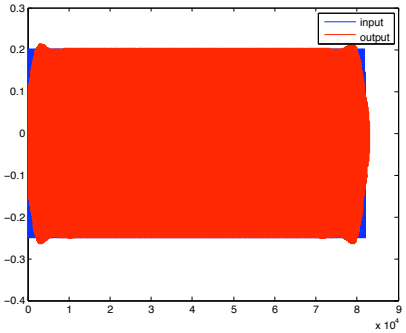
In figure 3.1 we see the filtering and mixing results when feeding the system with such a harmonic signal. We notice that the input output spectrum is identical over the hearing range. At the time domain we notice the rise of the bandpass filter step response, that happens both at the beginning and at the end as we use zero phase filtering. This is an inherent characteristic of the filters and cannot be avoided. In most cases though the attack time is not zero, so this is not considered to be a significant problem.




(a) Input Output Spectrum




(b) Input Output Spectrum zoomed




(c) Input Output - Time

Figure 3.1: System's Input -Output for Non - Modulating harmonic Signal

## 3.2 Vibrato Tracking

In order to test how the Vibrato Tracking performs for each implementation approach we had to go through different kinds of tests. These test consist of feeding the system with different kinds of signals, and then evaluating the MSE error of the input - output.

$$MSE_{dB} = 20 \cdot \log_{10} \cdot \frac{1}{N} \sum_{k=0}^{N-1} (x(k) - y(k))^2 \tag{3.1}$$

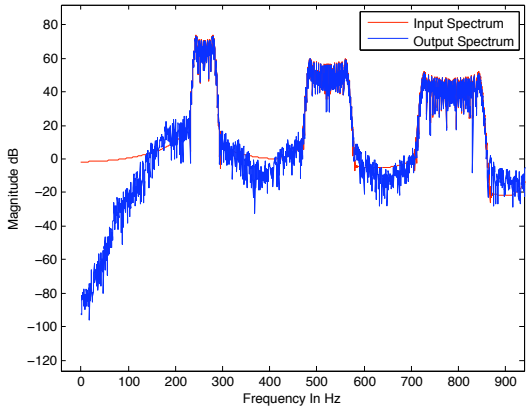
where $N$ is the length of the signal, and $x(\cdot)$, $y(\cdot)$ the input and the output of the system respectively.

To test how the system behaves with vibrato we fed the system with totally harmonic synthetic modulating sounds with different vibrato rate. We chose this approach because we need to be sure that the filters will be tuned correctly. If the sound was inharmonic we would introduce additional error, and the measurements we would take would not respond to reality. The synthetic signal we created is a weighted sum of sinusoids with frequencies that are modulating multiples of the fundamental.

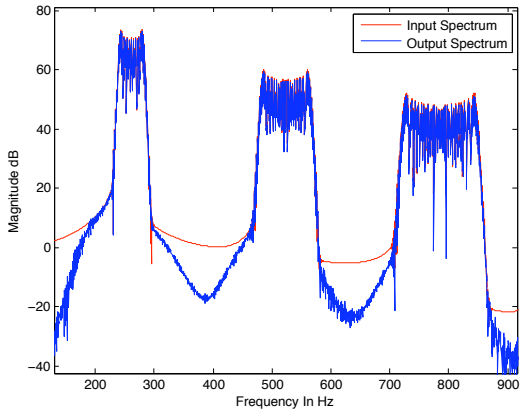
In the figure 3.2 we plot the input and output spectrum of the synthetic signal (in a selected frequency range), and its de-mixed - re - mixed version. We notice that the filter is tuned to the vibrato of the input signal.

In tables 3.1, 3.2 we plot MSE error in dB for different signal vibrato rate and different filter coefficients update rate. We notice that the larger the update rate of the filter coefficients (update for smaller vibrato deviations), the smaller the MSE in dB.

We chose to keep the per sample approach as it is showing some key advantages over the other approaches mentioned before. These are




(a) Vibrato Rate = 2.5Hz, Vibrato Depth = 20 Hz Vibrato Update Every 1 Hz of Difference, , $MSE_{dB} = -32.341$




(b) Vibrato Rate = 2.5Hz, Vibrato Depth = 20 Hz Vibrato Update Every 0.1 Hz of Difference, , $MSE_{dB} = -56.341$

Figure 3.2: Input, Output of Synthetic Spectrum

| Vibrato Rate Update (Deviation) | 0.5 | 0.1 | 0.05 | 0.01 |
|---|---|---|---|---|
| MSE dB | -52.88 | -64.11 | -68.62 | -71.03 |

Table 3.1: Vibrato Rate = 2.5 Hz

| Vibrato Rate Update (Deviation) | 0.5 | 0.1 | 0.05 | 0.01 |
|---|---|---|---|---|
| MSE dB | -77.64 | -78.6923 | -81.034 | -81.856 |

Table 3.2: Vibrato Rate = 5 Hz

- Relatively low complexity, as the filter is of low order, and we can optimize our algorithm to perform filtering operations in parallel, since these operations use the same data, and they are not depended from one another.

- High precision in frequency can be acquired easily, only by tuning the filters, in contrast to the time frequency domain approach in which we have to use quite long FFT size and peak interpolation which is a quite demanding process.

- Fast transient response is also achieved since there is no framing - windowing applied.

- The pitch of the instrument can be followed at any time and not only once during the frame duration. This has as a result not only to get a better quality harmonic part, but to obtain a more clean and uncorrelated to the harmonic part residual.

## 3.3 Inharmonicity

Also a very significant property is the inharmonicity that exists in several sounds, like piano and violin. We noticed that even though the first harmonics of a piano middle C ($C\#3$) are exact multiples of the fundamental frequency, deviations occur at the higher ones (1500 Hz and above). The same happens with bowed strings and guitar. Although they are considered harmonic, there are small deviations in higher harmonics, which start from lower for higher notes.

