Department of Signal Processing and Acoustics

REMES Final Report

The Finnish Work Environment Fund TSR Project no. 113252

Ville Mäntyniemi, Rémi Mignot, and Vesa Välimäki

Aalto University | SCIENCE + TECHNOLOGY | RESEARCH REPORT

Aalto University publication series
SCIENCE + **TECHNOLOGY** 16/2014

REMES Final Report

The Finnish Work Environment Fund TSR Project no. 113252

Ville Mäntyniemi, Rémi Mignot, and Vesa Välimäki

Aalto University
School of Electrical Engineering
Department of Signal Processing and Acoustics

Aalto University publication series
SCIENCE + **TECHNOLOGY** 16/2014

ISBN 978-952-60-5963-1 (printed)
ISBN 978-952-60-5964-8 (pdf)
ISSN-L 1799-4896
ISSN 1799-4896 (printed)
ISSN 1799-490X (pdf)
http://urn.fi/URN:ISBN:978-952-60-5964-8

Unigrafia Oy
Helsinki 2014

Finland

This work was fund by The Finnish Work Environment Fund
(Työsuojelurahasto) Grant no. 113252

441 697
Printed matter

Author
Ville Mäntyniemi, Rémi Mignot, and Vesa Välimäki

Name of the publication
REMES Final Report

Publisher School of Electrical Engineering

Unit Department of Signal Processing and Acoustics

Series Aalto University publication series SCIENCE + TECHNOLOGY 16/2014

Field of research Audio signal processing

Abstract

This report summarizes the results of the REMES project, "Realistic Machine and Environmental Sounds for a Training Simulator to Improve Safety at Work". Its aim is to improve the sound environment in working machine simulators.

Modern simulators are visually and operationally extremely advanced and realistic, but the sound environment is still limited. By improving the sounds in these simulators, the simulators can grow into exceptionally realistic training tools with the ability to fully educate future operators in a completely safe environment.

Existing sounds are improved and new sounds are created for three different simulator types: a forest harvester and forwarder simulator, a drill rig simulator, and a truck-mounted hydraulic platform simulator. The main sound types synthesized are hydraulic sounds, drilling sounds, feeding and delimbing sounds (forest machines), and basic contact sounds.

Linear predictive coding is used throughout the project in synthesizing several different sounds, including hydraulic sounds and harvester sounds. Spectral subtraction and non-negative matrix factorisation are also widely utilized in creating noiseless contact sound samples required in the forwarder simulator and noiseless drilling sound for the rig simulator. Other methods used include filtering, amplitude envelope extraction, and peak detection.

Keywords acoustics, drilling, forestry machines, hydraulics, simulators

ISBN (printed) 978-952-60-5963-1	**ISBN (pdf)** 978-952-60-5964-8	
ISSN-L 1799-4896	**ISSN (printed)** 1799-4896	**ISSN (pdf)** 1799-490X
Location of publisher Helsinki	**Location of printing** Helsinki	**Year** 2014
Pages 4+102	**urn** http://urn.fi/URN:ISBN:978-952-60-5964-8	

Tekijä
Ville Mäntyniemi, Rémi Mignot, and Vesa Välimäki

Julkaisun nimi
REMES Final Report

Julkaisija Sähkötekniikan korkeakoulu

Yksikkö Signaalinkäsittelyn ja akustiikan laitos

Sarja Aalto University publication series SCIENCE + TECHNOLOGY 16/2014

Tutkimusala Audiosignaalinkäsittely

Tiivistelmä

Tämä raportti esittelee tulokset REMES-hankkeesta, jonka tavoitteena oli parantaa työkonesimulaattoreiden ääniympäristöä.

Modernit simulaattorit ovat visuaalisesti ja ohjaustoiminnoiltaan erittäin kehittyneitä ja todenmukaisia, mutta niiden ääniympäristö on varsin rajoittunut. Kun ääntä saadaan parannettua, työkonesimulaattoreista tulee erittäin realistisia opetuslaitteita, joilla voidaan täydellisesti kouluttaa tulevia operaattoreita turvalliseen työympäristöön.

Kolmen eri työkonesimulaattorin nykyisiä ääniä parannetaan ja laajennetaan: harvesteri- ja kuormatraktorisimulaattori, poralaitteistosimulaattori ja autoalustaisen hydraulisen nostolavan simulaattori. Keskeiset syntetoitavat äänityypit ovat hydrauliset, poraus-, syöttö- ja karsimis- (harvesterissa) sekä kontaktiäänet.

Lineaariennustuskoodausta (LPC, Linear Predictive Coding) käytetään projektissa useiden erilaisten äänten, kuten hydraulisten ja harvesteriäänten, tuottamiseen. Spektrivähennyksen ja NMF-menetelmän (engl. Non-negative Matrix Factorization) avulla tuotetaan häiriöttömiä kontaktiääninäytteitä kuormatraktorin simulaattoriin ja porausääniä poralaitteiston simulaattoriin. Lisäksi käytetään muita menetelmiä, kuten suodatusta, amplitudiverhokäyrän irrotusta ja huipun ilmaisua.

Avainsanat akustiikka, hydrauliikka, metsäkoneet, poraaminen, simulaattorit

ISBN (painettu) 978-952-60-5963-1	**ISBN (pdf)** 978-952-60-5964-8	
ISSN-L 1799-4896	**ISSN (painettu)** 1799-4896	**ISSN (pdf)** 1799-490X
Julkaisupaikka Helsinki	**Painopaikka** Helsinki	**Vuosi** 2014
Sivumäärä 4+102	**urn** http://urn.fi/URN:ISBN:978-952-60-5964-8	

Preface

This report documents the results of the REMES project ("Realistic Machine and Environmental Sounds to Improve Safety at Work"). This work was conducted between February and October 2014. The REMES project continued the efforts of the Audio Signal Processing Research Group of Aalto University to apply sound synthesis methods to non-musical noises, which is a much younger field than music synthesis. A large part of this report is based on Ville Mäntyniemi's Master's thesis, which contains some extra information on listening tests conducted in the REMES project. Dr. Rémi Mignot contributed significantly to the analysis and synthesis of drilling sounds in which he successfully employed the non-negative matrix factorization method. Rémi also worked hard during the final editing of this report.

The project researchers and I are grateful to the Finnish Work Environment Fund, Creanex, and Sandvik Construction for supporting this research. Special thanks go to Mr. Sami Oksanen, whose role during the planning of the REMES project was essential.

Espoo, Thursday 23rd October 2014,

Vesa Välimäki

Contents

Preface **1**

Contents **3**

Abbreviations **7**

1. Introduction **9**

2. Background and Methods **13**

2.1 Sound Synthesis 13

2.1.1 Sampling and Wavetable Synthesis 13

2.1.2 Additive and Subtractive Synthesis 14

2.2 Linear Predictive Coding 14

2.2.1 Basic Principle 14

2.2.2 Linear Prediction 15

2.2.3 Autocorrelation Method 16

2.2.4 LPC Synthesis 17

2.3 Filtering 18

2.3.1 FIR-filter Design 18

2.3.2 Sliding Average Filter 19

2.3.3 Moog Filter 20

2.3.4 State Variable Filter 21

2.4 Spectral Subtraction 22

2.4.1 Basic Principle 22

2.4.2 Reducing Spectral Error 23

2.5 Peak Detection 24

3. Hydraulic Sounds **27**
3.1 Machine Hydraulics . . . 27
3.2 Synthesis Method . . . 27
3.3 Synthesized Hydraulic Sounds . . . 28
3.3.1 Basic Hydraulic Sound . . . 28
3.3.2 Fading Hydraulic Sound . . . 28
3.3.3 High Frequency Hydraulic Sound . . . 30
3.3.4 High Frequency Variable Hydraulic Sound . . . 32
3.3.5 Hydraulic Piston Contact Sound . . . 33

4. Harvester and Forwarder Sounds **37**
4.1 The Machines . . . 37
4.2 Feeding Sound . . . 39
4.3 Delimbing Sound . . . 40
4.4 Contact Sounds . . . 45
4.4.1 Logs Hitting the Screen . . . 46
4.4.2 Logs Hitting the Bunks . . . 46
4.4.3 Grapple Opening . . . 46
4.5 Hydraulic Pump Sound . . . 47
4.6 Load Brake Sound . . . 48

5. Drilling Sound Synthesis **53**
5.1 Background Noise Removal . . . 56
5.1.1 Non-negative Matrix Factorization . . . 56
5.1.2 Quasi-Stationary Noise Removal Using NMF . . . 58
5.1.3 Noiseless Drilling Sound Reconstruction . . . 59
5.2 Drilling Sound Analysis/Synthesis . . . 60
5.2.1 Time-Envelope Estimation . . . 63
5.2.2 First estimation . . . 65
5.2.3 Second Click Detection . . . 66
5.2.4 Second Estimation of the Envelope Parameters . . . 68
5.2.5 Last Click Detection . . . 69
5.2.6 Click Extraction . . . 69
5.2.7 Realistic Synthesis . . . 70
5.3 Software Package . . . 72

5.3.1 Annotation . . . 72
5.3.2 Analyzer . . . 73
5.3.3 Test and Parameter Refinement . . . 74
5.3.4 Drilling Data Exportation . . . 75
5.3.5 Real-Time Synthesis Library . . . 77
5.3.6 Demonstration Application . . . 84

6. Evaluation **89**
6.1 Listening Test at Creanex . . . 89
6.1.1 Procedure . . . 89
6.1.2 Results and Improvements . . . 90
6.1.3 Feeding . . . 90
6.1.4 Feeding and Delimbing . . . 91
6.1.5 Hydraulic Sounds . . . 91
6.1.6 Hydraulic Pump . . . 91
6.1.7 Forwarder Contact Sounds . . . 92
6.1.8 Load Brake Sound . . . 92
6.2 Listening Test at Sandvik . . . 93
6.2.1 Procedure . . . 93
6.2.2 Normal Drilling . . . 93
6.2.3 Under Feeding . . . 95
6.2.4 Over Feeding . . . 95
6.2.5 Deep . . . 96
6.2.6 Bending . . . 96
6.2.7 Rattling (Closed) . . . 97
6.2.8 Rattling (Open) . . . 97
6.2.9 General Comments, Conclusion and Improvements . . . 98

7. Conclusions and Future Work **99**

Bibliography **101**

Abbreviations

API	Application Programming Interface
AR	Autoreg-Ressive
ARMA	Autoreg-Ressive Moving Average
CPU	Central Processing Unit
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
GUI	Graphical User Interface
LP	Linear Prediction
LPC	Linear Predictive Coding
NMF	Non-negative Matrix Factorization
MA	Moving Average
MSE	Mean Square Error
RPM	Revolutions Per Minutes
SNR	Signal-to-Noise Ratio
STFT	Short-Time Fourier Transform

1. Introduction

Training simulators are rapidly becoming essential tools in the training of new machine operators. These simulators allow for a completely safe training environment, where future operators can efficiently learn the essential operations, tasks, and hazards. Ideally, a simulator-trained operator will be able to correctly and safely operate an actual machine without prior real-life experience. To reach this goal, the simulator environment must be as realistic as possible to minimize the transition between the simulator and the machine, cf. e.g. [1].

Although modern simulators are already visually and operationally extremely realistic and advanced (Fig. 1.1), the sound environment is still in need of considerable improvement. Sounds play a crucial role in operating several machines: they give the operator valuable information about the state of the machine, they inform of other machines in the area, and they give audible cues which help in operating the machines. Essentially, with a realistic sound environment the operators can be trained to distinguish defects in the machine, avoid hazardous situations, and operate the machines with the help of both visual and aural aids. This method of training will allow beginners to learn these aforementioned skills in a hazard-free environment instead of an actual worksite with real machines and people. Although simulator based sound synthesis work does exist relating to flight simulators (cf. e.g. [2] or [3]), little to no research can be found regarding working machine simulators, excluding drilling sounds [4].

This project has focused on improving the sound environment of three different types of working machine simulators: a forestry harvester and forwarder simulator, a drill rig simulator, and a truck mounted hydraulic platform simulator. The actual machines in question are presented in Fig. 1.2. The method of improving the sounds in these simulators will be to improve the existing sounds and to add completely new

Figure 1.1. Working machine simulators: forest machine simulator (left) and drill rig simulator (right). Pictures retrieved from www.creanex.com.

sounds which are currently missing, but necessary for a more realistic soundscape. The current sound synthesis method used in the simulators is sampling synthesis [5], with some basic processing such as pitch shifting and amplitude modulation. Sampling synthesis will remain as the basis for the whole sound system, i.e. sound files will be played back in the simulator when they are required, but other methods of sound synthesis will be implemented to improve the authenticity of these sounds and to add new sounds. Subtractive synthesis [5] will play a major role in synthesizing several different sounds in this project, as one common method falling under this category is Linear Predictive Coding (LPC) (cf. e.g. [6] or [7]), which is a vital tool widely utilized in this project. Also the Non-negative Matrix Factorization is used for drilling sounds, to remove the background noise.

The structure of this report is as follows. Section 2 begins with an introduction to sound synthesis and the theory behind the methods used to synthesize the sounds created in this project. Section 3 explains the synthesis process behind creating hydraulic sounds, which were missing from the simulators completely. Hydraulic sounds can be very prominent in the actual machines and they exist in both the forestry machines and drill rigs, making them a vital part of this project. Section 4 deals with sound synthesis related to the forest machine simulators. Important sounds synthesized in this section include, feeding, delimbing, and contact sounds. Section 5 focuses on drilling sound synthesis, the most essential sound in a drill rig. Although synthesis work based on physical models can be found related to drilling sounds (cf. e.g. [4], or [8]), this work will focus on improving the existing sample based drilling sound and not utilize the more advanced physical models. Section 6

Figure 1.2. Top left: forest harvester, top right: forest forwarder, bottom left: truck mounted hydraulic platform, bottom right: drill rig. Pictures retrieved from www.ponsse.com, www.miningandconstruction.sandvik.com, and www.bronto.fi.

presents the results of two listening tests regarding the hydraulic, forest simulator sounds, and drilling sounds. The listening tests were carried out to obtain valuable feedback and comments on the synthesized sounds to help further improve the sound environment. Finally, Section 7 summarizes the project and discusses future works.

2. Background and Methods

This section provides an introduction to sound synthesis and the theory behind the methods used, they will be frequently mentioned in next sections 3 and 4. Nevertheless, because of the different approach used for the drilling sounds, some special methods has been especially designed for them, and are described in Sec. 5.

2.1 Sound Synthesis

Today sound synthesis is generally divided into four different classes: processed recordings, spectral models, physical models, and abstract algorithms [9]. This project will focus mostly on synthesis in the processed recordings and spectral model classes. Synthesis methods falling under these two categories will be explained in more detail below.

2.1.1 Sampling and Wavetable Synthesis

As explained in [5], sampling and wavetable synthesis are currently the most popular methods used in sound synthesis and they are both examples of processed recordings. In sampling synthesis, recorded sounds are simply played back with or without processing. An example of this is a digital piano, where recordings of different piano keys being played are used to synthesize a piano sound. There are several recordings for each key to simulate different attacks, which are needed to create a realistic sounding digital piano. Sampling synthesis is the main method used in the simulators to synthesize sound prior to the start of this project. Recorded machine sounds were played backed in the simulators with some simple processing, including amplitude variations and pitch shifting.

In wavetable synthesis a single period of a sound waveform is stored into a table and then repeated. For musical instruments it is common to store separate parts of the sound. For example, the attack, sustain, and release segments of the wanted sound can be stored separately and then played back when needed to create a more realistic synthesis result. Filters are typically used with wavetable synthesis to achieve variation in sound through spectral control. In addition to this, samples may be pitch shifted to a certain degree to allow for more flexibility. If a sample is pitch shifted too much, it will start to sound unnatural. Instead several different samples with different frequencies should be used and then pitch shifting should implemented between these samples to obtain the required frequency. This method is called multisampling and it can be used with sounds created in this project.

2.1.2 Additive and Subtractive Synthesis

Two examples of spectral modeling are additive and subtractive synthesis. Additive synthesis is simply the method of combining several sine waves to create a sound. In subtractive synthesis a spectrally rich waveform is filtered with a suitable filter to achieve the required sound. The excitation signal is commonly white noise or an impulse train and it is filtered with a filter which will shape the spectrum in a required way. One common subtractive synthesis method is Linear Predictive Coding (LPC), which can be used to extract the spectral shape of a signal and design a suitable filter to model this spectrum. LPC is used extensively in this project, especially in synthesizing hydraulic sounds. Linear predictive coding will be explained in more detail in the next section.

2.2 Linear Predictive Coding

2.2.1 Basic Principle

Linear predictive coding is a common technique used in audio signal processing and especially speech processing, cf. [6] and [7]. It is used to present the spectral characteristics of a signal. LPC assumes signals to present a source-filter model, where a source is excited by a linear filter, e.g. in speech processing the source is the vocal cords and the filter models the vocal tract. The LPC filter models the spectral shape

of the sound produced by the vocal tract and a white noise excitation can be used to synthesize a speech signal. This is called LPC synthesis and it is widely utilized in this project, although not for its most common application, speech processing, but instead for machine sound synthesis. LPC synthesis will be explained later in more detail.