Even if the work conducted here regards harmonic sounds, we would like to present

a vulnerability of the current framework when we give as an input inharmonic sounds. Here each filter bank consists of filters tuned to multiples of the fundamental frequency, which means that if the harmonics deviate from their standard location, due to the filtering operation, they will be attenuated, and the final result will be less brighter than the original sound.

To solve this we could employ some physical modeling knowledge. The amount of inharmonicity of the $n^{th}$ harmonic can be given by

$$I_n = nf_1(n^2 - 1)A \tag{3.2}$$

where $n$ is the harmonic number, $f_1$ is the fundamental frequency, and A the inharmonicity factor and depends on the materials, the characteristics of the string and on the force that excites it . The issue here is that when we are given a sound, we know nothing about the characteristics of the string and how was it played, which means we cannot go to conclusions about the inharmonicity factor $A$.

To solve this problem we can use a hybrid approach carried out in both time and frequency domain. More specifically the filtering and the transformation operations would take place in time domain. The frequency domain would help us to solve for the inharmonicity factor $A$ by measuring the deviation of each harmonic from the standard location, and at the same time changing the values of the inharmonicity factor until we get exactly to the actual location each harmonic.

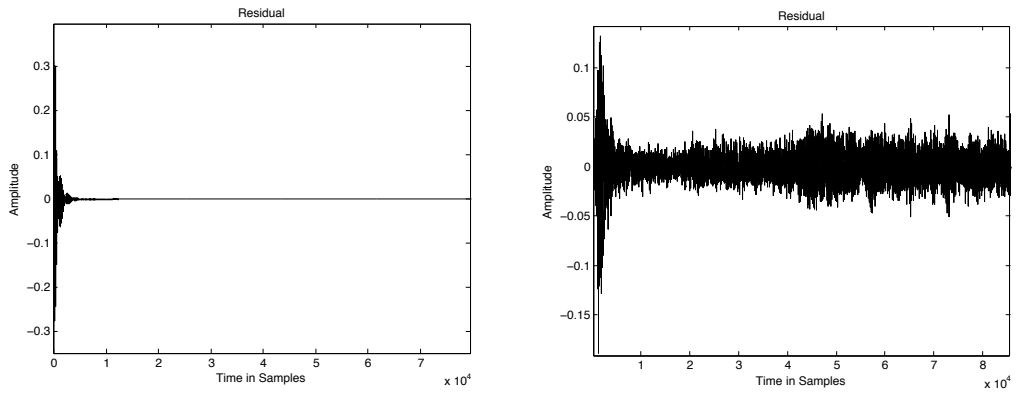
## 3.4 Obtaining Residual

Often sounds (like violin, piano, saxophone etc) feature a residual part apart from the harmonic one. To re-synthesize a good quality sound we should take in account also the residual part. Assuming that the sound is harmonic, and that we follow

the vibrato adequately, we can obtain the residual by re-mixing the result, without any pitch transforms) and subtracting it from the initial audio signal. Then to finally form the output audio we add to the transformed sound the residual part we obtained.

We performed this process for violin, piano and electric piano sound. The results for the electric piano and the violin were better as they are more harmonic than the piano sound. The residual part of the violin is the sound of the bow on the strings. The one of the electric piano is a bell sound that exists on Fender Rhodes, and finally for the piano is the hammer that hits it's strings.

Here we present the results for the violin and Rhodes(e-piano) sounds. For the violin we had to tune the filters a bit by inspecting the spectrum, to obtain a better residual that would not feature the harmonics that are not filtered out due to instrument's inharmonicity.




(a) Rhodes (b) Violin

Figure 3.3: Residual in time domain for e-piano and violin

In figure 3.3 we notice the small amplitudes of the residual part for both instruments. The listening tests were very encouraging, as for the e-piano we managed to isolate the inharmonic bell sound that comes up when we play a note, and for the violin the bow that excites the violin strings. In figure 3.3 we show the input output spectrum

for the e-piano.(no transformations included). We processed one note so that the spectrum would be simple enough to point out the residual part. We can see (in red) that around 5 kHz lies the inharmonic bell sound. This does not happen in the processed signal. To obtain the residual we subtract the two signals in time domain, and there is no loss because of the zero phase filtering.

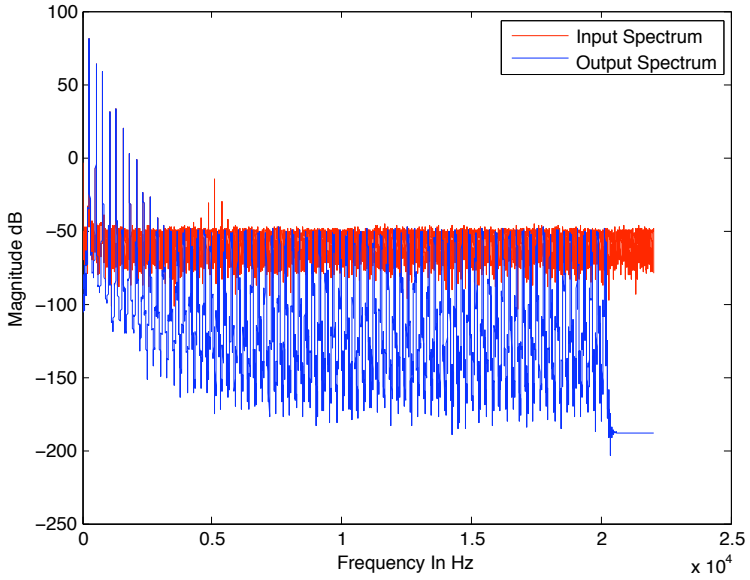



Figure 3.4: Rhodes. Input Output Spectrum $C3$

To obtain correctly the residual for non harmonic sounds we need to compensate for the inharmonicity as mentioned before.

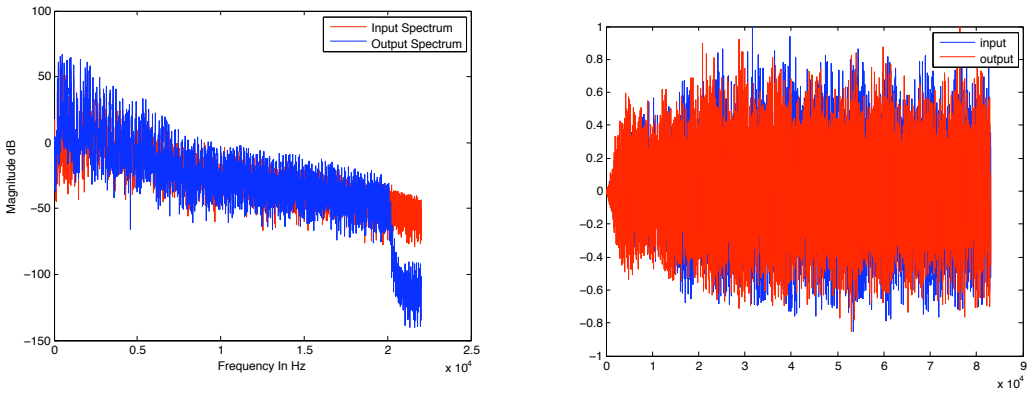
In order to measure the quality of real sounds we applied the whole process of de-mixing, chord transformation, re-mixing, and compared with a chord that featured the same notes as the transformed one. the sounding results were persuading and natural. In some cases we needed to increase the vibrato update rate, in order to avoid amplitude modulation in the transformed signal.

An upcoming issue is that the pitch shifting algorithms used, shift the phase of the signal smoothly in time. Although this is not perceived by the listener it adds a

significant amount of error when measuring it with criteria like MSE. Thus the need of having some perceptual criteria to measure the difference is apparent. A way to take only the amplitude information in account would be to examine the magnitude spectrums of these two signals by taking in account the human hearing threshold. Here we have to mention that we tested both pitch shifting algorithms (SOLA & Resampling, PSOLA) and they performed at the same levels in terms of MSE. Thus finally we chose to use the lighter one which is PSOLA, as the computation of the Pitch Marks it is quite simple for harmonic signals of known frequency.

## 3.5 More Sound Examples

Here we present the spectrums of the de-mixed - transformed (major to minor) signals versus the spectrums of the original minor signals. Also we show these signals in time domain.



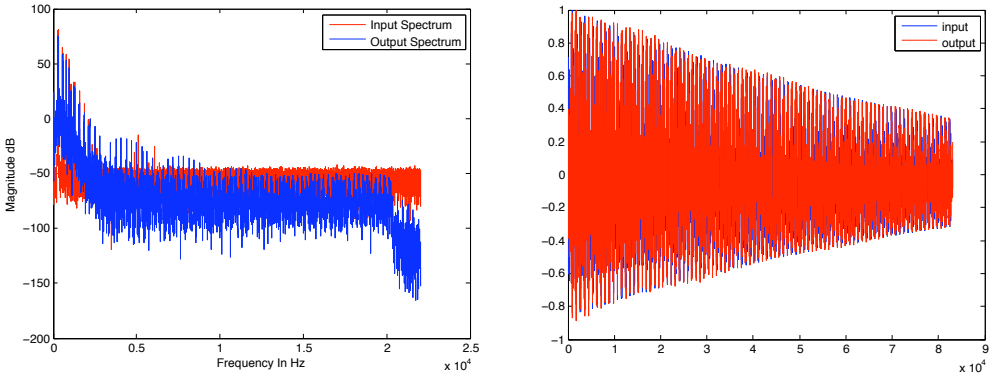

(a) Violin Minor Original vs Re-synthesized Spectrum

(b) Violin Minor Original vs Re-synthesized Time Domain

Figure 3.5: Violin Comparison

Violin is quite harmonic so the results we take are very good. We see that the spectrums of both signals are almost identical. Furthermore in the time domain

representation the transition effect of the filters does not degrade the signal as the attack stage of the violin sound is not very short.




(a) E-Piano Minor Original vs Re-synthesized Spectrum

(b) E-Piano Minor Original vs Re-synthesized Time Domain

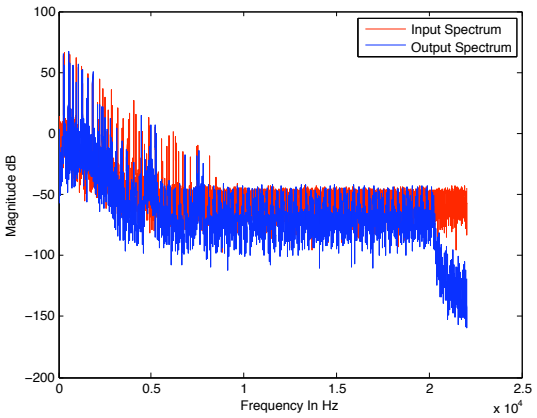
Figure 3.6: E-Piano Comparison

For the e-piano the results are even better. We have almost perfect match between the spectrums of the two signals. if we zoom-in at the figure we can see the effect of the filters on the transient. The original has almost zero attack. The transformed - re-mixed one features about 11 milliseconds of attack time.

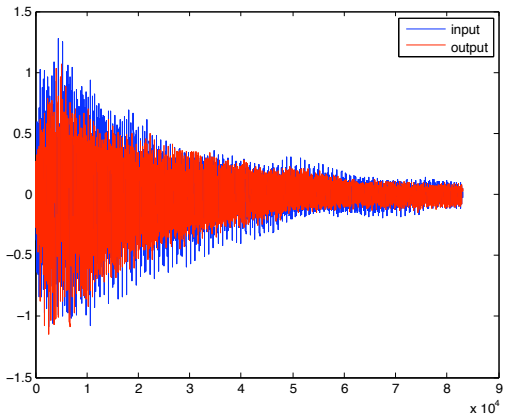
The results are not satisfactory for the piano sound. This sound is inharmonic, with an increasing harmonic position deviation as frequency increases. This has as a result not to filter correctly the signal at the de-mixing stage. Actually many of the harmonics are filtered out due to this fact.