A basic signal can be written as

$$S(z) = U(z)H(z), \tag{2.1}$$

where z is the Z-transform variable and $U(z)$ is the Z-transform of the excitation signal u_n, which is filtered by the transfer function of the spectral shaping filter $H(z)$, which is an estimate of the spectral shape of the signal $s(n)$. For continuous spectra, the excitation $U(z)$ is assumed to have a flat magnitude spectrum. In this project white noise is almost exclusively used as the excitation signal.

There are three different cases of the LPC-model: the all-pole model (autoregressive = AR model), the all-zero model (moving average = MA model), and the pole-zero model (autoregressive moving average = ARMA model), cf. [7]. The all-pole model is the most widely used and will be used exclusively in this project. In the all-pole model, the signal $s(n)$ is a linear combination of past values with an input u_n and gain G

$$s(n) = \sum_{k=1}^{p} a_k s(n-k) + G u_n. \tag{2.2}$$

The transfer function of the all-pole model is given as

$$H(z) = \frac{G}{A(z)} = \frac{G}{1 - \sum\limits_{k=1}^{p} a_k z^{-k}}. \tag{2.3}$$

Linear prediction must be used to determine the predictor coefficients a_k of the FIR filter $A(z)$, which is commonly known as the inverse filter.

2.2.2 Linear Prediction

Linear prediction is the method of predicting future samples by forming estimates from linear combinations of previous samples. The linear predictor is shown in Eq.(2.4), where $\hat{s}(n)$ is the estimate, $s(n-k)$ the previous sample, and a_k the predictor coefficients. The amount of previous samples used for the linear prediction is presented by p, which is the order of the inverse filter. The idea behind linear prediction is to calculate the predictor coefficients a_k, so that the difference between the

estimate $\hat{s}(n)$ and the actual sample $s(n)$ is as small as possible [5].

$$\hat{s}(n) = \sum_{k=1}^{p} a_k s(n-k) \tag{2.4}$$

The error of the estimate is the difference between the actual sample and the estimate, which is also known as the residual

$$e(n) = s(n) - \hat{s}(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k). \tag{2.5}$$

In linear prediction, the predictor coefficients are obtained by minimizing the Mean Square Error (MSE)

$$E_{MSE} = \sum_{n} e(n)^2 = \sum_{n} \left(s(n) - \sum_{k=1}^{p} a_k s(n-k) \right)^2 \tag{2.6}$$

by taking the derivative in relation to the predictor coefficients and setting it to zero

$$\frac{\delta E}{\delta a_i} = 0, \qquad 1 \leq i \leq p \tag{2.7}$$

Then we arrive at

$$\sum_{k=1}^{p} a_k \sum_{n} s(n-k)s(n-i) = \sum_{n} s(n)s(n-i), \qquad 1 \leq i \leq p. \tag{2.8}$$

By expanding Eq.(2.6) and substituting Eq.(2.8), the minimum total squared error, E_p, can be obtained

$$E_p = \sum_{n} s_n^2 + \sum_{k=1}^{p} a_k \sum_{n} s_n s_{n-k}. \tag{2.9}$$

After this step, there are two common choices for calculating the predictor coefficients: the autocorrelation method and the covariance method [7]. The autocorrelation method is used in this project and is explained below.

2.2.3 Autocorrelation Method

In the autocorrelation method, the error in Eq.(2.6) is minimized within the boundaries $-\infty < n < \infty$ (in practice, windowing is applied), which reduces Eqs.(2.8) and (2.9) to

$$R(i) = \sum_{k=1}^{p} a_k R(i-k), \qquad 1 \leq i \leq p \tag{2.10}$$

and

$$E_p = R(0) - \sum_{k=1}^{p} a_k R(k). \tag{2.11}$$

The optimal LPC shown in Eq.(2.10) is presented below in matrix form

$$\begin{bmatrix} R(0) & R(1) & \dots & R(p-1) \\ R(1) & R(0) & \ddots & \vdots \\ \vdots & & \ddots & R(1) \\ R(p-1) & \dots & R(1) & R(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{bmatrix}. \tag{2.12}$$

Because the left side of the matrix equation is a Toeplitz matrix (each descending diagonal from left to right is constant), the predictor coefficients can be recursively obtained using the Levinson-Durbin Algorithm [7]

$$E_0 = R(0) \tag{2.13}$$

$$k_i = \left[R(i) - \sum_{j=1}^{i-1} a_j^{(i-1)} R(i-j) \right] \Big/ E_{i-1} \tag{2.14}$$

$$a_i^{(i)} = k_i \tag{2.15}$$

$$a_j^{(i)} = a_j^{(i-1)} - k_i a_{i-j}^{(i-1)}, \qquad 1 \leq j \leq i-1 \tag{2.16}$$

$$E_i = (1 - k_i^2) E_{i-1}. \tag{2.17}$$

Equations (2.14) - (2.16) are recursively calculated with $i = 1, 2, 3, ...p$. From this algorithm we can then obtain the optimal predictor coefficients for the inverse filter

$$A(z) = 1 - \sum_{k=1}^{p} a_k z^{-k}. \tag{2.18}$$

As presented in [7], the gain can be written as

$$G = \sqrt{E_p} = \sqrt{R(0) - \sum_{k=1}^{p} a_k R(k)} \tag{2.19}$$

2.2.4 LPC Synthesis

Linear prediction can be considered as the process of dividing a signal into two parts: the inverse filter $A(z)$ and the residual $e(n)$. The opposite process of this analysis

method is LPC synthesis, where the original signal can be obtained by filtering the residual with the inverse of $A(z)$, also known as the LPC filter

$$S(z) = E(z)\frac{1}{A(z)} = \frac{E(z)}{1 - \sum_{k=1}^{p} a_k z^{-k}}. \tag{2.20}$$

The LPC filter models the spectral shape of the signal and it can be effectively applied in sound synthesis. Instead of filtering the residual of the original signal, a suitable excitation signal is selected (usually white noise), which is then filtered with the LPC filter. The output is a synthetic sound signal with the spectral qualities of the original signal. Due to the noisy properties of the residual, replacing it with a synthesized noise signal is a perceptually valid method. By using sufficiently large filter orders p the difference between the signals may be inaudible.In addition, increasing the filter order p greatly augments the detail of the spectral envelope, which is why high filter orders ($p = 1000$) are preferred throughout this project. The LPC synthesis method is applied throughout this project.

2.3 Filtering

Filtering is an essential tool in signal processing and audio signal processing, so it is no surprise it is also widely practiced in this project. Basic FIR-filters (finite impulse response filters) are applied throughout the project, including low-pass, band-pass, band-stop, and high-pass filters [10]. In addition to these basic filters, Moog filters [11] or [12], state variable filters, and sliding average filters [13] are also utilized in the processing of certain sounds.

2.3.1 FIR-filter Design

The FIR-filters are designed using the classic window design method [14] with a Kaiser window [15]. In the window design method, the specifications of the filter are determined in the frequency domain by the wanted cutoff frequencies of the pass-band and stopband and their corresponding maximum ripple values. The filter is then designed to meet these specifications using a specific window function. If no window function is used, i.e. the window is a rectangular window, there will always be passband ripple, which can be observed in Fig. 2.1.

As seen in Fig. 2.1, passband ripple can be easily minimized by using a nonrectan-

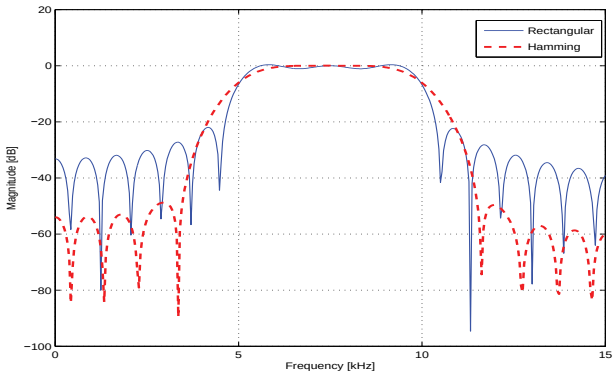

Figure 2.1. Example spectrum of a band-pass filter with a passband of $[5, 10]$ kHz using a rectangular window and a Hamming window.

gular window, e.g. Hamming, Hanning, Chebyshev, or Kaiser window. In this case, a Kaiser window will be implemented due to its flexibility in allowing the user to control the width and levels of the main and side lobes. The Kaiser window is given as

$$w(k) = \frac{I_0\left[\beta\sqrt{1-\left(\frac{k-p}{p}\right)^2}\right]}{I_0(\beta)}, \qquad k = 0, 1, 2, ..., N-1 \text{ and } p = (N-1)/2, \tag{2.21}$$

where, I_0 is the zeroth order modified Bessel function of the first kind, cf. [16]

$$I_0(z) = \sum_{k=0}^{\infty} \frac{\left(\frac{1}{4}z^2\right)^k}{(k!)^2} \tag{2.22}$$

The parameter β controls the ratio of the main lobe width and side lobe levels: as β is increased, the main lobe width increases and the side lobe amplitudes decrease. The optimal solution is to try to reduce the sidelobe amplitudes without increasing the main lobe width too much [14]. Three different Kaiser windows are shown in Fig. 2.2. Note how there is no passband ripple and how increasing β affects the main and side lobes.

2.3.2 Sliding Average Filter

The sliding average filter, also known as a moving average filter, is a very useful FIR filter. The output of the filter is the average of a number of points determined by the window size N

$$y(k) = \frac{1}{N}\sum_{i=0}^{N-1} x(k-i). \tag{2.23}$$

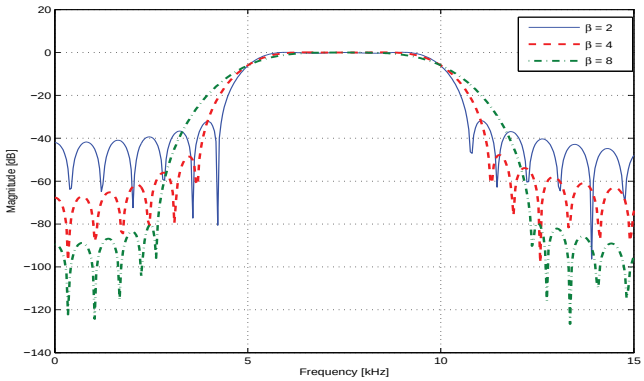

Figure 2.2. Three different Kaiser windows with different values of β with a passband of [5,10] kHz.

A whole set of data can be processed by sliding the window through the data, so that an average will be calculated for each sample from N neighboring samples. This process efficiently smooths out the data removing any short-time fluctuations and emphasizing long-term changes. In signal processing, this process can be considered low-pass filtering and it can be used to filter out noise or to extract the amplitude envelope of a signal, cf. [13]. Amplitude envelope extraction is what the sliding average filter will be used for in this project.

The amplitude envelope of a signal is extracted by first performing a full wave rectification, after which it is filtered with the sliding average filter. The window size N determines how much smoothing is applied to the signal, i.e. how accurately all the valleys and peaks are featured. Increasing the window size increases the smoothness of the amplitude envelope and less features of the original signal will be present. Figure 2.3 presents the process explained above.

2.3.3 Moog Filter

The Moog filter is a time varying filter commonly used in musical applications such as synthesizers, effects, and samplers. Originally published as a voltage-controlled filter by Robert Moog in 1965 [11], the Moog filter has since been converted into a non-linear digital implementation by Huovilainen [12], which included five nonlinear functions inside the filter sections. A more efficient single nonlinearity version of the digital Moog filter implementation is presented by Välimäki and Huovilainen in [17]. This version of the Moog filter implementation will be applied in the processing of certain sounds during the project. Essentially, the filter is able to slide the cutoff

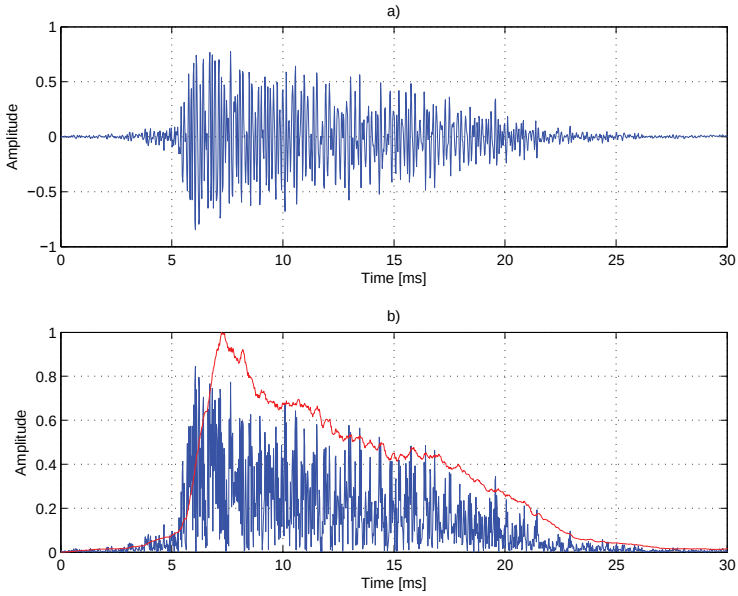

Figure 2.3. a) Original signal and b) full-wave rectified signal with amplitude envelope in red.

frequency of a low-pass filter with a resonance from one frequency to another (Fig. 2.4). This feature is excellent for creating a sweeping-type sound effect, where the frequency characteristics of a sound change in relation to time.

2.3.4 State Variable Filter

The state variable filter is a filter which provides three different outputs: low-pass, high-pass, and band-pass. In addition to this, it allows for the independent control of the cutoff frequency and damping factor. These features prove the filter suitable for many musical applications. The difference equations of the three different outputs are given in [18] as

$$
\begin{aligned}
y_l(n) &= F_1 y_b(n) + y_l(n-1) \\
y_b(n) &= F_1 y_h(n) + y_b(n-1) \\
y_h(n) &= x(n) - y_l(n-1) - Q_1 y_b(n-1),
\end{aligned}
$$

where y_l is the low-pass output, y_b the band-pass output, and y_h the high-pass out-

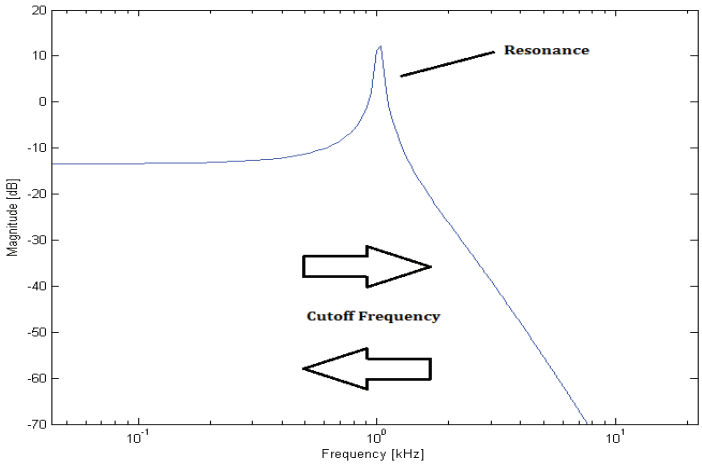

Figure 2.4. Variable cutoff frequency of the lowpass-type Moog filter.

put. The tuning coefficients F_1 and Q_1 are given as

$$F_1 = 2\sin\left(\pi f_c / f_s\right) \tag{2.24}$$

$$Q_1 = 2\zeta. \tag{2.25}$$

where, f_c and ζ are the tuning parameters and f_s is the sampling frequency. In this project the filter will be used as a variable bandpass filter, which can also be used to create a wah-wah effect commonly used in guitar effects.

2.4 Spectral Subtraction

2.4.1 Basic Principle

Spectral subtraction is a method used for removing noise from a signal, cf. [19]. It is an invaluable tool used in this project enabling the use of high quality sampling synthesis. The idea behind spectral subtraction is to subtract the magnitude spectrum of the noise from the noisy signal magnitude spectrum, leaving behind the spectrum of the clean part of the signal. The noisy signal magnitude spectrum is calculated from a segment of the signal with only noise present. The spectrum of the obtained clean

signal can than be converted back to the time domain to generate a noiseless audio signal. The steps behind this method will be explained in more detail below.

The noisy input signal is analyzed by taking half-overlapped windowed segments of the signal. The noisy signal in the time domain is given as the sum of the clean signal $s(k)$ and the noise $n(k)$

$$x(k) = s(k) + n(k). \tag{2.26}$$

Converting to the frequency domain using the Fourier transform gives

$$X(e^{j\omega}) = S(e^{j\omega}) + N(e^{j\omega}), \tag{2.27}$$

where j is the imaginary unit and w angular frequency defined as

$$\omega = 2\pi f \tag{2.28}$$

In practice, Short-Time Fourier Transform (STFT) is used to calculate the magnitude spectra of sections of the signal as it changes over time, cf. [20].