The result is a blurred piano sound, with a low pass filtering effect. Furthermore the obtained residual features the harmonics that were filtered out at the de-mixing stage along with the hammer sound. This makes it useless, as if we add it to the harmonic part we will get a mix of the major chord along with the minor.

The sound examples can be found at `http://iua-share.upf.edu/wikis/smc/Chris_Stamat`.




(a) Piano Minor Original vs Re-synthesized Spectrum




(b) Piano Minor Original vs Re-synthesized Time Domain

Figure 3.7: Piano Comparison

# Chapter 4

# Conclusions

In this Thesis we implemented and experimented with a framework that given a source MIDI score, MIDI events like vibrato a target score and a polyphonic instrument source audio file, transforms the audio object in time domain, in terms of pitch shifting and time stretching, in order to form the target chord.

This framework, after it's completion can work as a module in a more integrated system. This system would consist initially of a transient and multi pitch detection stage, that would obtain information that describe the input signal. By using this information, the current framework would apply any desirable transformations on the input audio signal.

Furthermore, this framework can also be used to filter out ambient noise that can be captured during a live performance, by maintaining at the same time the timbre of the instrument.

During carrying out this Thesis we dealt mostly with harmonic sounds coming from Rhodes and violin. For the system built, we reviewed and tested it's different aspects, like the vibrato tracking, the tuning and the behavior of the filter banks on inharmonic sounds and the quality of the obtained residual part. We also pointed out it's vulnerabilities like sound blur when dealing with inharmonic sounds and

proposed methods to overcome these issues.

# Chapter 5

# Further Work and Directions

In this section we point out the vulnerabilities of the framework at the current development stage, and we mention the directions we will follow until the final completion of this thesis.

- **Dealing successfully with inharmonic sounds**. When processing inharmonic sounds like piano, the algorithm performs well for frequencies up to 2 kHz, where the signal follows a harmonic pattern. Above that frequency the harmonics deviate significantly. Since the filter bank is tuned at fixed frequencies that follow a harmonic pattern, it is expected that the de-mixing stage will fail to filter them out correctly. The result is a blury piano sound that lacks presence and clarity. Also when trying to obtain the residual part, in order to mix it with the transformed audio, we notice that apart from the piano hammer sound that operates on the strings we also add these harmonics that we failed to filter out correctly at the previous stage.

  Further work on this stage would include the analysis of the harmonic pattern of several inharmonic sounds [16], and the derivation of such formulas that we would embed at the filter bank tuning stage. This would have as a result the correct filtering of such sounds, better quality in the transformed audio,

in terms of presence, and finally better residual quality.

- **Compensating overlapping harmonics**. We presented a way to deal with overlapping harmonics when the amount of overlap is 100%. An interesting direction to follow would be to compensate when the overlap factor is less than 100%, which means that a harmonic lies within the range of the filter which is used to filter out the harmonic of our interest.

- **Benchmarking the approaches mentioned in 2.2.2**. Arbitrary Vibrato Update Rate and Frame Processing for Vibrato Update are the two different implemented approaches. We need to spend some time to evaluate and tune both of them to measure their performance, to find which one is more suitable for several classes of sounds.

- **Multipitch Detection**. This will not be implemented here, but it is a very appealing direction to follow in research, implementation and experimentation. Several studies have been made on this field [14], [15],[17] , and with good results. It would be very interesting to create a framework that does a multi pitch and transient analysis of polyphonic signals and provides with relevant information the current one.

# Chapter 6

# Code

In this chapter we present the code development that carries out the several operations of the developed framework

## 6.1 Main Script

Listing 6.1: Main Script that calls the different analysis - transformation functions implemented

```
clear;
clc;
close all;
Fs = 44100;


%Creating frequency Matrix for 3rd Octave
for i = 1 : 13
    freqs(i) = 6.875* 2 ^ ((3 + 60 + (i-1))/12);
end

%Loading notes from files
x = zeros(13, 88200);
filelist = dir('./violin2/*.wav');
for i = 1 : length(filelist)
    [in, Fs] = wavread(['./violin2/' filelist(i).name]);
    in = in(:,1)';
    in = in(1: 88200);
    x(i,:) = in;
end

%Forming chord C3 Major
input = x(1,:)+ x(5,:);%+ x(8,:);% + x(12,:);
input = input ./ max(input);

% [input, Fs] = wavread('./guitar/guitar2.wav');
```

```
% input = input(1:88200,1)';

% ambience = wavread('voice4.wav')';
% ambience = ambience(88201+1:2*88200);
% input = ambience;

% Note index will be fed to functions to give note info
noteIdx = [1 5];
outNoteIdx = [1 4];

%Getting the filter multiplier Coefficients (ones if no harmonic
%overlapping)

filterMult = assignCoeff(noteIdx);


% Process -> Demixing and Mixing the signal
% Any pitch shifting will be called within this function.

% input = 0.25*syntheticSignal(freqs(1));% + 0.25*syntheticSignal(freqs(5)) + 0.25*syntheticSignal(f
% [y, reMixedOriginal] = filt1_harm_v2(input,noteIdx,outNoteIdx, filterMult);

% [y, CHANNELS,residual] = filt1_harm_v7(input,noteIdx,outNoteIdx,filterMult);
% input = syntheticSignal();
[y, CHANNELS,reMixedOriginal] = filt1_harm_v8Final(input,noteIdx,outNoteIdx, filterMult);
% [y, CHANNELS,reMixedOriginal] = filt1_harm_v9(input,noteIdx,outNoteIdx, filterMult);


% [y, reMixedOriginal] = filt1_harm_v3(input,noteIdx,outNoteIdx, filterMult);


input = x(1,:) + x(5,:);

% Getting spectrums of I/O

% x = x(5000:10000);
% y = y(5000:10000);
nFFT = length(x)-1;
F = 0 : Fs/nFFT : Fs/2 - Fs/nFFT;
X = fft(input(1,:) .* hanning(length(x))', nFFT);
X = 20*log10(abs(X(1:0.5*nFFT)));
figure,plot(F,X,'-r'); hold on;

% y = y ./ max(abs(y));
Y = fft(y.*hanning(length(y))', nFFT);
Y = 20*log10(abs(Y(1,1:0.5*nFFT)));
plot(F,Y,'-');
legend ('Input Spectrum','Output Spectrum');
xlabel('Frequency In Hz')
ylabel('Magnitude dB')
hold off;


% residual = input - reMixedOriginal(1:length(input));
% output = y(1:length(input)) + residual;

% O = fft(output .* hanning(length(x))', nFFT);
% O = (abs(O(1:0.5*nFFT)));
%
% In = fft(x(4,:) .* hanning(length(x))', nFFT);
% In = (abs(In(1:0.5*nFFT)));
%
```

```
% I = find(In > max(In) - 40);
% MSE = (1/length(I)) * sum((O(I) - In(I)).^2)

% L = 10000;
% msedB = (1/L) * sum((output(L:25*L) - x(4,L:2*L)).^2)



% sound(x,Fs)
% sound(y,Fs)
%
wavwrite(input,Fs,'Violin2maj.wav')
wavwrite(y,Fs,'Violin2min.wav')
```