The spectral subtraction estimator

$$\hat{S}(e^{j\omega}) = \left[|X(e^{j\omega})| - \mu(e^{j\omega})\right] e^{j\theta_x(e^{j\omega})} \tag{2.29}$$

is obtained by replacing the magnitude $|N(e^{j\omega})|$ of $N(e^{j\omega})$ with its average value $\mu(e^{j\omega})$ (taken from only-noise part of signal) and the phase $\theta_N(e^{j\omega})$ of $N(e^{j\omega})$ with the phase $\theta_x(e^{j\omega})$ of $X(e^{j\omega})$. This estimate causes a spectral error given by

$$\epsilon(e^{j\omega}) = \hat{S}(e^{j\omega}) - S(e^{j\omega}) = N(e^{j\omega}) - \mu(e^{j\omega})e^{j\theta_x}. \tag{2.30}$$

2.4.2 Reducing Spectral Error

This spectral error causes unwanted audible changes in the signal and these effects should be reduced in some manner. There are four different methods available for reducing the effects of this error: magnitude averaging, half-wave rectification, residual noise reduction, and additional signal attenuation during noise-only segments. The spectral subtraction algorithm in this project will implement magnitude averaging and residual noise reduction.

Magnitude averaging uses averaging of spectral magnitudes to reduce the spectral error. $|X(e^{j\omega})|$ is replaced with $\overline{|X(e^{j\omega})|}$ where

$$\overline{|X(e^{j\omega})|} = \frac{1}{M} \sum_{i=0}^{M-1} |X_i(e^{j\omega})| \tag{2.31}$$

and $X_i(e^{j\omega})$ is the ith time-windowed transform of $x(k)$. This gives

$$S_A(e^{j\omega}) = \left[\overline{|X(e^{j\omega})|} - \mu(e^{j\omega})\right] e^{j\theta_x(e^{j\omega})} \tag{2.32}$$

modifying the spectral error

$$\epsilon(e^{j\omega}) = S_A(e^{j\omega}) - S(e^{j\omega}) \cong \overline{|N|} - \mu. \tag{2.33}$$

Residual noise can be observed as narrow bands of magnitude spikes randomly placed. Transformed back into the time-domain, these random magnitude spikes will cause unwanted sound effects and they should be removed. Residual noise is reduced by

$$|\hat{S}_i(e^{j\omega})| = |\hat{S}_i(e^{j\omega})|, \text{ for } |\hat{S}_i(e^{j\omega})| \geq \max|N_R(e^{j\omega})|$$
$$|\hat{S}_i(e^{j\omega})| = \min\left\{|\hat{S}_j(e^{j\omega})|j = i-1, i, i+1\right\}, \text{for } |\hat{S}_i(e^{j\omega})| < \max|N_R(e^{j\omega})|$$

where $\max|N_R(e^{j\omega})|$ is the maximum value of the noise residual measured during the noise-only segment of the signal. The maximum value of the noise residual is measured using STFT over several time frames.

After the modified magnitude spectrum is obtained, the signal is then coverted back to the time-domain using inverse FFT with the overlap-add method to form the new clean output signal, cf. [19].

2.5 Peak Detection

Peak detection is the process of locating the local maxima or "peaks" of a signal. A value in a signal is considered to be a peak when its amplitude is higher than that of its neighbors. Peak detection is an important tool in signal processing, as the peaks of a signal are often of interest [21]. In this project, a peak detection algorithm is used which compares each signal value to its neighbors and concludes whether it is a peak according to the following parameters:

- Minimum peak height: the minimum amplitude required to be considered a peak
- Minimum peak separation: the minimum distance between peaks, any peaks closer than the minimum distance to another peak will not be considered peaks

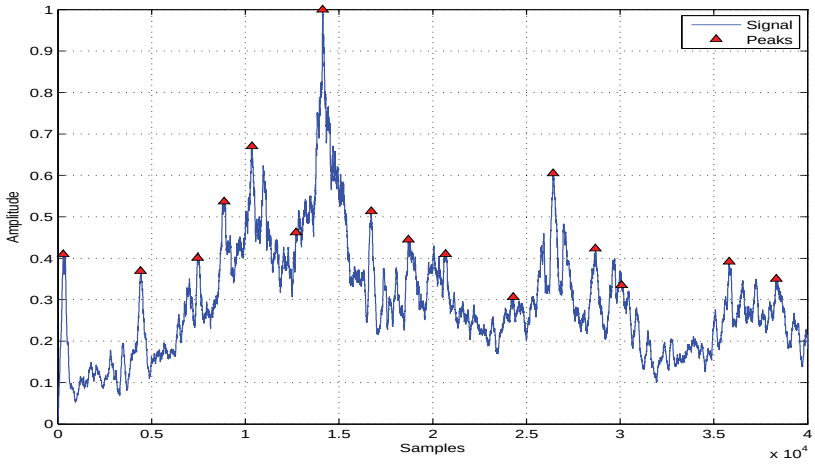

Figure 2.5. Peak detection used on a signal.

– Minimum height difference: the minimum difference in amplitude required between neighboring values to be considered a peak.

Often, it is required to perform filtering on a signal before peak detection can be a viable option. Filtering is used to remove noise, as a noisy signal will easily lead to false detection of peaks. A signal should not be smoothed out too greatly with filtering though, as this can cause important and valid peaks to disappear [21]. An example of peak detection is shown in Fig. 2.5.

3. Hydraulic Sounds

3.1 Machine Hydraulics

One main component missing from the simulators was the sound of hydraulics in the machines. Hydraulics are used in operating the boom and many other features of the drill rig, harvester and forwarder and also to lift up the carriage on a truck-mounted platform. As these sounds can be quite dominant in the actual machines, it was required to come up with a way to synthesize these sounds for the simulators. However, it should be noted that in the more modern drill rigs hydraulic sounds are quite inaudible as the cabin is heavily sound proofed.

3.2 Synthesis Method

Linear predictive coding (LPC) was chosen as the most suitable method for synthesizing hydraulic sounds and the results were excellent. Using this method, synthesized WAV files were created which could then be played back in the simulators in real-time. LPC, as explained in Sec. 2.2, is used to extract the spectral envelope of the original signal, which can then be used as a basis for sound synthesis. Using LPC, we can calculate a filter corresponding to the spectrum of the original hydraulic sound signal, which is then used to filter a white noise excitation resulting in a synthesized hydraulic sound. In this case, extremely high filter orders were used ($p = 1000$), as there was no need for real-time processing and these values achieved exceptional results. The lengths of the original hydraulic sound samples, from which the spectral envelopes were extracted, ranged from 100 ms to 150 ms. An advantage to using this

method is the possibility of synthesizing sound signals of arbitrary length, i.e. the length of the white noise excitation determines the length of the synthesized signal. This allows for the possibility of creating longer sounds, which can prove difficult if only sample based synthesis is used due to the periodic sound caused by looping the same short sample. In addition, by synthesizing several different samples with different white noise excitations, the signals will not cause a periodic sound even when looped.

3.3 Synthesized Hydraulic Sounds

This section will cover the different types of hydraulic sounds that were synthesized. All the original recording samples used for the basis of the synthesis are taken from a longer reference recording. This main file is a minute long recording of the hydraulic sounds in a drill rig presenting several different sound events possible in a hydraulic system.

3.3.1 Basic Hydraulic Sound

The most basic and prominent hydraulic sound is the basic "hiss" sound, which can be seen in Fig. 3.2 below. The LPC filter is calculated from a 100 ms portion of the original hiss sound and then a white noise excitation is filtered with it. Figures 3.1 and 3.2 present the different stages of the synthesis of this sound: the spectral envelope of the original signal calculated using LPC and the white noise excitation signal in Fig. 3.1 and the time and frequency domain signals of the original and synthetic signal in Fig. 3.2.

3.3.2 Fading Hydraulic Sound

The fading hydraulic sound is a sound which occurs as the hydraulic fluids stop flowing within the system. This sound was synthesized first by using an LPC filter (calculated from a 150 ms portion of the original signal) with an order of $p = 1000$ and a white noise excitation to achieve the basic hydraulic sound. Figure 3.3 presents the spectrum of the original signal, the LP-spectrum, the white noise excitation, and the spectrum of the synthesized signal.

To simulate the sound of the hydraulic fluids stopping their flow, a Moog filter,

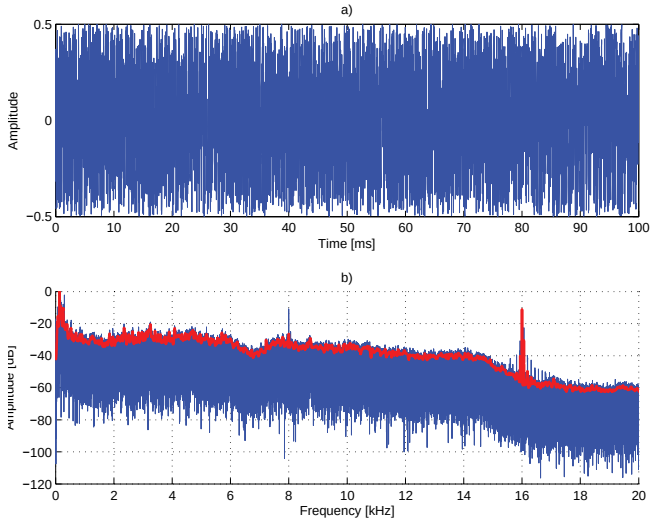

Figure 3.1. a) White noise excitation signal and b) spectrum of original signal in blue and LP spectrum in red.

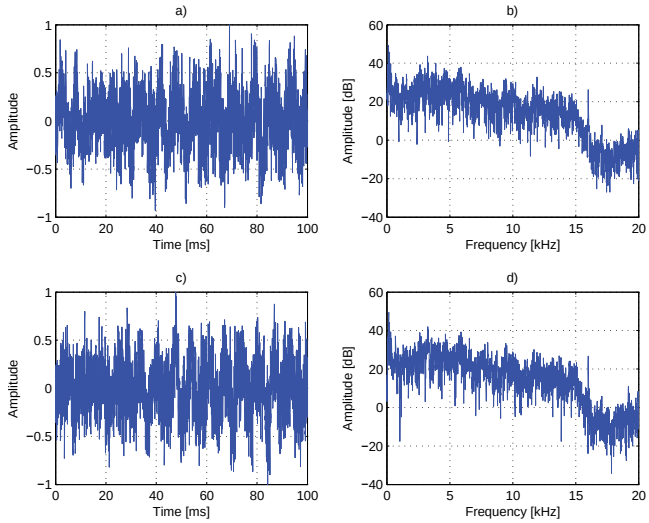

Figure 3.2. Time domain presentations and spectra of the a)b) original signal and c)d) synthetic signal.

explained in Sec. 2.3.3, was used to sweep a low-pass filter to a cutoff frequency of 700 Hz with a logarithmic frequency envelope. In addition, a linear amplitude envelope was used to fade out the sound level at the end of the sample. The basic

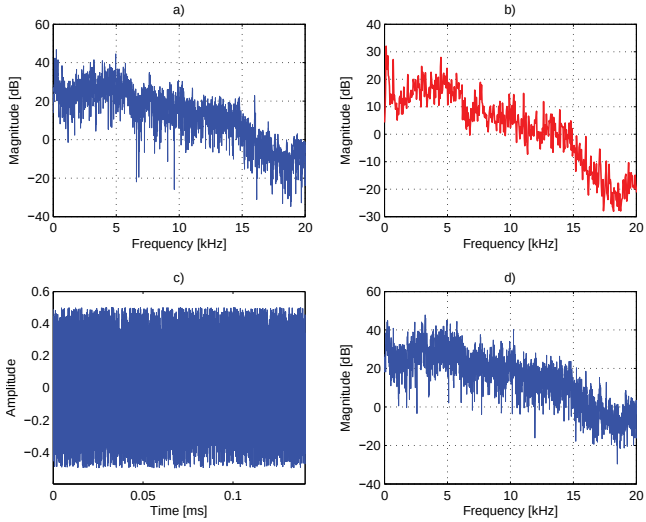

Figure 3.3. a) Spectrum of original signal, b) LP spectrum, c) white noise excitation, and d) spectrum of synthesized signal.

hydraulic sound was multiplied with the amplitude envelope and the output of the Moog filter was added to the end. The output of the Moog filter is the basic hydraulic sound filtered by the Moog filter with a low pass filter sweep from 2200 Hz to 700 Hz. The cutoff frequency envelope, the amplitude envelope, and the output of the Moog filter are presented in Fig. 3.4.

3.3.3 High Frequency Hydraulic Sound

In addition to the basic hydraulic sound, a higher frequency sound was also required. To achieve this, a high frequency hissing sound (100 ms) was first band-pass filtered from a recording of actual hydraulic sounds. The high frequency hydraulic sound could be heard in the frequency range of 4.5 to 8.5 kHz, thus a band-pass filter meeting these specifications was implemented. The band-pass filter utilized a pass band of 4.5 to 8.5 kHz and with transition bands at 4.4 to 4.5 kHz and 8.5 to 8.6 kHz. A Kaiser-window, explained in Sec. 2.3.1, was used as the windowing function for the design of the FIR digital filter, which can be seen in Fig. 3.5. The spectrum of the original and filtered signal is presented in Fig. 3.6.

After obtaining the filtered output, linear predictive coding could again be used to extract the spectral envelope of the required sound sample. LPC was used in the

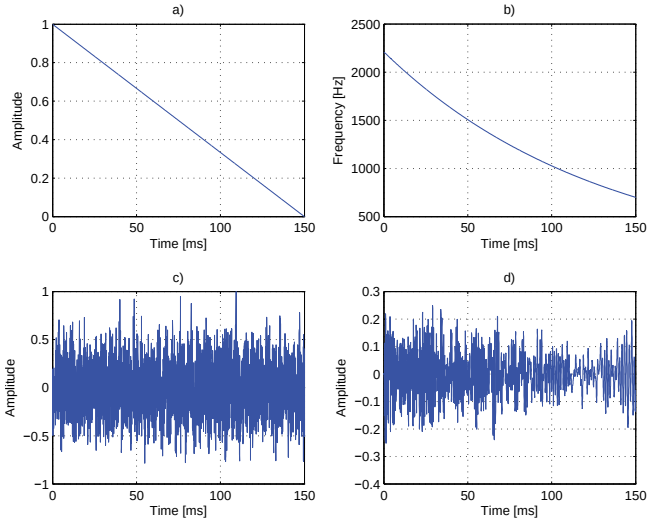

Figure 3.4. a) Amplitude envelope, b) cutoff frequency envelope, c) original hydraulic sound without fade out, and d) Moog filter output.

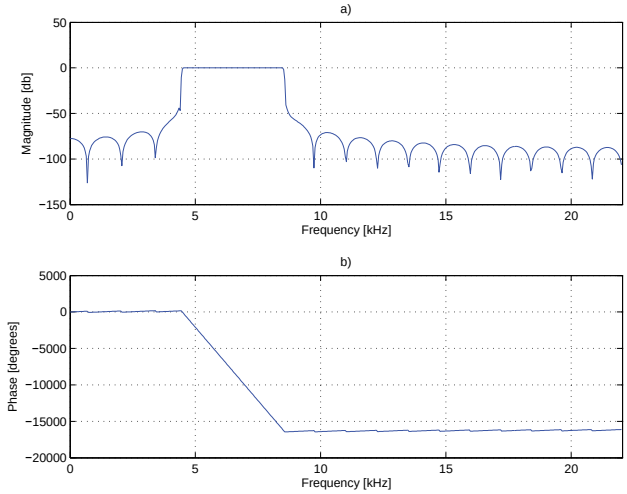

Figure 3.5. a) Frequency response and b) phase response of the band-pass filter with pass band at $[4.5, 8.5]$ kHz.

exact same manner as with the basic hydraulic sound, i.e. filter order $p = 1000$ and white noise as the excitation signal. Figure 3.7 presents the filtered spectrum, the LP spectrum, the white noise excitation, and the spectrum of the synthesized signal.

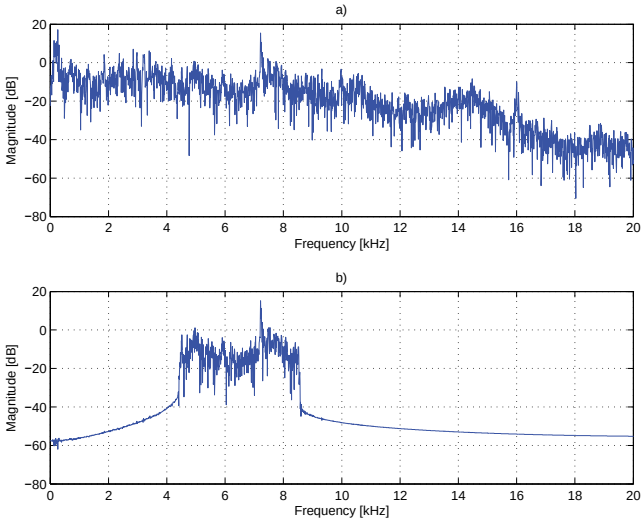

Figure 3.6. Spectrum of a) the original signal and b) band-passed signal.

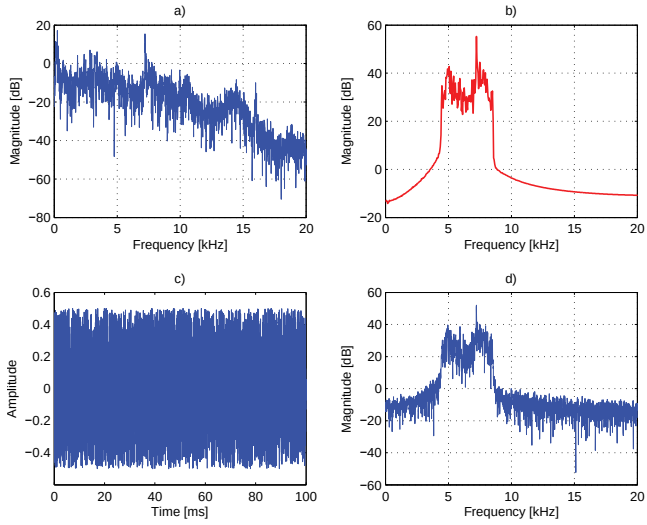

Figure 3.7. a) Spectrum of original signal, b) LP spectrum, c) white noise excitation, and d) spectrum of synthesized signal.