## 6.2 Processing Per Note Duration

Listing 6.2: Filtering and Transformation per Note Duration

```
function [y, fVec, reMixedOriginal] = filt1_harm_v2(x, noteIdx, outNoteIdx, filterMult)



%Creating frequency index
for i = 1 : 13
    freqs(i) = (6.875* 2 ^ ((3 + 60 + (i-1))/12));
end

N = length(noteIdx);
Fs = 44100;
p = 3;    %IIR filter order
p_f = 101; % FIR filter order

chanOut = zeros(N, length(x)); %each row contains a demixed channel


M = 1; %oversampling factor
Fs = M*Fs;
x = upsample(x,M);

y = zeros(1, length(x));
reMixedOriginal = zeros(1,length(x));


load e-piano.mat;


nHarms = 100;
L = nHarms;

for i = 1 : N

    L = nHarms;
    mix = zeros(1,length(x));
    j = 1;

    fVec = zeros(1,nHarms);    %contains fundamental and harmonic frequencies nHarm*f0
    for j = 1 : nHarms
        fVec(j) = freqs(noteIdx(i)) * j;
        j = j + 1;
```

```
    end


    I = find(fVec > .5*Fs/M - 2050);


    if(length(I) > 0)
        L = min(nHarms, I(1));
    end

    num = zeros(L, 2*p+1);
    den = zeros(L, 2*p+1);

    %IIR Filter Design
    for k = 1 : L
        w1 = (fVec(k) - 7) / (0.5*Fs);
        w2 = (fVec(k) + 7) / (0.5*Fs);
        %d = fdesign.bandpass('n,fc1,fc2',10,w1, w2); %for higher order
%         filters
        %Hd(k) = design(d,'butter');
        [num(k,:), den(k,:)] = butter(p, [w1 w2],'z');

    end

    % filtering with badpasses for all harmonics of each channel
    for k = 1 : L
        out(k,:) = filtfilt(num(k,:), den(k,:), x);%.* 0.9 * filterMult(i,k);
        %out(k,:) =filter(Hd(k), x);
        %out(k,:) =filter(Hd(k), out(k,end:-1:1)).* filterMult(i,k);
        %out(k,:) = out(k,end:-1:1);
    end

%     summing all the harmonics of each fundamental
    for j = 1 : L
        mix = mix + out(j,:);
    end

    reMixedOriginal = reMixedOriginal + mix;

    %---Here the pitch shifting will take place----
    if(noteIdx(i) ~= outNoteIdx(i))
        ratio = freqs(outNoteIdx(i)) / freqs(noteIdx(i))
        Pmarks = computePmarks(mix, freqs(noteIdx(i)), Fs);
        outmix = psola(mix,Pmarks,1,ratio);

    else
        outmix = mix;
    end

    %-----------------------------------------------

    CHANNELS(i,:) = mix;


    % Mixing the final signal
    y = y + outmix;
end

y = downsample(y,M);
```