3.3.4 High Frequency Variable Hydraulic Sound

The high frequency variable hydraulic sound is a synthesized sound used to try and imitate the sound of the hydraulic fluids trying to squeeze through the system. The

sound is a high frequency sound which varies in frequency, causing a squealing like sound. The sound is synthesized from the synthesized output of the high frequency hydraulic sound presented in the previous section, by further filtering it with a state variable filter, which is commonly known for the "wah wah" effect used in guitar effects. This filter alters the band-pass of the filter higher and lower in frequency, causing a high frequency "wah wah" effect as seen in the spectrogram in Fig. 3.8.

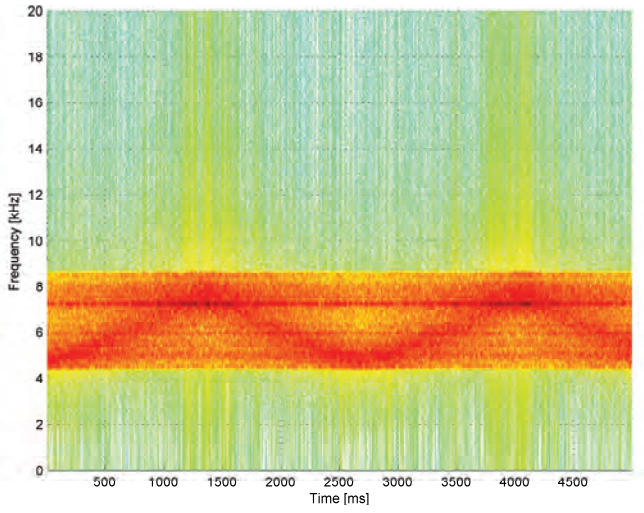

Figure 3.8. Spectrogram of the high frequency variable sound.

3.3.5 Hydraulic Piston Contact Sound

The hydraulic system also consists of contact sounds or "thump" sounds caused by the pistons of the machine. These sounds were also synthesized using linear predictive coding with a filter order of $p = 1000$, but instead of a white noise excitation, a simple impulse was utilized. Using LPC to create synthetic versions of the sounds allowed for noiseless samples, as the original signals contained some background noise. Several different LPC-filters were calculated from different contact sound samples from the recordings and the same impulse excitation was filtered with all filters. The impulse excitation and the spectra of the LPC filters can be seen in Fig. 3.9. Each synthesized sample differed from each other due to the different LPC filters, allowing for a varying pallet of contact sound samples. The time and frequency domain

presentations of these synthesized samples can be seen in Figs. 3.10, 3.11, and 3.12. The original and synthetic signals differ slightly in the time-domain, but perceptually the signals sound very similar, which can be noticed in the similarity between the original and synthetic spectra.

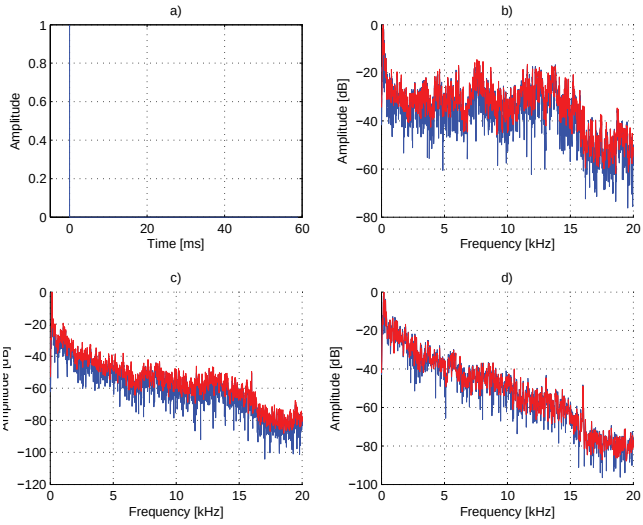

Figure 3.9. Impulse excitation in a) and the original spectrums (blue) and LP spectrums (red) of b) thump 1 c) thump 2 d) thump 3.

As can be seen in Figs. 3.10, 3.11, and 3.12 above, most of the energy in the spectra is concentrated around the lower frequencies. The spectra of the original and synthetic contact sounds are presented between 0 to 4 kHz in Fig. 3.13 below, allowing for a more detailed comparison of the spectral characteristics.

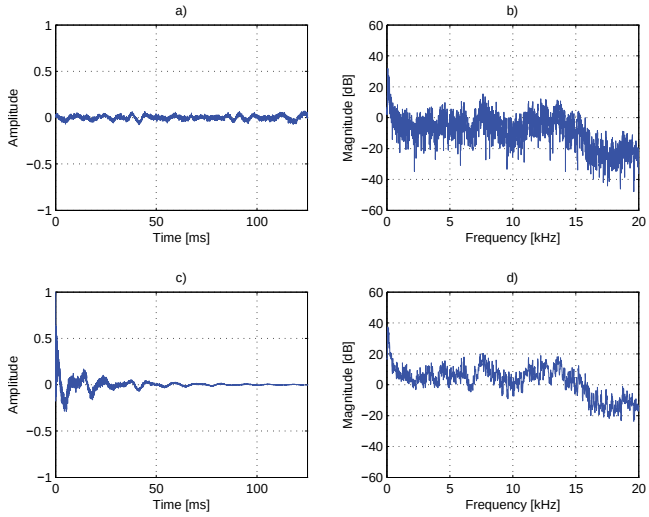

Figure 3.10. Thump 1: The original signal presented in a) the time domain and b) frequency domain. The synthesized signal presented in c) the time domain and d) frequency domain.

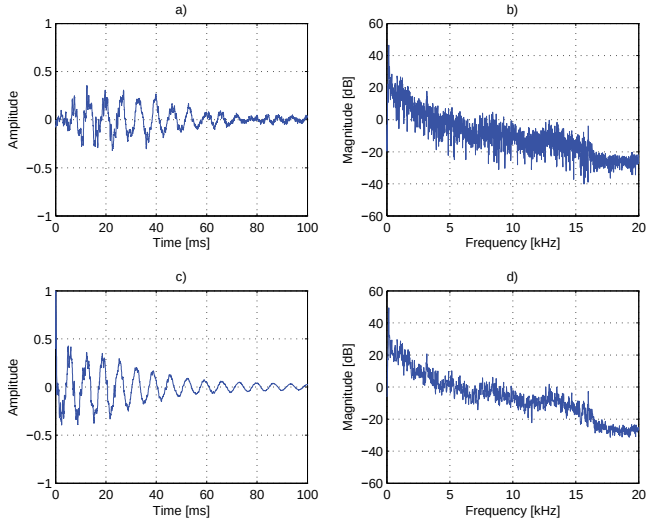

Figure 3.11. Thump 2: The original signal presented in a) the time domain and b) frequency domain. The synthesized signal presented in c) the time domain and d) frequency domain.

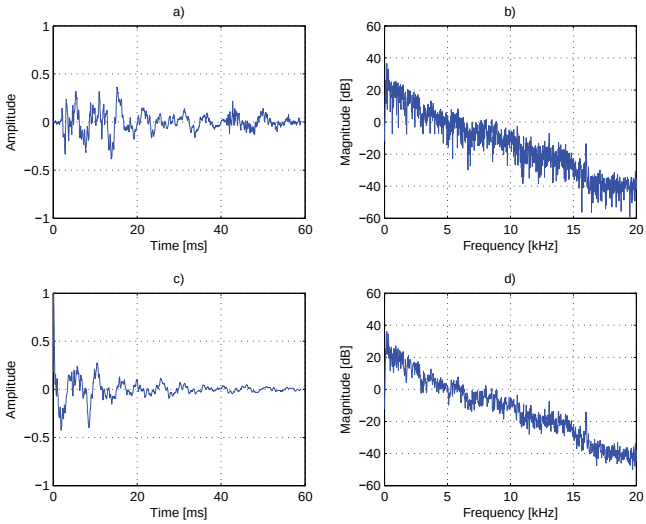

Figure 3.12. Thump 3: The original signal presented in a) the time domain and b) frequency domain. The synthesized signal presented in c) the time domain and d) frequency domain.

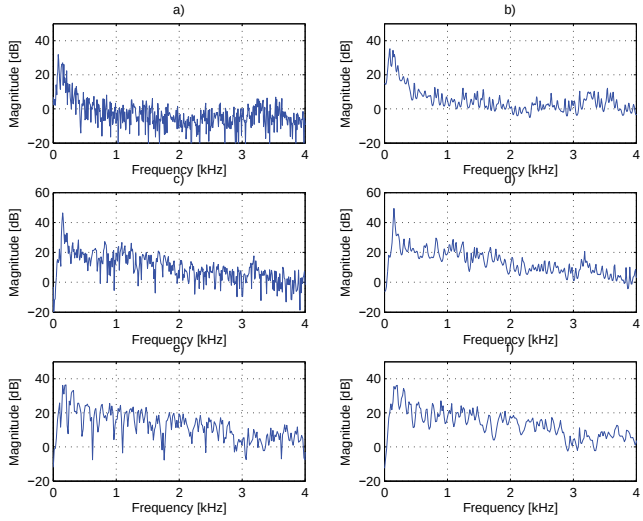

Figure 3.13. Spectra of a) thump 1 original signal, b) thump 1 synthetic signal, c) thump 2 original signal, d) thump 2 synthetic signal, e) thump 3 original signal, and f) thump 3 synthetic signal.

4. Harvester and Forwarder Sounds

One main focus of this project was the synthesis of typical sounds found in two important forest machines: the harvester and forwarder. Among the required sounds for the simulator for these machines were: feeding, delimbing, several different types of contact sounds, and hydraulic sounds which were already explained in Sec. 3.

4.1 The Machines

The harvester and forwarder are typically used together in forestry: the harvester is used to cut down trees and delimb them and the forwarder is used to transport the logs from the forest. In addition to the increased speed and effectiveness these machines bring to logging, they also provide a safer work environment as the operators are stationed safely inside a protected driving cabin, away from the dangers of falling trees and motorsaws.

A modern forest harvester can be seen in Fig. 4.1. The machine typically consists of a cabin, a diesel engine, wheels or tracks, an extendable boom, and a harvester head. The diesel engine is used to power the vehicle and the boom combined with the harvester head through a hydraulic system. The combination of a powerful diesel engine and heavy duty wheels or tracks make forest machines mobile and robust on terrains of all kinds. The boom is used to move around the harvester head, which consists of the tools required for felling and delimbing trees. The average harvester head employs a chain saw for cutting the tree, delimbing knives used for removing branches, and feed rollers for moving the tree through the harvester head. The last two parts of the harvester head are mostly of interest in this project as they are the cause of two important sound events: feeding and delimbing.

Figure 4.1. A forest harvester. Retreived from www.ponsse.com

A forest forwarder is presented in Fig. 4.2. Similar to the harvester, it also consists of a cabin, diesel engine, wheels or tracks, and an extendable boom. Instead of a harvester head, the forwarder employs a grapple at the end of the boom. The grapple is used to grab various amounts of logs and move them on and off the carriage. The grapple also contains a load brake, which is used to prevent the grapple from swinging too drastically. The carriage is used to store and transport the logs on the machine. The carriage consists of the bunks (vertical poles at the side) and the screen (the back of the carriage). The grapple, logs, bunks, screen, load brake, and hydraulics are the cause of the sounds which will be analyzed later below.

Figure 4.2. A forest forwarder. Retreived from www.ponsse.com.

4.2 Feeding Sound

Feeding is the process of moving a tree trunk through the harvester head. Moving the log allows for it to be sawed at desired positions and also to allow the delimbing knives to remove branches from the log. The feeding sound is caused by feeding a section of the log with no branches through the harvester head. The sound is the noise of the feed rollers scraping against the bark of the tree.

A video of a forest harvester at work was used as the starting point for the synthesis of a feeding sound. A segment of audio containing a feeding event was extracted from the video and analyzed and processed for the final synthesis steps. The extracted sound sample contained a fair amount of background noise, including engine and wind noise, and thus spectral subtraction, presented in Sec. 2.4, was required to achieve a cleaner sample. Even after spectral subtraction, some engine noise was still present in the sample which would adversely affect the quality of the synthesis. To attenuate the effect of the engine noise, two FIR filters (Sec. 2.3.1) were implemented:

1. A low-pass filter with a cutoff frequency at 8 kHz and a filter order of 99
2. A band-stop filter (notch filter) with a stopband at 500-600 Hz and a filter order of 1970

The low-pass filter was used to remove some of the high frequency components caused by the diesel engine of the harvester. Next, the notch filter was implemented to remove a very distinct and loud squealing sound between 500-600 Hz. The effects of filtering the original sound sample with these two FIR filters can be seen in the spectrograms shown in Fig. 4.3.

By implementing spectral subtraction and filtering with the two aforementioned FIR-filters, the sound sample was clean enough to be used as a reference sample for the synthesis. An LPC filter was calculated from the filtered sound sample with a filter order of $p = 1000$. As explained in Sec. 2.2, the LPC filter contains the spectral characteristics of the original signal. A white noise excitation signal was filtered with the LPC filter leading to a purely synthetic signal with the same spectral envelope as the original signal.

The next step in the synthesis was to correctly model the amplitude envelope of the original signal which is presented in Fig. 4.4. This was achieved with full wave rectification and a sliding average filter using the method explained in Sec. 2.3.2.

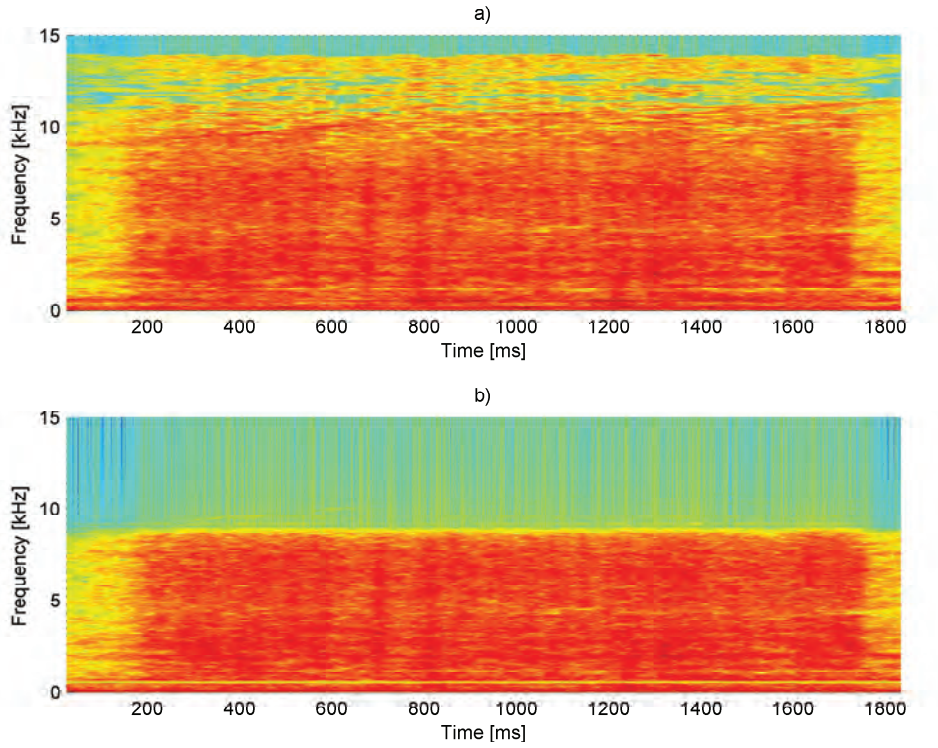

Figure 4.3. Spectrograms of a) original signal (spectral subtraction implemented) b) filtered signal.

A window size of $N = 200$ was used for the sliding average filter. The achieved amplitude envelope was then multiplied with the output of the LPC synthesis and as a result, a purely synthetic signal was obtained with the correct amplitude and frequency characteristics.

The time domain presentations and spectrograms of the original and synthetic signals can be observed in Fig. 4.5. Both signals look fairly similar in both the time domain and frequency domain, implying a successful synthetic result.

4.3 Delimbing Sound

The delimbing sound is closely related to the feeding sound presented above. The sound is caused by branches being cut by the delimbing knives in the harvester head as the log is being fed through it. So essentially, there will also always be a feeding

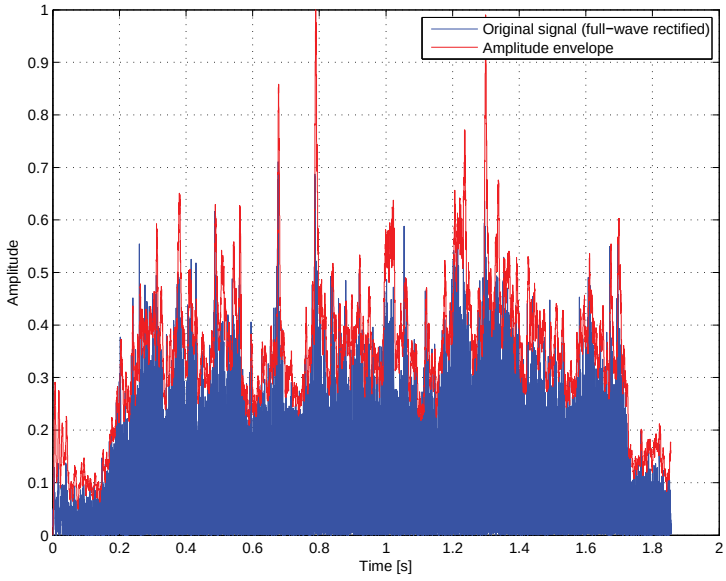

Figure 4.4. Time-domain presentation of the full-wave rectified original signal and the amplitude envelope calculated using a sliding average filter.

sound underneath the delimbing sound, as feeding is required for delimbing to take place. The approach chosen to synthesize the sound of a delimbing event was to create audio samples of branches being cut and combine them with the previously synthesized feeding sound. This allows for two separate types of sounds which can then be easily combined or played separately depending on the situation as the log is being fed through the harvester head:

1. No branches on the log $\Rightarrow$ feeding sound only
2. Branches on the log $\Rightarrow$ feeding and delimbing sound combined

The sound of branches breaking was synthesized in the same fashion as the feeding sound itself. A real-life sound sample of a branch breaking was used as a basis for the synthesis process. First, the LPC filter modeling the spectral content was calculated from this sound sample using a filter order of $p = 1000$. This filter was then used to filter a white noise excitation, resulting in a noise-like signal with the correct spectral characteristics. Next, full-wave rectification and a sliding average filter with a window size of $N = 100$ were utilized to obtain the amplitude envelope of the original

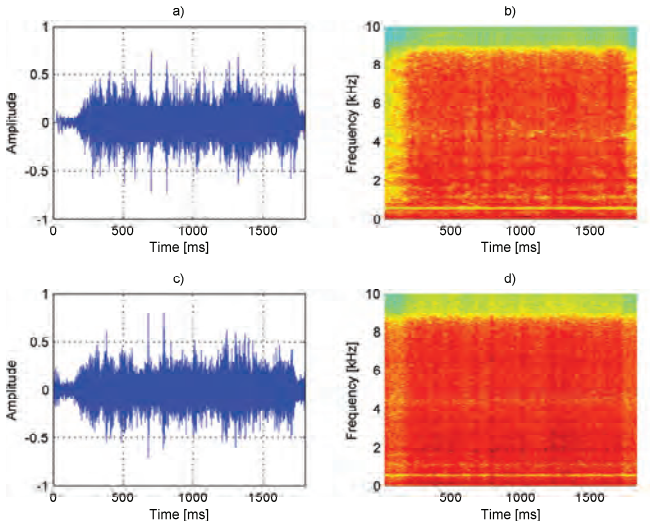

Figure 4.5. Time domain presentations and spectrograms of the a)b) original and c)d) synthetic feeding sound.

signal, cf. Fig. 4.6. Lastly, the resulting spectral envelope and amplitude envelope were multiplied to obtain a synthetic sound sample, cf. Fig. 4.7.

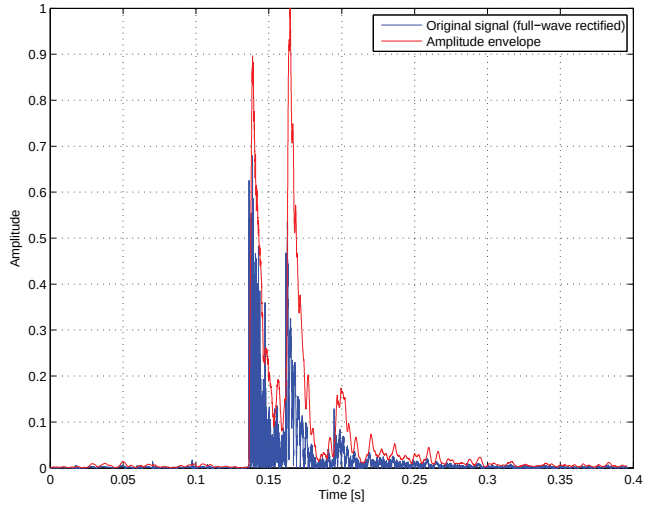

Figure 4.6. Time domain presentation of the original full-wave rectified signal of a branch breaking sound and its amplitude envelope (red).

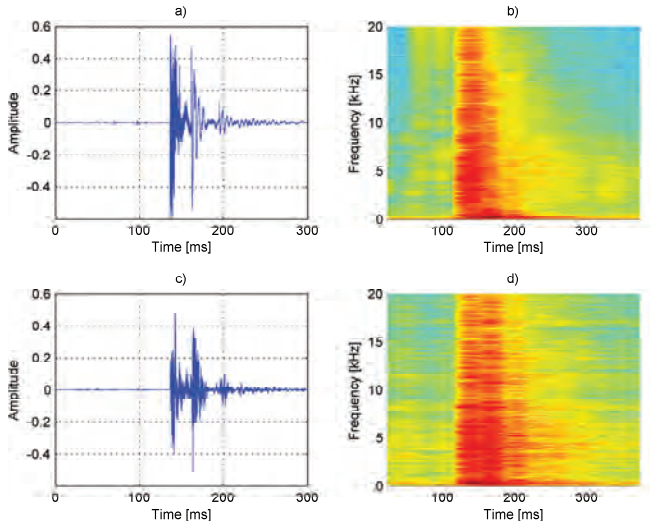

Figure 4.7. Time domain presentations and spectrograms of the a)b) original branch breaking sound and c)d) the synthetic signal created from it.

Several different real-life branch sounds were used to synthesize different versions of branch-breaking sounds adding variation and realism to the final combined sound of feeding and delimbing. Different synthetic branch-breaking sounds were chosen at random and spaced according to the peaks of the amplitude envelope of a real-life delimbing sound. The location of the peaks and their amplitudes were calculated using peak detection as explained in Sec. 2.5. A minimum peak height of 0.4 and a minimum peak separation of 1400 samples were used for the peak detection algorithm. The minimum height difference was set to zero. The detected peaks are shown in Fig. 4.8 below. Figure 4.9 presents a sample of several different delimbing sounds and the combination of the feeding and delimbing sound in the time domain.

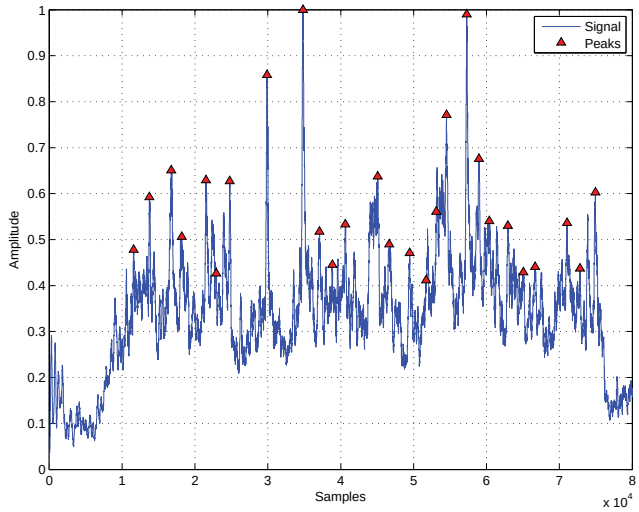

Figure 4.8. Peak detection of the amplitude envelope of a real delimbing sound.

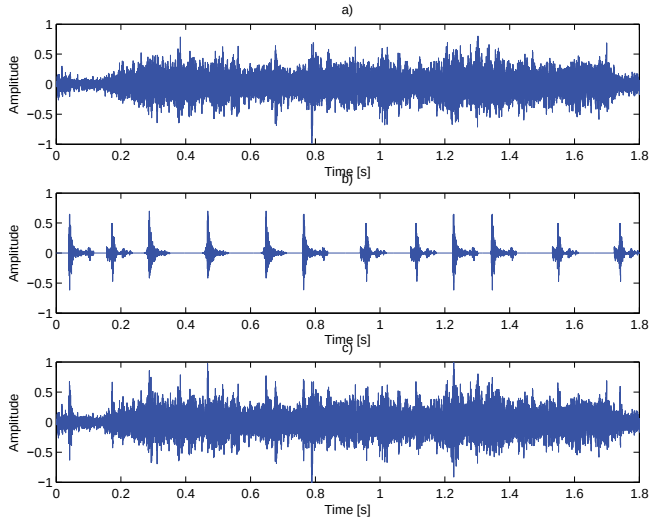

Figure 4.9. Time domain presentations of a) synthetic feeding sound only, b) delimbing samples, and c) feeding and delimbing combined.

4.4 Contact Sounds

The work related to the sounds of the forwarder simulator was mostly based on contact sounds, i.e. sounds caused by logs or parts of the machine making contact with the forwarder. Basic contact sounds consisted of: logs hitting the screen, logs hitting the bunks, and motion of the grapple opening and stopping. These aforementioned sound events are a fundamental part of the soundscape for a forest forwarder operator, and as such are required in the simulator. For example, the sound of logs hitting the screen is caused by the operator as the logs are deliberately struck against the screen to ensure they are as far back on the carriage as possible. Hearing this sound notifies the operator of the position of the logs.

The sound samples of the events described below were extracted from a video of a forest forwarder at work. The sound samples are quite short, but consist of heavy background noise caused by the diesel engine and strong gusts of wind. Spectral subtraction was used to remove the excess noise, resulting in clean sound samples, cf. Sec. 2.4. In other words, no actual synthesis was required for the contact sounds, as the processed sound samples were already of good enough quality.

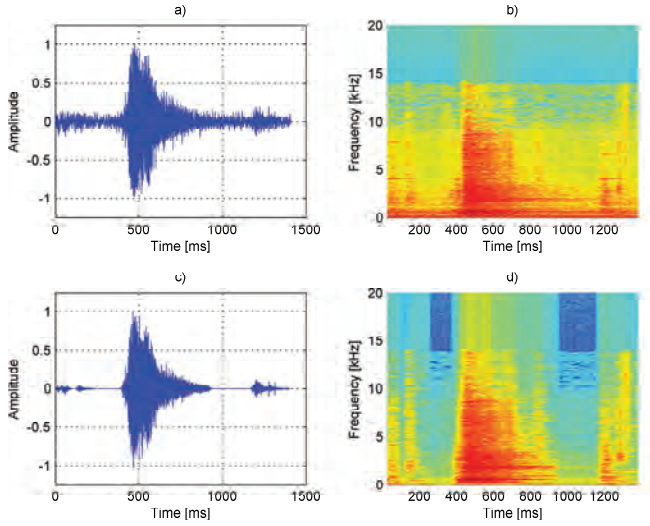

Figure 4.10. Time domain presentations and spectrograms of a)b) a noisy sample and c)d) a clean sample of the sound of logs hitting the screen.

4.4.1 Logs Hitting the Screen

As explained above, the sound of the logs hitting the screen gives vital information to the operator, and is thus a very important sound to have in the simulator. Figure 4.10 presents the original and processed sound samples of this event. As can be noticed from both the time domain and spectrogram presentations, there is significantly less noise present in the processed sound samples. This is especially evident in the spectrogram, where only the sound of interest is amplified and everything else is attenuated.

4.4.2 Logs Hitting the Bunks

The sound of the logs hitting the bunks is also a common event while using the forwarder. The bunks hold the logs on the carriage from both sides and contact with them is inevitable. The ringing-like sound of the bunks gives the operator important cues about the position of the grapple and logs, a valuable aid compared to visual observation only. Recognizing this sound already in the simulator phase of training can help future operators be well prepared for use of the actual machine. As with the screen contact sound, spectral subtraction succeeds in removing the background noise from the bunk contact sound sample, as is apparent in Fig. 4.11. Both the time domain and spectrograms of the processed signal show a significant decrease in noise.

4.4.3 Grapple Opening

The grapples is the key component in the forwarder which allows it to operate as it does. It is as vital to the forwarder as the harvester head is to the harvester. The grapple is a claw-like tool which is used to grab and release logs, so they can be lifted off the ground and onto the forwarder (and vice versa). The opening of the grapple causes a sound event which should be very familiar to experienced operators. In addition to the opening itself, an additional sound is heard as the grapple reaches its maximum opened position. Although the opening of the grapple might not be considered a contact sound, there is still some contact in the grapple as it reaches its maximum. Again, spectral subtraction is successful in removing the background noise from the noisy sample. Figure 4.12 shows the noisy and noiseless signals in the time domain and their corresponding spectrograms.

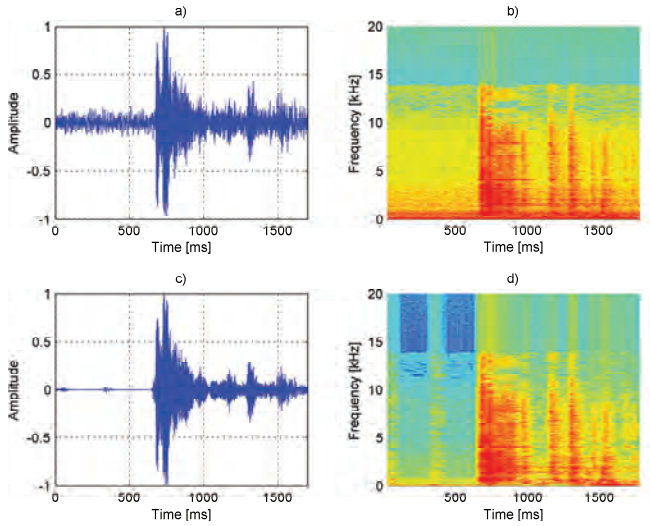

Figure 4.11. Time domain presentations and spectrograms of a)b) a noisy sample and c)d) a clean sample of the sound of logs hitting the bunks.

4.5 Hydraulic Pump Sound

The hydraulic pump sound is a high frequency whistle-like sound found in the harvester. It is quite audible right before and after sawing, feeding, or delimbing and it is caused by the hydraulics of the machine. The sound can quite easily be mistaken for the sound of the turbo which is also a similar high frequency whistle, although at a higher frequency. The hydraulic pump sound can be heard in the range of 8.6 kHz to 10.5 kHz as seen in Fig. 4.13. The sound is visible in the spectrogram as a slim darker descending line in the aforementioned frequency range.

The sound of the hydraulic pump was synthesized by first implementing a FIR band-pass filter at 8.6 - 10.5 kHz. This filtered output was used as the input for an LPC filter (order, p = 1000), which was then used to filter a white noise excitation, as was done with other hydraulic sounds explained in Sec. 3. To add some variation in frequency to the synthetic version, a variable state filter was utilized, cf. 2.3.4. The state variable filter alters the band-pass of the filter, causing the frequency to decrease and increase. Figure 4.14 presents the different steps of the synthetic process. Although in the synthetic version the behavior of the darker line does not exactly replicate that of the original signal, it still sounds perceptually very similar.

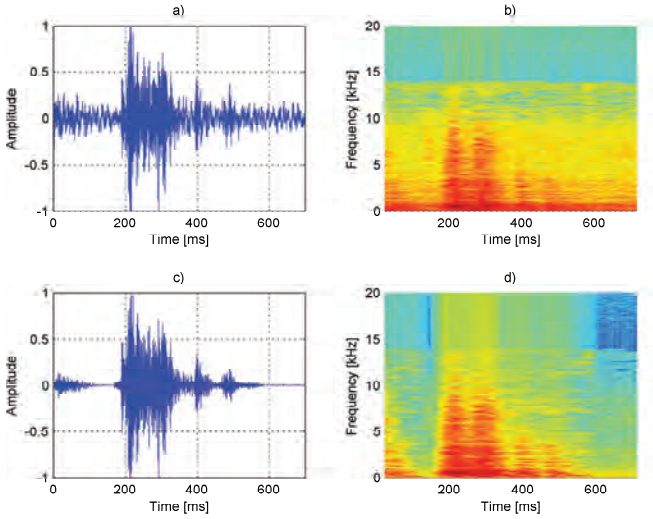

Figure 4.12. Time domain presentations and spectrograms of a)b) a noisy sample and c)d) a clean sample of the sound of logs the grapple opening and stopping.

4.6 Load Brake Sound

The load brake is a hinge-type component connecting the grapple to the boom of the forwarder. As its name implies, it operates as a brake preventing the grapple from swinging too violently as the boom is shifted around. The sound of the load brake can serve as an audible cue to the operator informing of too abrupt and reckless movements. This is why the sound of a load brake could serve as a valuable asset in a simulator, enabling the trainee to properly distinguish proper boom movements though audio cues via the load brake sound.

The sound of the load brake is practically inaudible in the forwarder videos used in previous sections of this chapter, thus a more vague approach had to be utilized for the synthesis. According to experts working with the simulator and machines, the sound is similar to a squeaky bicycle saddle, thus a sound clip of a bicycle saddle squeaking was used as the source material and basis for the synthesis.