## 6.3 Processing Per Frame Duration

Listing 6.3: Filtering and Transformation per Frame Duration

```
function [y, originalRemixed] = filt1_harm_v3(xIn, noteIdx, outNoteIdx, filterMult)

% x -> input signal
% N -> fft size
% w -> window
% H -> hop size

%Creating frequency index
for i = 1 : 13
    freqs(i) = 6.875* 2 ^ ((3 + 60 + (i-1))/12);
end

N = length(noteIdx);
Fs = 44100;
p = 3;    %IIR filter order


chanOut = zeros(N, length(xIn)); %each row contains a demixed channel



nHarms = 100;
L = nHarms;
M = 2500;        %window length
H = 500;         %hop size;
w = hanning(M); %window

pin = 1;
pend = length(xIn) - M;

xIn = [zeros(1,2*M),xIn];

y = zeros(1,length(xIn));
mix = zeros(1,length(xIn));
originalRemixed = zeros(1, length(xIn));


%chanOut = zeros(N, length(xIn)); %each row contains a demixed channel


%sumWins = zeros(1,length(xIn)); %Summing the windows to divide output to cancel AM

mix = zeros(1, length(xIn));
filteredFrame = zeros(1, length(xIn));

load violin2Pitch.mat;
noteVibrato = [zeros(1,2*M),xIn];

num = zeros(1, 2*p+1);
den = zeros(1, 2*p+1);

for i = 1 : N
    L = nHarms;
    noteVibrato = [zeros(1,2*M) freqInfo(noteIdx(i),:) - freqs(noteIdx(i))];
    I = find(noteVibrato > 12);
    noteVibrato(I) = 12;
    I = find(noteVibrato < -12);
```

```
noteVibrato(I) = -12;

%plot(noteVibrato);


fVec = zeros(1,nHarms);   %contains fundamental and harmonic frequencies nHarm*f0
for j = 1 : nHarms
    fVec(j) = freqs(noteIdx(i)) * j;
end

I = find(fVec > 0.5*Fs - 2050 );


if(length(I) > 0)
    L = min(nHarms, I(1));
end


out = zeros(1, length(xIn));
mix = zeros(1, length(xIn));
%freq1 = [];
%freq2 = [];

%FOR EACH HARMONIC OF EACH NOTE
for k = 1 : L
    out = zeros(1, length(xIn));
    disp([ num2str(k/L*100) , '%' ]);

    filteredFrame = zeros(1,M);

    pin = 1;
    pend = length(xIn) - M;

    %         w1 = (fVec(k) + meanFrameVibrato - 20) / (0.5*Fs);
    %         w2 = (fVec(k) + meanFrameVibrato + 20) / (0.5*Fs);
    %         [num(k,:), den(k,:)] = butter(p, [w1 w2],'z');


    while pin < pend

        meanFrameVibrato = noteVibrato(floor((2*pin+1+M) / 2));%mean(noteVibrato(pin+1:pin+M))

        %freq1 = [freq1 fVec(k) + meanFrameVibrato - 12]
        %freq2 = [freq2 fVec(k) + meanFrameVibrato + 12]


        w1 = (fVec(k) + meanFrameVibrato - 20) / (0.5*Fs);
        w2 = (fVec(k) + meanFrameVibrato + 20) / (0.5*Fs);
        %disp(' estimating filter ');
        [num, den] = butter(p, [w1 w2],'z');

        x = xIn(pin+1 : pin+M);

        % filtering with badpasses for all harmonics of each channel
        filteredFrame = filtfilt(num, den, x).* 0.9 *filterMult(i,k);
        filteredFrame = filteredFrame .*w';

        % summing all the harmonics of each fundamental
        out(pin+1 : pin + M ) = out(pin+1 : pin + M ) + filteredFrame * H/M;


        %summing windows
```

```
            %sumWins(pin+1 : pin + M ) = sumWins(pin+1 : pin + M ) + w';
            pin = pin + H;

        end

        mix = mix + out;
    end

    %---Here the pitch shifting will take place----
    if(noteIdx(i) ~= outNoteIdx(i))
        ratio = freqs(outNoteIdx(i)) / freqs(noteIdx(i));
        Pmarks = computePmarks(mix, freqs(noteIdx(i)), Fs);
        outmix = psola(mix,Pmarks,1,ratio);
    else
        outmix = mix;
    end

    y = y + outmix;
    originalRemixed = originalRemixed + mix;


end

y = y(2*M+1:end);
originalRemixed = originalRemixed(2*M+1:end);
y = y ./ max(abs(y));
```