The sound was synthesized in the same manner as the feeding sound. First, LPC (p = 1000) was used to calculate the spectral characteristics of the original sound which was then used to filter a white noise excitation. Next, the original signal was full-wave rectified in the time domain and it was filtered with a sliding average filter

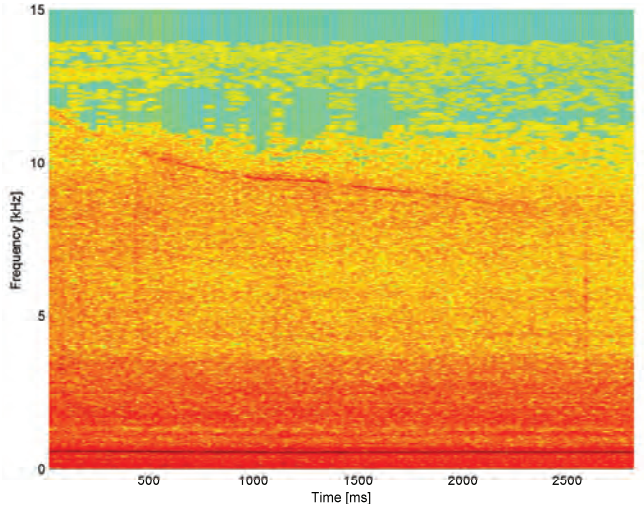

Figure 4.13. Spectrogram of the hydraulic pump sound. The sound itself is the dark red line around 8.6 kHz - 10.5 kHz.

with a window size $N = 200$, cf. Fig. 4.15.

The LPC filtered excitation signal was then multiplied with the amplitude envelope to achieve the final synthetic result which is presented in Fig. 4.16. Even though the reference sound sample is not an actual load brake sound, the spectrograms and time domain presentations clearly show how the synthetic version resembles the reference sound. By applying this method to an actual load brake sound, the result would be of much higher quality.

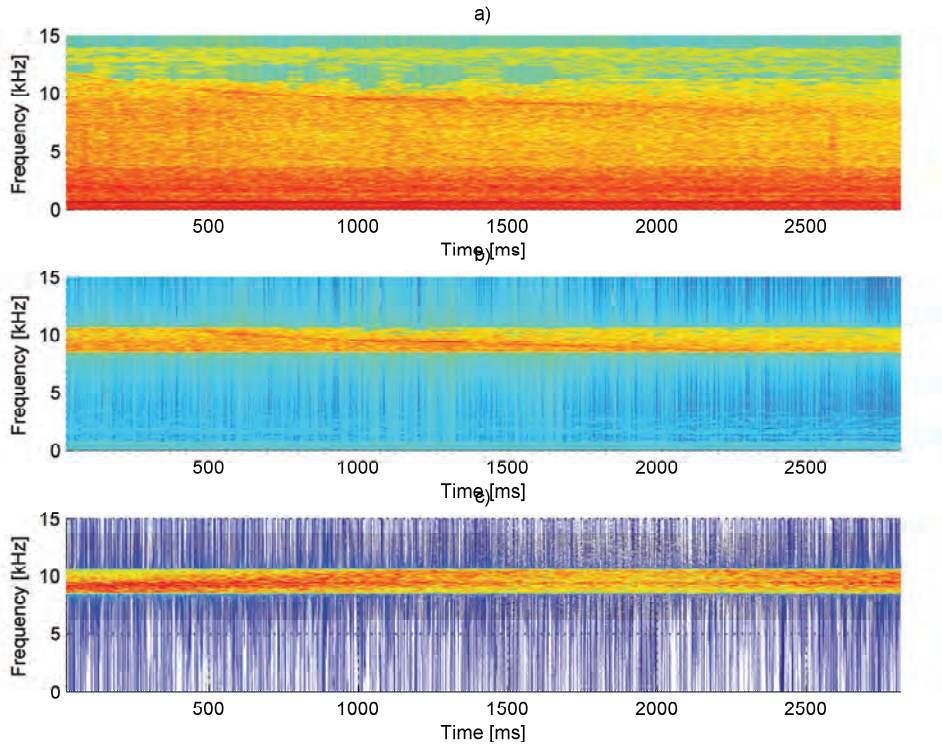

Figure 4.14. Spectrograms of the different synthesis steps. a) the original signal, b) the band-pass filtered signal, c) the synthetic signal filtered with a state variable filter.

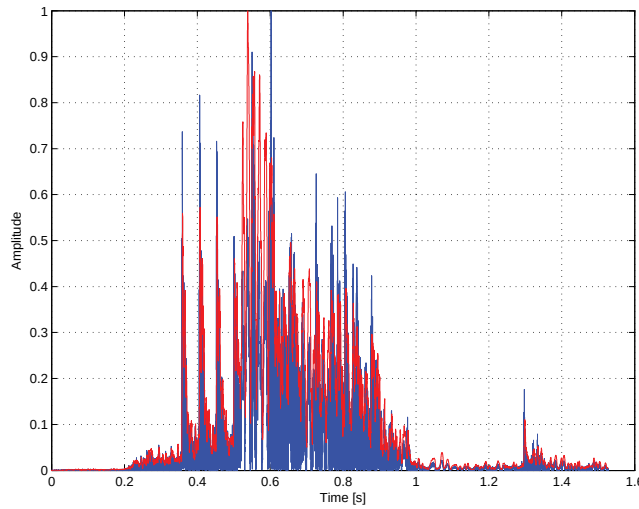

Figure 4.15. Time domain presentation of the original full-wave rectified signal and its amplitude envelope (red).

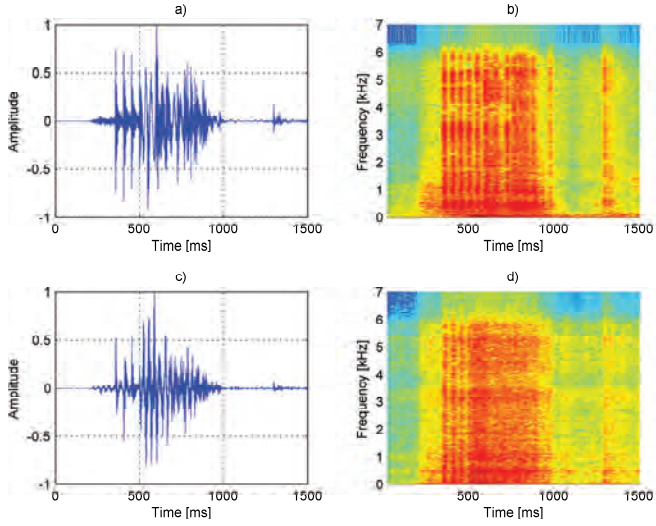

Figure 4.16. Time domain presentations and spectrograms of the a)b) original signal and c)d) synthetic signal.

5. Drilling Sound Synthesis

Drilling sounds, which are obviously an essential part of any drill rig simulator, mostly consists of the sound of the rock drill. The main component of the rock drill is the piston, which converts the hydraulic energy into mechanical energy to actually cause a drilling motion in the system. The boom of a drill rig holds several extra rods, which are connected together to form drill strings connected to the rock drill allowing it to dig deep into the ground [22]. Although this project will not go into more detail about the specifications and types of rock drills and drill rigs, it is worth mentioning that the drilling sounds analyzed in this project are from percussive hydraulic rock drills. More information about rock drills can be found in [22]. The drilling machine is illustrated in figs. 5.1 and 5.2.

The rock drill and its rods cause five main sound types typically found in drilling situations:

1. Normal drilling: the typical drilling sound which should be heard when drilling.
2. Underfeed: feeding is the method of pushing the drilling tool against the rock. In an underfeed situation, the drill is not pushed hard enough against the rock and the sound can be easily distinguished from normal drilling by experienced drill rig operators.
3. Overfeed: overfeeding is caused by pushing the drill too hard against the rock. This causes a slight variation in sound compared to normal drilling, but compared to underfeeding, it is not as easily recognizable even to experienced operators.
4. Rattling (threads closed): rattling is the sound of the rods being removed from the rod string. The rods are shook heavily to open the threads holding the

rods together. In this situation the threads are still closed causing a clearly distinguishable sound.

5. Rattling (threads open): the sound of rattling when the threads finally open is a very short but extremely loud and high frequency sound event, which is clearly different compared to rattling with the threads still closed.

Additionally, not only the sound may depend on the nature of the rock, but also the number of rods and the deep of the drilling. Note that, the rod string may bend in rare situations, and if this phenomenon happens, the operator must detect it and stop the drilling.

According to the purpose of the project, which is about the *safety* at work, the realistic sound synthesis is a requirement of this work. Indeed, if the operators are already experienced with drilling sounds before using a "real" drilling machine, they will be able to detect abnormal drilling situations and to avoid breakage of material in real situation, and eventually an accident. Then, the rig simulator is a useful tool for the training of the use of the machines, but also to learn the sounds. For that, the drilling sound synthesizer must be able to reproduce all possible drilling situations, mentioned above, with a high sound quality.

Figure 5.1. Drilling machine of Sandvik. Picture retrieved from the web site: http://construction.sandvik.com

The aim of this part of the project is the analysis of the recorded drilling sounds and the realistic synthesis, together with a very low-cost real-time computation, and

Figure 5.2. Drilling system of the machine. Picture retrieved from the web site: http://construction.sandvik.com

a flexible method which allows the change of some parameters such as the drilling frequency, number of impacts per second.

To achieve this, several recordings of drilling sounds were available, around 600 sound files, for almost all mentioned drilling situations. Most of these recordings are given with a document which provides some information: type of drilling, number of rods, ... Note that in most of the recordings, some different drilling situations are successively present.

Then, because of the very high number of drilling sounds, we decided to develop an automatic analysis of all available sounds. The derived algorithm must be as robust as possible for all drilling situations, and it must not need manual intervention.

First in sec. 5.1 the background noise of the recording is removed in order to only keep the drilling sound of interest. This noise usually corresponds to the diesel engine. Then, all the analysis steps of the procedure are described in sec. 5.2 and the simple synthesis computation is explained. Finally, in sec. 5.3, we describe the delivered software package. The developed programs allow to annotate the recordings, analyze the annotated parts, test and modify the results, export the computed data to real-time data, and compute the real-time synthesis.

5.1 Background Noise Removal

All the available recordings of drilling sounds are corrupted by an inherent background noise coming from the diesel engine or the hydraulic system. Because of the really different natures of this background noise and the drilling sound of interest, it is not possible to simultaneously analyze and synthesize them as a unique entity. Moreover, we are here only concerned by the drilling sound, first because the diesel engine, which is the most dominant background noise, is efficiently reproduced by the current version of the rig simulator, and second because the hydraulic sounds are considered in sec. 3.

To analyze and synthesize a noiseless sound which is originally noisy, some approaches are possible. First, when the sound of interest can be relevantly modeled, and if the analysis is robust to noise, it is possible to reconstruct the noiseless sound, directly from the analysis, cf. e.g. [23]. In our case, it seems really difficult to define a fine modeling of the drilling sound, which is compatible with a low-cost and flexible synthesis. Second, using an estimation of the spectrum of the background noise, it is possible in many cases to apply an adaptive spectral subtraction to remove the background noise and to extract the noiseless sound of interest, cf. e.g. [19] or sec. 2.4. Unfortunately, this assumes that the noise spectrum is constant in time, but in our case it slowly changes because of the moving engine speed, RPM (Revolutions Per Minutes), for instance.

In this work, we propose an approach which only assumes a quasi-stationary background noise, which may slowly change in time, and a given Signal-to-Noise Ratio (SNR). Not only this hypothesis is well adapted to our noise, but the drilling sound has radically different properties, which may provide an efficient extraction of drilling sounds. The analysis is based on the Non-negative Matrix Factorization which is briefly described in next section.

5.1.1 Non-negative Matrix Factorization

For audio applications, the Non-negative Matrix Factorization (NMF) is used to approximate the matrix of the magnitude spectrogram, cf. e.g. [24, 25]. Using a Short-Time Fourier Transform (STFT), cf. e.g. [18], the $(M \times N)$ spectrogram matrix $V = |X|$ represents the N successive "instantaneous" magnitude spectra of the sound, which are given by its columns with M frequencies. Using this time-frequency representa-

tion, we know the variation in time of the frequency components.

In this case, the Non-negative Matrix Factorization consists in the approximation of the $(M \times N)$ matrix V, which has only non-negative elements, into the product WH as follows:

$$V \approx WH \iff V_{m,n} \approx \sum_{k=1}^{K} W_{m,k} H_{k,n}, \tag{5.1}$$

where W is a $(M \times K)$ matrix and H is a $(K \times N)$ matrix. The dimension K of the factorization is chosen much smaller than M and N, i.e. $K \ll M$ and N.

Consequently, the Non-negative Matrix Factorization models the columns of V as a weighted sum of the columns of W. As a first conclusion, W is considered as the frequency dictionary giving the basis with size K on which the spectrogram V is decomposed. And since the kth row of the matrix H gives the time-varying weight of the kth word of W, column k, the matrix H is considered as the matrix of the time-activation. Figure 5.3 illustrates these remarks.

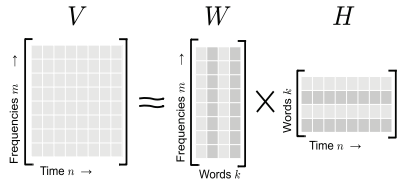

Figure 5.3. Illustration of the Non-negative Matrix Factorization.

Basically, the algorithm of this factorization consists in the minimization of a "distance" between the original spectrogram V and its approximation $\widetilde{V} = WH$. A standard choice is the Euclidean distance $D_e(V||\widetilde{V}) = \sum_{m,n}(V_{m,n} - \widetilde{V}_{m,n})^2$. Nevertheless, with NMF, it is more common to use the generalized Kullbach-Leibler divergence, cf. e.g. [26], because it is better adapted for sparse representations. This divergence is:

$$D_{kl}(V||\widetilde{V}) = \sum_{m,n} \left(V_{m,n} \log \frac{V_{m,n}}{\widetilde{V}_{m,n}} + V_{m,n} - \widetilde{V}_{m,n} \right). \tag{5.2}$$

Remark that this divergence is not symmetrical, $D_{kl}(V||\widetilde{V}) \neq D_{kl}(\widetilde{V}||V)$, that's why we speak about "divergence" and not distance.

The solving of this minimization problem is based on the iterative Newton algorithm, cf. e.g. [27]. Starting from initial non-negative matrices W and H, usually randomly chosen, the matrices W and H are successively and iteratively updated

using the following lines:

$$H \leftarrow H \otimes \left[W^T \Big(V \oslash \big(WH\big) \Big) \right] \oslash \big(W^T U\big) \tag{5.3}$$

$$W \leftarrow W \otimes \left[\Big(V \oslash \big(WH\big) \Big) H^T \right] \oslash \big(U H^T\big) \tag{5.4}$$

where $\leftarrow$ denotes the assignment statement, $.^T$ the matrix transpose, $\otimes$ the array product (element-wize product, noted $.*$ with Matlab e.g.), $\oslash$ the array division (element-wize), and U is the $(M \times N)$ matrix with 1 everywhere. Remark that there is no matrix inversion, so the computation of these lines is fast, especially using dedicated matrix computation as with the Matlab software.

We do not give any detail about the calculus which lead to this computation, we refer the interesting reader to [26] e.g. for more details.

5.1.2 Quasi-Stationary Noise Removal Using NMF

As said previously, the value $H_{k,n}$ of the activation matrix gives the contribution of the kth column of W at the time index n. Then, if the kth row of H slowly varies in time, the contribution of the kth word of W also varies slowly. To remove the background noise assuming that it is quasi-stationary, the basic idea of our approach is to constrain some rows of H to vary slowly.

Let's define K_n the size of the noise basis, the modified NMF algorithm for the background noise removal consists in adding a "smoothing" operation of the K_n first rows of H after every update eq. (5.3). As a result, this new iterative algorithm will implicitly learn the noise basis, which is stored in the corresponding K_n first columns of W. At the same time, since the properties of the drilling sound are opposite, this process also learns the sound basis, which is stored in the remaining $K - K_n$ other columns of W.

The smoothing operation of the first rows of H only consists in a linear filtering with a very low cutoff frequency f_c. For example, if $f_c = 2$ Hz, the obtained noise is authorized to vary only twice a second. In the same time, if $K_n = 2$, in principle this method tries to extract a slowly time-varying noise, by modeling it as a moving mix of two different noises.

Nevertheless, as such, this approach does not succeed to remove the background noise, because at convergence, the gain of the corresponding rows of H are too low, and the whole spectrogram V is actually modeled by the other $K - K_n$ words. To solve this problem we add a second constraint: choosing the value σ of the Signal-to-Noise

Ratio (SNR), which is the ratio between the energy of the signal, drilling sound, and the energy of the background noise, we artificially modify the gains such that this ratio is checked, and without modifying the total energy.

Let's define the energy operator $E\{X\} = \sum_{m,n} X_{m,n}^2$ for all $(M \times N)$ matrices, and let's define V_n and V_s the spectrogram of the noise and the signal respectively. Then, $V_n = W_n H_n$ and $V_s = W_s H_s$ with W_n and W_s the signal and the noise dictionaries, $[W_n, W_s] = W$; and H_n and H_s the corresponding activation matrices, $[H_n^T, H_s^T] = H^T$. The additional constraints are then:

$$E\{V_s\} + E\{V_n\} = E\{V\}, \tag{5.5}$$

$$E\{V_s\} \ / \ E\{V_n\} = \sigma. \tag{5.6}$$

With $\alpha_n = E\{V\}/(E\{V_n\}(\sigma + 1))$ and $\alpha_s = \alpha_n \sigma E\{V_n\}/E\{V_s\}$, the previous constraints are verified by assigning

$$H_n \leftarrow \alpha_n H_n \quad \text{and} \quad H_s \leftarrow \alpha_s H_s \tag{5.7}$$

after every update equations eqs. (5.3) and (5.4).

Note that additionally, the columns of W are normalized at every iteration, but this normalization does not affect the modeling because the rows of H are divided by the norms. With $\lambda_k = \sum_m W_{m,k}^2$, the ℓ_2 norms of the columns of W, and $\lambda = [\lambda_1, \lambda_2, \dots \lambda_K]$, we apply the assignments: $W \leftarrow W \operatorname{diag}(\lambda)$ and $H \leftarrow \operatorname{diag}(\lambda)^{-1} H$, after every updates (5.4).

Because it may be difficult to know the "real" SNR of the input noisy sound, it seems to be interesting to release the SNR adjustment eq. (5.7). For that, first the iterative algorithm is computed with Q_1 iterations where both constraints are applied, smoothing and gain adjustment, then Q_2 additional iterations are computed with only the smoothing. Consequently, during the Q_1 first iterations, the noise dictionary is learned by forcing the SNR, then the algorithm continues from this solution and converges to a close solution, with a possible different SNR.

5.1.3 Noiseless Drilling Sound Reconstruction

With the obtained approximation of the magnitude spectrogram $\widetilde{V} = WH$, and with the phase matrix $\Phi = \angle X$ of the original spectrogram, it is possible to reconstruct the time signal with the inverse Short-Term Fourier Transform of $\widetilde{X} = \widetilde{V} \otimes \mathrm{e}^{j\Phi}$. Then, the derivation of the estimated noiseless drilling sound, y, follows the same principle: the

inverse Short-Term Fourier Transform of $Y = V_n \otimes e^{j\Phi} = (W_n H_n) \otimes e^{j\Phi}$ is computed. This corresponds to a source separation where the drilling sound of interest and the background noise are considered as two distinct sources.

Note that the phase of the noiseless signal is not estimated with NMF, only its magnitude is approximated, that's the reason why the phase Φ of the noisy input spectrogram is used. Because this phase contains also the contribution of the background noise, this reconstruction may provide some artifacts. Nevertheless, we must notice that at the time/frequency points where the signal is dominant in magnitude, the corresponding phase is mainly produced by the signal itself. On the contrary, when the signal is dominated by the noise, the used phase is the phase of the noise, but in this case the frequency component is usually masked by the neighbor signal components with higher magnitudes. Consequently, even if the reconstruction is not perfect, it is usually satisfying from the perceptual point of view, especially with the drilling sound which are spectrally rich.

It is also possible to reconstruct the background noise using the same principle. But first, this operation has a poor quality because of a frame rate effect, and second this noise does not interest us.

For the project we used the following parameters: first the sampling rate F_s is forced by the recordings that Creanex and Sandvik gave to us, and it was around 48000 Hz. The Short-Term Fourier Transform is computed using a Hann sliding window with size 1024 samples and a fine hop size of 128 samples. The bilateral spectra with size 2048 are reduced to consider only the unilateral spectrogram corresponding the frequency range $[0, F_s/2]$, then $M = 1025$ bins. The NMF dimension is $K = 82$, and the chosen noise dimension is $K_n = 2$, using the cutoff frequency $f_c = 2$ Hz. During the $Q_1 = 100$ first iterations, the SNR has been constrained to 1 (0dB), and the algorithm continues with $Q_2 = 100$ other iterations.

5.2 Drilling Sound Analysis/Synthesis

Having the denoised drilling sounds, the new challenging task is to analyze them in order to synthesize them using a realistic and very low-cost method, but also with the possibility to change the frequency and other parameters.

The previous drilling sound synthesis of the drill rig simulator was used as the basis for the new synthesis method. The drilling sound was created by simply looping

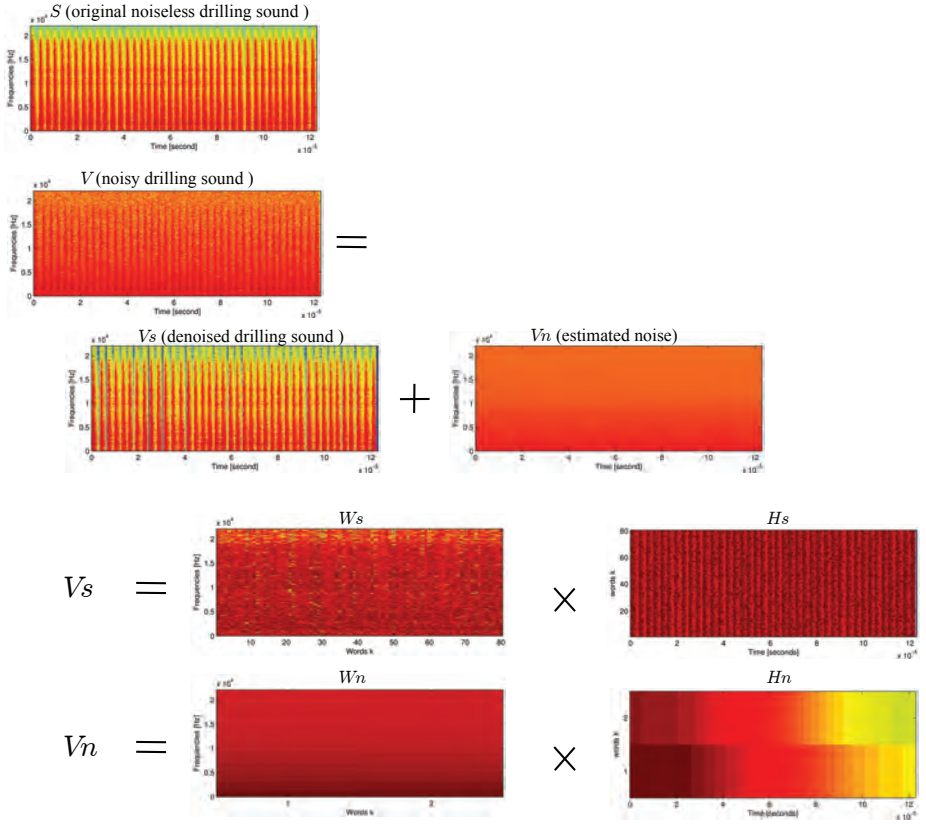

Figure 5.4. Illustration of the NMF based denoising. Here an original noiseless drilling sound is synthesized, and is given by the spectrogram S. Then a colored noise is added to produce the input noisy sound with spectrogram V. The method is computed, which provides the estimated denoised spectrogram $V_s = W_s H_s$, which must be as close to S as possible, and the resulting noise matrix $V_n = W_n H_n$. As expected, the rows of H_n vary slowly, and the rows of H_s vary faster.

a wave-file of four single synthetic strikes from a rock drill, which will henceforth be referred to as clicks. The drilling frequency was changed using pitch shifting, which reduces the time between clicks as wanted, but also modify the frequency component of the individual clicks. Optimally, as in real-life, increasing the drilling frequency should only cause the reduction of the time between clicks.

To create a synthetic drilling sound which varies the distance between clicks according to the drilling frequency, single clicks have to be first extracted from the original denoised sounds. The next step was to create a function which repeats a single click at the required frequency. By doing this, drilling sound samples of any given frequency could be correctly synthesized by just changing the distance between single click sounds; and to add variance in the drilling sounds, the single clicks are chosen at random each time from all the possible click samples.

Nevertheless, the clicks are different according to the situation (normal drilling, underfeed, overfeed, rattling, etc.), and also according the number of used rods and the type of the drilled rock. Because we have more than one thousand annotated sounds, it is not possible to manually extract a collection of clicks for each sound. Figure 5.5 shows an example of drilling sounds, and it illustrates the difficulty to detect and to extract the clicks. Moreover, in most cases the single clicks overlap with the following ones. Then, in this part we propose a method to automatically extract a collection of some clicks for each sound, with a duration possibly longer than the time separating two clicks, by removing the contribution of the neighbor clicks.

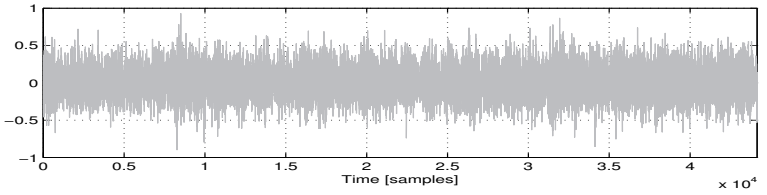

Figure 5.5. Example of a denoised drilling sound. About 34 clicks occur in this sample with duration 1 second.

In this section, first we give some tools for the time-envelope estimation of the drilling sounds, and then all steps of the analysis are successively described.

5.2.1 Time-Envelope Estimation

Integrated Energy To detect the click positions in time and their amplitudes, we define here a smooth time function based on energy integration.

With x_n the denoised drilling sound, w_n a finite weighting window with size $2N+1$, we define the smooth energy function e_n as follow:

$$e_n = \sum_{j=-N}^{N} x_{n+j}^2 w_j^2 \tag{5.8}$$

Note that the sliding window w_n can be defined for $n < 0$, and it is not necessarily symmetrical.

This energy function has the property to efficiently smooth the signal and to raise the clicks as obvious maxima. Then it will be easier to detect the click occurrences by analyzing the peaks of e_n than analyzing the peaks of x_n. This function is similar to eq. (2.23) of sec. 2.3.2, but is more efficient for drilling sounds. Remark that as seen below, the window shape w_n is an important feature in the click detection, and it will be refined later.

Whitening At first look, as such the energy function e_n has obvious peaks which directly correspond to the click occurrences, but has also many local maxima which do not correspond to a click. Moreover, in some cases it is difficult to distinguish a "false" local maximum from a "true" click occurrence when the click has a low amplitude.

For this reason, the original denoised drilling sound is whitened by filtering it by the inverse filter obtained by a linear prediction. The well-known Linear Prediction Coding (LPC), cf. [7], provides an Autoregressive filter $H(z) = 1/A(z)$, with a frequency response which is an estimate of the spectral envelope of the analyzed signal, as seen in sec. 2.2. Then, we compute the LPC filter from the signal x_n and we filter it by the FIR filter $A(z)$. This operation provides a deconvoluted signal y_n which has a flat spectral envelope. We speak about "whitening".

Instead of computing the energy function e_n with the signal x_n, we propose here to analyze its white version y_n. This inverse filtering has the property to remove the minimal phase part, and in some cases, it concentrates the energy of a single click at its beginning. This remark is not strictly true because of the remaining phase contribution which continues to spread the energy, nevertheless, as shown in fig. 5.6, the energy function e_n is significantly smoother with the white signal, and most of "false" maxima left, which will limit the number of wrong detections.

In practice, first the average spectrum $\overline{X}$ is computed by the mean of the rows of the spectrogram V_s, cf. sec. 5.1.1. Then the autocorrelation sequence ρ_n is computed using the inverse Fourier transform of $|\overline{X}|^2$, and the LPC filter $H(z) = 1/A(z)$ is computed using the Levinson algorithm, cf. [7]. Finally, the white signal is computed by filtering x_n by $A(z)$. Here the chosen LPC order is 1024, which is quite high and provides an expensive filter, but this process is computed off-line during the analysis only.

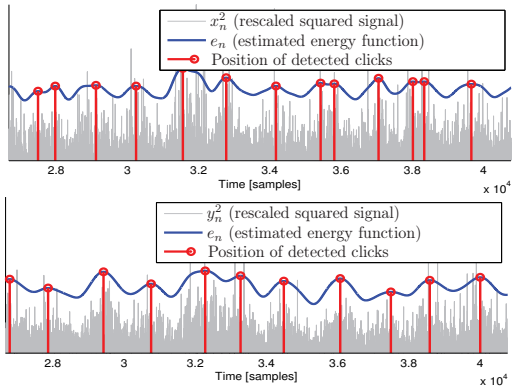

Figure 5.6. Illustration of the time energy function estimation and click detection, using the original signal x_n (top figure) and its white version y_n (bottom figure). In this example, the weighted window is a symmetrical Hann window with size $2N + 1 = 1023$. Remark the three false detections with x_n, which do not occur with y_n.

Modeling the time-envelope of clicks The single clicks are modeled as the product of an unknown signal, possibly stochastic and different for every click, and of an envelope which is the same for all clicks. We here propose to model this envelope using as few parameters as possible.

The envelope of a single click is modeled using an Attack/Decay model. The envelope is separated into two parts:

- The increasing attack with length N_a and parameter α_a. Its formula $\forall n \in [0, N_a]$ is

$$a_n = \begin{cases} \dfrac{n}{N_a}, & \text{if } \alpha_a = 0, \\ \dfrac{1 - \mathrm{e}^{-\alpha_a n}}{1 - \mathrm{e}^{-\alpha_a N_a}}, & \text{otherwise.} \end{cases} \tag{5.9}$$

  Note that this envelope increases from 0 at $n = 0$ to 1 at $n = N_a$. An example of this behavior is shown in fig. 5.7.

- The decreasing decay part has an exponential behavior with α_d the positive damping factor:

$$a_n = \mathrm{e}^{-(n-N_a)\alpha_d}, \quad \forall n \geq N_a. \tag{5.10}$$

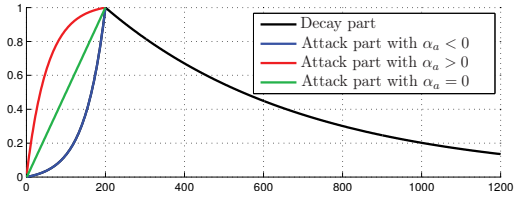

Figure 5.7. Illustration of an individual click envelope with different values of the attack factor. Here the attack time is 200 samples.

In the following sections, the full procedure for the click detection and the envelope estimation is described.

5.2.2 First estimation

First click detection: Based on the estimated energy function computed with the white signal y_n, and with a Hann weighting window with length 1023 samples, a first click detection is done by picking up the local maxima of e_n. Also, in order to know the limits of the clicks, the estimated click position are associated to the pair of local minima on the left and on the right. Then, for all detected clicks, we have: the positions P_m of the peak, P_l and P_r of the closest minimum on the left and on the right respectively.

First estimation of the envelope parameters: The previous size of the window is chosen to eliminate enough wrong local maxima and so to improve the click detection. But the obtained energy function is too smooth to make possible a good parameter estimation. Then a new energy function e'_n is computed with a smaller window with size 511 samples and the envelope parameters α_a, N_a and α_d are estimated by matching the modeled envelope a_n to the square root of the energy function, $\nu(n) = \sqrt{e'_n}$, on all time ranges $[P_l, P_r]$, for all detected clicks, cf. fig. 5.8.

- N_a: the attack time is computed as the median value of all $P_m - P_l$, for all detected clicks.

- α_d: the estimation of the decay factor is straightforward. Using all the values $\nu(P_m)$ and $\nu(P_r)$, for all clicks, all α_a are obtained solving

$$\nu(P_r) = \mathrm{e}^{-\alpha_d(P_r - P_m)}\,\nu(P_m) \Leftrightarrow \alpha_d = \frac{-1}{(P_r - P_m)} \log \frac{\nu(P_r)}{\nu(P_m)},$$

  and the unique estimated value is the mean of all computed values.

- α_a: the estimation of the attack factor is not easy because it is not possible to isolate it from eq. (5.9); we use an iterative procedure to solve the problem. With $P_{l/2} = (P_m + P_l)/2$ the middle point between P_m and P_l, the obtained attack must join the minimum point in P_l and the maximum point in P_m passing through the middle point in $P_{l/2}$, which means that the following equation must be solved:

$$\nu(P_{l/2}) = a(P_{l/2} - P_l)\,\nu(P_m).$$

  This solving iterative procedure is based on the simplex method, cf. [28].

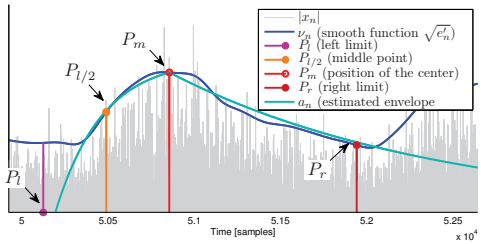

Figure 5.8. Illustration of the first parameter estimation. Here the parameters N_a, α_a and α_d are estimated such that the envelope of the individual click passes through the 4 drawn points. Remark that the fitting is not perfect in this example, because the estimation also takes into account the other clicks.

5.2.3 Second Click Detection

In the previous section, the click detection and the envelope modeling were based on the calculus of the energy functions e_n and e'_n. Unfortunately, this click detection is highly biased, because of the used window w_n which is a Hann window. For example, this window is symmetrical, and because the envelope of the clicks is not symmetrical, the maximum of e_n is delayed and cannot be at the central position of the click, it is usually delayed to the right, cf. e.g. fig. 2.3.

But, having a coherent estimation of the click envelope a_n we can significantly reduce this bias by replacing the Hann window by the click envelope itself. Then a new

refined energy function e_n is computed using $w_n = a_n$, and again, the click positions in time are detected using the local maxima of e_n.