## 6.4 Processing Per Sample

Listing 6.4: Filtering and Transformation per Sample

```
function [y, fVec, residual] = filt1_harm_v8(x, noteIdx, outNoteIdx, filterMult)


% x = x(1:5000);

%Creating frequency index
for i = 1 : 13
    freqs(i) = (6.875* 2 ^ ((3 + 60 + (i-1))/12));
end

N = length(noteIdx);
Fs = 44100;
p = 3;   %IIR filter order


% chanOut = zeros(N, length(x)); %each row contains a demixed channel

y = zeros(1, length(x));



load violin2Pitch.mat;

vibrato = zeros(1,length(x));
residual = zeros(1,length(x));

nHarms = 25;
L = nHarms;
```

```
mix = zeros(1,length(x));
out = zeros(1,length(x));
out2 = zeros(1,length(x));


num = zeros(1, 2*p+1);
den = zeros(1, 2*p+1);


for i = 1 : N

    fVec = zeros(1,nHarms); %contains fundamental and harmonic frequencies nHarm*f0

    harmCount = 10;
    for j = 1 : nHarms

        fVec(j) = freqs(noteIdx(i)) * j;

    end

    %     plot(fVec);
    L = nHarms;
    mix = zeros(1,length(x));
    mix2 = zeros(1,length(x));


    I = find(fVec >= 0.5*Fs - 2050); % Harmonics Up To 20000 Hz


    if(length(I) > 0)
        L = min(nHarms, I(1));
    end

    % vibrato around the center frequency
%     [sig, vibrato] = syntheticSignal();
%
%     vibrato = vibrato - freqs(noteIdx(i));

    vibrato = freqInfo(noteIdx(i),:) - freqs(noteIdx(i));
    Vibrato = vibrato - min(vibrato)+1;

    threshold = .5; %in Hz

    %figure, plot(Vibrato);

    tic
    index = 1;
    count = 2;
    IDX = [];
    IDX(1) = 1;
    while(index < length(x))
        I = find(abs(Vibrato(index) - Vibrato(index:end)) >= threshold);
        if max(I) > length(x)
            break;
        end

        if(length(I) >=1)
            index = index + I(1);
        else
            index = index + 500;
```

```
        end

        IDX(count) =  index;
        count = count+1;

    end
    toc
    %     vibrato = zeros(1,88200);
%
%     figure, plot(IDX,'+black');
%     title('Vibrato Updates');
%     xlabel('Index No');
%     ylabel('Time In Samples')



    for k = 1 : L

        w1 = (fVec(k) - 80) / (0.5*Fs);
        w2 = (fVec(k) + 80) / (0.5*Fs);
        [num, den] = butter(p, [w1 w2],'z');



        cnt = 2;

        b = zeros(1,7);
        a = zeros(1,6);

        mix = zeros(1,length(x));
        for index = 1 : length(x)

            if(index == IDX(cnt) && (cnt) < length(IDX))
                %IIR Filter Design
                w1 = (fVec(k) + k * vibrato(IDX(cnt)) - 80) / (0.5*Fs);
                w2 = (fVec(k) + k * vibrato(IDX(cnt)) + 80) / (0.5*Fs);

                if w2 >=1 break; end

                [num, den] = butter(p, [w1 w2],'z');
                cnt = cnt+1;
            end


            % Filtering with badpasses for all harmonics of each channel

            b(1) = x(index);

            % Filtering |(FORWARD FILTERING)|
            out(index) = num(1) * b(1) + num(2) * b(2) + num(3) * ...
                b(3) + num(4) * b(4) + num(5) * b(5) + num(6) * ...
                b(6) + num(7) * b(7) - ...
                den(2) * a(1) - den(3) * a(2) - den(4) ...
                * a(3) - den(5) * a(4) - den(6) * a(5) - den(7) * a(6);

            b(7) = b(6);
            b(6) = b(5);
            b(5) = b(4);
            b(4) = b(3);
            b(3) = b(2);
            b(2) = b(1);
```

```
            a(6) = a(5);
            a(5) = a(4);
            a(4) = a(3);
            a(3) = a(2);
            a(2) = a(1);
            a(1) = out(index);

            mix(index) = mix(index) + out(index);

        end




        b = zeros(1,7);
        a = zeros(1,6);
        for index = length(x):-1:1

            if(index == IDX(cnt))
                %IIR Filter Design

                w1 = (fVec(k) + k * vibrato(IDX(cnt)) - 80) / (0.5*Fs);
                w2 = (fVec(k) + k * vibrato(IDX(cnt)) + 80) / (0.5*Fs);

                if w2 >=1 break; end

                [num, den] = butter(p, [w1 w2],'z');
                cnt = cnt-1;
            end


            % Filtering with badpasses for all harmonics of each channel

            b(1) = mix(index);

            % Filtering |(FORWARD FILTERING)|
            out2(index) = num(1) * b(1) + num(2) * b(2) + num(3) * ...
                b(3) + num(4) * b(4) + num(5) * b(5) + num(6) * ...
                b(6) + num(7) * b(7) - ...
                den(2) * a(1) - den(3) * a(2) - den(4) ...
                * a(3) - den(5) * a(4) - den(6) * a(5) - den(7) * a(6);

            b(7) = b(6);
            b(6) = b(5);
            b(5) = b(4);
            b(4) = b(3);
            b(3) = b(2);
            b(2) = b(1);

            a(6) = a(5);
            a(5) = a(4);
            a(4) = a(3);
            a(3) = a(2);
            a(2) = a(1);
            a(1) = out2(index);

            mix2(index) = mix2(index) + out2(index);

        end

    end
```

```
    mix2 = mix2 .* filterMult(i,k);
    residual = residual + mix2;

    %---Here the pitch shifting will take place----
    if(noteIdx(i) ~= outNoteIdx(i))
        ratio = freqs(outNoteIdx(i)) / freqs(noteIdx(i))
        Pmarks = computePmarks(mix2, freqs(noteIdx(i)), Fs);
        outmix = psola(mix2,Pmarks,1,ratio);
        %              outmix = pshift(mix2,ratio);
    else
        outmix = mix2;
    end


    % Mixing the final signal

    y = y + outmix;

end
%normalizing output to inputs gain.
y = y ./ max(abs(y)) * max(abs(x));
% y = y(tailLen+1: end - tailLen);
```

## 6.5 Compensation for Overlapping Harmonics

Listing 6.5: Compensation for Overlapping Harmonics. This function returns gain the coefficients of each filter used

```
function [coeffs] = assignCoeff(noteIdx)


% Loading Notes
x = zeros(13, 88200);
filelist = dir('./e-piano/*.wav');
for i = 1 : length(filelist)
    [in, Fs] = wavread(['./e-piano/' filelist(i).name]);
    in = in(:,1)';
    in = in(1:88200);
    x(i,:) = in;
end


%Creating frequency index
for i = 1 : 13
    freqs(i) = 6.875* 2 ^ ((3 + 60 + (i-1))/12);
end


fVec = [freqs(noteIdx)];
load violin.mat;

%fVec  fundamental frequencies
%spEnv harmonic coefficients

[r,c] = size(spEnv);
nHarms = c;

%Contsruct Frequency Matrix
```

```
fMatrix = zeros(length(fVec), nHarms);

for i = 1 : length(fVec)
    for j = 1 : nHarms
        fMatrix(i, j) = fVec(i) * j;
    end
end

fMatrix = round(fMatrix);

[r,c] = size(fMatrix);

%transforming coeff matrix to vector for the "find" function compatibility

coeffMatrix = [];
for i = 1 : length(fVec)
    coeffMatrix(i,:) = spEnv(i,:);
end
coeffVec = coeffMatrix(:);

[r,c] = size(coeffMatrix);
parsedIdx = zeros(r,c);

count = 1;
flag = 0;

for i = 1 : r
    for j = 1 : c
        I = find(fMatrix == fMatrix(i,j));


        if length(I) == 1 % no overlapping harmonics
            coeffs(i,j) = coeffVec(I)/coeffVec(I);

        else
            %checking to see if we passed the same indexes for secnd time
            %because if yes we have to invert the indexes
            for k = 1 : r
                check = parsedIdx(k,1:length(I)) - I';
                if (check == zeros(1,length(I))) flag = 1;
                    break;
                end
            end

            if (flag == 0)
                coeffs(i,j) = coeffVec(I(1)) / (coeffVec(I(1)) + coeffVec(I(2)));
            elseif(flag == 1)
                coeffs(i,j) = coeffVec(I(2)) / (coeffVec(I(1)) + coeffVec(I(2)));
                flag = 0;
            end

             parsedIdx(count, 1:length(I)) = I;
             count = count+1;
        end
    end
end
```

## 6.6 Measurement of the Timbre of the instrument

Listing 6.6: Measurement of the instruments timbre by data analysis or by approximation

```
clc;
filelist = dir('./violin2/*.wav');

nHarms = 25;
spEnv = zeros(length(filelist), nHarms); % spectral envelope values
spEnv2 = zeros(length(filelist), nHarms); % spectral envelope values

fs = 44100;

a = 0.3

for i = 1 : length(filelist)

    Hz = 6.875* 2 ^ ((3 + 60 + (i-1))/12);  %note to Hz;
    bin = floor((Hz / fs * 4096)) ;          % Hz to fft Bins



    [x, fs] = wavread(['./violin2/' filelist(i).name]);
    x = x(:,1:end)';
    disp(filelist(i).name);

    [X, f] = plot_fft(x(floor(length(x)/2) : floor(length(x)/2)+10000), 22050);
    X = X ./ max(X);

    %     figure, plot(X); %hold on;

    for j = 1 : nHarms
        idx = bin * j;

        % warping spectrum to get correct values
        % because the mapping from Hz to bin is not always
        % bin accurate
        if(idx < 2048-16)
            range = 15;
            [val,pos] = max([X(idx - range : idx + range)]);
            spEnv(i,j) = X(idx - range + pos - 1);
        else
            spEnv(i,j) = 0;
        end

        spEnv2(i,j) = 1 / (1+a*(j-1));
    end
end

save violin.mat spEnv;


figure, plot(1:nHarms, (spEnv(1,:)), 1:nHarms, (spEnv2(1,:)),'r');
legend('Direct Measuring','Approximation')
xlabel('Harmonic Index');
ylabel('Amplitude');
title('Harmonic Amplitudes');
```

## 6.7 Synthetic Signal

Listing 6.7: Creation of a synthetic harmonic signal that features vibrato. Used to measure analysis - reconstruction performance

```
function [y, vibrato] = syntheticSignal(freq);

nHarms = 10;

% freq = 261.6256;
Fs = 44100;
x = zeros(1,88200);

vibrato = zeros(1,88200);
F = 1.5;
Gain = 20.5;
vibrato = Gain * ( sin(2*pi*F*(1:88200)/Fs));

vibrato = zeros(1,88200);

phase = 0;

for i  = 1 : nHarms
    for j = 1 : 82000
        x(j) = x(j) + (1/((i+1)^3))*cos(phase);

        phase = phase + 2*pi*(freq*i +  i*vibrato(j)) / Fs;

    end
end

y = x;
vibrato = freq+vibrato;

y = y ./ max(abs(y));
```

# Bibliography


[1] Jordi Bonada Sanjaume. *Audio Time-Scale Modification in the Context of Professional Audio Post-production*. Research work for PhD Program Informatica i Cominicacio digital.

[2] Zölzer, U. e. a., *DAFX Digital Audio Effects*, John Wiley & Sons, eds., 2002.

[3] S. Roucos, A.M. Wilgus. *High quality time-scale modification for speech.* In Proc. ICASSP, pp. 493-496, 1985.

[4] Sanjit K. Mitra. *Digital Signal Processing. A Computer Based Approach* McGraw Hill , 3rd Edition2006,

[5] Alain de Cheveigne, Hideki Kawahara *Yin, a Fundamental Frequency Estimator for Speech and Music*, PROOF COPY 020204JAS

[6] A. Krukowski, I.Kale, G.D. Cain *Decomposition of IIR Transfer Functions Into Parallel Arbitrary-Order IIR Subfilters.* IEEE Nordic Signal Processing Symposium (NORSIG' 96), Dipoli Convension Center & Department of Electrical Engineering Espoo, Finland. 24-27 Sep.96

[7] E. Moulines and F. Charpentier. *Pitch synchronous waveform processing techniques for text-to speech synthesis using diphones. Speech Communication*, 9(5/6):453-467, 1990.

[8] Tuomas Virtanen, *Sound Source Separation in Monaural Music Signals Tampere University of Technology*, Finland, November 2006

[9] Mark Every, *Separation of Musical Sources and Structure from Single-Channel Polyphonic Recordings* University of York, UK, February 2006

[10] Anssi Klapuri, *Signal Processing Methods for the Automatic Transcription of Music* Tampere University of Technology, Finnland, March 2004

[11] Emmanuel Vincent, *Instrument Models for Source Separation and Transcription of Music Recordings* University of Paris 6, France, December 2004

[12] J-L. Durrieu, G. Richard, B. David, *Singer Melody Extraction in Polyphonic signals using source spearation methods*, in Proc of ICASSP'08.

[13] . Gillet, G. Richard. *Transcription and separation of drum signals from polyphonic music.* in IEEE Trans. on ASLP, Volume 16, N°3, March 2008 Page( s):529 - 540.

[14] Matti Karjalainen and Tero Tolonen *Multi-Pitch and Periodicity Analysis Model for Sound Separation and Auditory Scene Analysis.*Helsinki University of Technology Laboratory of Acoustics and Audio Signal Processing

[15] Arshia Cont, Shlomo Dubnov, David Wessel *Realtime Miltiple-Pitch and Multiple-Instrument Recognition for Musical Signals Using Sparse Non-Negative Constraints.*Proc. of the 10th Int. Conference on Digital Audio Effects (DAFx-07), Bordeaux, France, September 10-15, 2007.

[16] Bruno L. Giordano (1), Davide Rocchesso (2). *The effect of inharmonicity on the perceived quality of piano tones*

[17] Patricio de la Cuadra, Aaron Master, Craig Sapp *Ef[U+FB01]cient Pitch Detection Techniques for Interactive Music* Center for Computer Research in Music and Acoustics, Stanford University.

[18] Amari, S., Cichocki A., Yang, H.H., *A new learning algorithm for blind signal separation*, NIPS 95, MIT Press, 8, 1996

[19] Puntonet, C., Mansour A., Jutten C., *A geometrical algorithm for blind separation of sources*, GRETSI, 1995

[20] Bell, A.J., Sejnowski T.J., *An information-maximization approach to blind separation and blind deconvolution*, Neural Computation 7, 1995.

[21] Jordi Sole Casals, Christian Jutten, Annise Taleb *Source Separation Techniques Applied to Linear Prediction*