Indeed, if the (denoised) white signal y_n is the product of a stationary signal u_n with variance σ^2, and an envelope signal g_n which consists in a succession of click envelopes a_n, i.e. $g_n = \sum_i \beta_i a_{n-P_i}$ with P_i the time position of the ith click and β_i its amplitude, the energy function is written

$$\begin{aligned} e_n &= \sum_{j=-N}^{N} y_{n+j}^2 a_n^2 = \sum_{j=-N}^{N} u_{n+j}^2 \big(\sum_i \beta_i a_{n+j-P_i} \big)^2 a_j^2 \\ &= \sum_{j=-N}^{N} u_{n+j}^2 \big(\sum_i \beta_i a_j a_{n+j-P_i} \big)^2, \end{aligned} \tag{5.11}$$

and since $a_n \geq 0$, the function $\big(\sum_i \beta_i a_j a_{n+j-P_i} \big)^2$ has local maxima at $n = P_i$, the true click positions, and also e_n because here u_n is assumed to be stationary.

Nevertheless, this operation does not provide a good solution because of some reasons, and we need to add some limitations. First, as explained below, the first parameter estimation of the previous section is also biased, and usually the decay factor is over-estimated, then w_n is defined using the envelope a_n with a decay factor α_d half than the estimated value. Second, the size of the window $w_n = a_n$ is chosen to be half of the mean distance of the detected clicks in the previous section. This limitation reduces the influence of neighbor clicks, and improves the resolution. Third, the new energy function e_n has a lot of "false" local minima. Then the search of the "good" local maxima is done by searching the absolute maxima of the new e_n over the time ranges: $[P_l - N_a, P_m - N_a]$, where P_l, P_r and N_a are the minima positions and the attack time estimated previously. Here the ranges are delayed by N_a, because now the maxima does not give the center of the click, but its beginning in principle.

If no maximum is found in a selected range, then we assume that there is no corresponding click, and the range is deleted. Among the obtained maxima, which are the click positions P_c corresponding to the refined estimation of the beginning of all click occurrences, we also delete some of them such that the minimal distance between two neighboring detected clicks is smaller than half of the median. The choice of the detections to delete depends on the positions themselves and also on the amplitudes e_{P_c}.

5.2.4 Second Estimation of the Envelope Parameters

Having the refined click positions P_c, with less false detections, we now can also refine the estimation of the envelope parameters N_a, α_a, and α_d. However, the estimation of the previous section, based on a fitting of some points of the energy function e'_n, was also biased. Indeed, even with small sliding window w_n, the energy function e'_n does not follow the envelope a_n. Here we propose a more accurate approach. Remark that the previous method makes sense because this new one does not work with an inaccurate detection of click position, which was the case previously.

Because only 3 parameters have to be optimized, we here propose to compute an iterative algorithm for the solving of a non-linear optimization problem. The chosen algorithm is based on the simplex method, cf. [28] and is implemented in Matlab as the standard function *fminseach()*. This algorithm is convenient in our case because of the small number of parameters and because of the complex criterion to minimize.

With a good estimation of the envelope a_n, dividing the signal y_n by the combined envelope $g_n = \sum_i \beta_i a_{n-P_i}$, with P_i the new refined position and $\beta_i = \sqrt{e(P_i)}$ its amplitude, the obtained signal z_n should have an envelope as flat as possible. Then the criterion to minimize is based on the flatness of the energy function f_n computed with z_n. Consequently, from the three parameter values, the criterion C is computed as follows:

- The envelope a_n of an individual click is computed using eqs. (5.9) and (5.10), for all $a_n \geq 0$. Note that $a_n = 0$ for $n < 0$.
- The "combined" envelope g_n is computed: $g_n = \sum_i \beta_i a_{n-P_i}$.
- The signal z_n is computed as follows: $z_n = y_n / g_n$. To avoid the division by 0, the signal before the first detected click is removed.
- The energy function f_n is computed with z_n and a Hann sliding window w_n with size 1024 samples: $f_n = \sum_{j=-N}^{N} z_{n+j}^2 w_j^2$.
- The flatness is now tested by computed the square sum of the difference between f_n and its mean. If z_n is flat, f_n is close to a constant function and the following criterion is small:

$$C = \sum_{n=0}^{N} \left(\frac{1}{N} \sum_{i=0}^{N-1} f_i - f_n \right)^2$$

With this definition of the criterion, the simplex algorithm [28] provides the parameter values which minimize C.

Remark 1: the shape of the w_n is not important here because the energy function is not used to detect clicks or to estimate parameters, but it is just an intermediary function to test the flatness of the envelope of z_n.

Remark 2: the simplex algorithm tests a lot of different set of parameters, to find the minimum. Then to accelerate the computation, the energy function is calculated using an FFT (Fast Fourier Transform) and a product in the frequency domain rather than a cross-correlation computation which is much more consuming here.

5.2.5 Last Click Detection

Again, with the refined envelope parameters we can obtain a refined detection of click positions, and their corresponding amplitudes as in sec. 5.2.3. But now, because the estimation of α_d is not biased, its value is not divided by 2.

Because the new click position P_c are refined, we could imagine to make an iterative process to refine again the envelope parameters, and so on. But in the most favorable cases, the improvements are insignificant, and in the worst cases, the process may be unstable, and may provide worst results. Then, the estimation of the click position P_c, their corresponding amplitude β_c, and the envelope parameters N_a, α_a and α_d stops here.

5.2.6 Click Extraction

With all detected values P_c and β_c for all clicks, and the envelope parameters N_a, α_a and α_d, this section explains how a collection of some click is extracted from the denoised signal x_n. The total number of detected clicks is denoted C_d, and the number of extracted clicks is denoted C_e. Basically, $C_d \approx 70$ for 2 seconds of signals with a frequency of 35 clicks per second, and the chosen number of extracted clicks is $C_e = 10$.

The chosen extracted clicks must check the following constraints: first, to reduce the overlap of the next neighbor clicks, the distance between P_c and P_{c+1} must be as high as possible. Second, to reduce the contribution of neighbor clicks, on the left and on the right, its amplitude β_c must be higher than β_{c-1} and β_{c+1}. Then, all the C_d clicks are sorted according to these criteria, and the C_e preferred clicks are selected

for extraction.

In a first step, the signal y_n is flattened, as in sec. 5.2.4. The flat signal z_n is obtained by the division of y_n by the new "combined" envelope

$$g'_n = \max_{i \in [1, C_d]} \Big(\beta_i a_{n-P_i} \Big).$$

Note that here we use the "max" operator instead of the sum operator. The use of the max avoids the "cumulation" of the tails of the envelopes, whereas the use of the sum favors this cumulation. In section 5.2.4, this effect is used to limit the value of α_d during the optimization, and now it is preferable to limit this effect.

As a result, the signal z_n is flat, which means that the effect of the click envelope is canceled, as shown in the middle sub-figure of fig. 5.9.

Now to extract an individual click with a damping tail which overlaps with the following clicks, we just have to multiply the flat signal z_n by the envelope a_n of an individual click, cf. the bottom sub-figure of fig. 5.9. In this way, neither the contribution of the tail of the previous click is removed, and nor the signal of the following one, in a strict sense. But the contribution of neighbor clicks is efficiently removed for the following reasons: first, thanks to the simultaneous masking, the extracted click of interest efficiently masks the tail of the previous one because its envelope is smaller. Second, the amplitude of the following one is efficiently reduced and thanks to the temporal masking, its contribution is not audible in most of the cases.

Finally, since the clicks are associated with the white signal y_n, then we cancel this whitening by filtering the extracted click samples by the LPC filter $H(z) = 1/A(z)$, cf. sec. 5.2.1. Moreover, since the extracted clicks have different amplitudes, they are normalized to a unique norm.

5.2.7 Realistic Synthesis

The synthesis only consists in playing the C_e extracted clicks with a random selection and with a quasi-constant frequency. However, to provide a more realistic synthesis, it is needed to add a fluctuation in frequency and in amplitude.

First the mean period T_0 in samples of the click occurrences is computed as the mean of the distances of detected clicks:

$$T_0 = \frac{1}{C_d - 1} \sum_{c=1}^{C_d - 1} P_{c+1} - P_c,$$

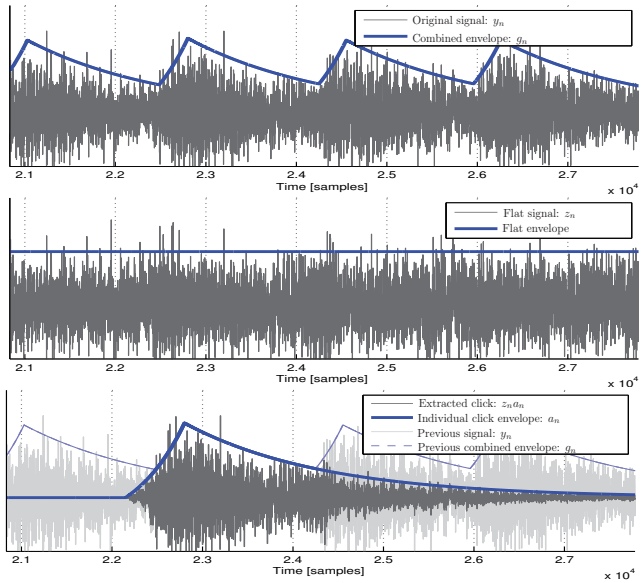

Figure 5.9. Illustration of the click extraction, based on the flattening of the drilling sound. In the top figure, the original signal y_n is shown together with its estimated combined envelope g_n, produced by all detected clicks. In the middle figure, the flattened signal $z_n = y_n/g_n$ is illustrated. Finally, the bottom figure illustrates the extraction of an individual click by the product of the flat signal z_n and the modeled envelope a_n of the click. Note that, at the center of the clicks, the signal is absolutely unchanged, then the percussive nature of the click is conserved; and note that the following click which overlaps with the tail of the wanted click, is efficiently reduced, which makes it imperceptible.

and the mean frequency is then $F_0 = F_s/T_0$. Also, to take account of the variance of the click position, the fluctuation in term of frequencies, the standard deviation is computed as follows:

$$\sigma_T = \left(\frac{1}{C_d - 1} \sum_{c=1}^{C_d - 1} \left(P_{c+1} - P_c - T_0 \right)^2 \right)^{\frac{1}{2}}$$

Second, in the same way the mean amplitude B and its standard deviation are computed:

$$B = \frac{1}{C_d - 1} \sum_{c=1}^{C_d - 1} \beta_i$$

$$\sigma_B = \left(\frac{1}{C_d - 1} \sum_{c=1}^{C_d - 1} \left(\beta_i - A \right)^2 \right)^{\frac{1}{2}}$$

Then, the amplitude of the played clicks are chosen according a Gaussian distribution centered at B and with the variance σ_B^2, and the distances between the clicks

are chosen with a Gaussian distribution centered at T_0, in respect to the mean frequency F_0 in Hz, and the variance σ_T^2. This procedure provides some fluctuations in the amplitudes and the positions which makes the synthesis more realistic.

The selection of the click, among the C_e extracted clicks, is done randomly, but it need to follow some constraints: first, the consecutive repetition of two identical clicks makes the sound synthetic, then for every new click, the selection forbids to select the previously played. Second, in the case where the original signal is long, the timbre of the clicks may change. Then if two consecutive clicks have too different timbres, the synthesis sound may have a *rough* aspect. To avoid this, the original position of the extracted clicks is stored, and a slowly varying time point, *phase*, periodically moves between the first and the last positions. Each time a new click must be played, we choose the click among the 5 closest clicks to this point. This guaranteed to consecutively select clicks which have similar timbres. The cycle of this point has a duration of 4 seconds approximately, but has also a random behavior in order to reduce the perception of *looping*.

5.3 Software Package

This sub-section describes all developed softwares which will be useful further analyses, and especially to implement the real-time synthesis on a rig simulator. The analysis part is developed using the software Matlab, and the real-time synthesis part in C++ language. To facilitate the edition, most of the softwares are Graphical User Interfaces (GUIs).

Note that, even if the C++ synthesis has been developed in a way to facilitate its integration into the simulator, some additional tasks are needed to make a completely safe integration. It is mainly designed to help developers.

5.3.1 Annotation

Because most of the recordings contain some different successive drilling situations (normal drilling, overfeed, rattling, ...), it is necessary to annotate the limits of each part for each sound. The automatic annotation is difficult, and it is not the subject of this work. Moreover, a bad annotation may provide a worse analysis. Then a Graphical User Interface has been developed in Matlab to annotate easily and quickly all the sounds.

First, receiving the name of a directory which contains all the recording files, this software makes the complete list of sounds, and successively prints the signals in a window, cf. fig. 5.10. Then, the user can listen the sound and annotate it by manually placing some markers to delimit all different drilling parts contained in the file. To make these operations faster, the software is especially designed to reduce the number of actions. For example, the left mouse button is used to add a new part, to stretch an existing part or to merge two different parts; the right button is used to remove, to reduce or to cut a part; the middle button is used to move the printed signal or to zoom in or to zoom out. Also the key “space” is used to listen the sound from the current position of the cursor, and using the modifier key “control” it is possible to listen only the selected part. Finally, the key “e” allows to switch to the next sound, and the key “s” allows to save the annotations in a file. Some other tools are also available and their list is given pressing the key “h”.

Remark that to facilitate the analysis, the user has to choose parts as long as possible and as stable as possible, which means that the timbre of the sound may be the same from the beginning to the end.

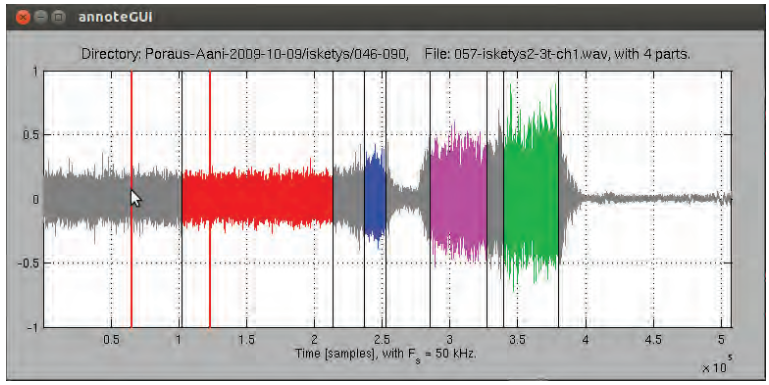

Figure 5.10. Screenshot of the annotator software. In this example, 4 parts are annotated with different colors. Note that the first part corresponds to a normal drilling sound, and the 3 other correspond to 3 different rattling sounds.

5.3.2 Analyzer

The analyzer is a software which computes the analysis of all annotated parts. There is no manual intervention, except one question at the beginning in order to know what sounds must be analyzed.

First, the software makes the list of all annotated files, and also a list of all previously analyzed files. The user has the choice to (re)analyze all annotated files, only the newly annotated files, or just the files of a sub-directory. At the moment of the end of the project, 1296 parts are annotated, and the complete analysis lasts approximately 7 hours, using a 4 core CPU at 3.20 GHz. All steps of sec. 5.2 are computed successively for all annotated parts. Note that to make this process faster, the length of the parts are limited by 2 seconds. Finally, all analyzed parameters, such as the extracted clicks, the envelope parameters N_a, α_a and α_d, and the statistical parameters, F_0, σ_T, B and σ_B, are stored into some Matlab data files. Figure 5.11 shows the printed text during the analysis.

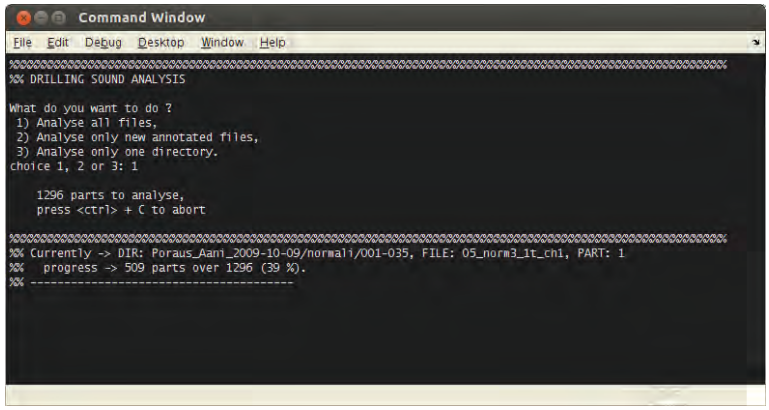

Figure 5.11. Printed text of the analyzer.

5.3.3 Test and Parameter Refinement

To check and eventually modify the analyzed parameters, another GUI has been developed using Matlab, cf. fig. 5.12.

This software loads the analyzed drilling data, computed by the analyzer, and proposes to compare the original noisy sound, the denoised drilling sound and the synthesis. Also, some sliders and editor controls allow to modify the synthesis parameters, and eventually to save the results in the data file.

The interface is decomposed as follows: the “Choose File” panel allows to select the sound, directory and file name, and the annotated part. The “Listen” panel allows to select the original, the denoised and the synthesized sound for listening. It can modify the global volume and activate or stop the playback. The “Parameters” panel is

for the manual edition of all parameters. All the 7 parameters are plotted there: frequency F_0, amplitude B, deviation position σ_T, deviation amplitude σ_B, attack time N_a, attack damping α_a, and decay damping α_d. Finally, the "curves" panel illustrates the clicks distribution and the modeled envelope a_n; and the bottom window shows the annotated parts of the current file.

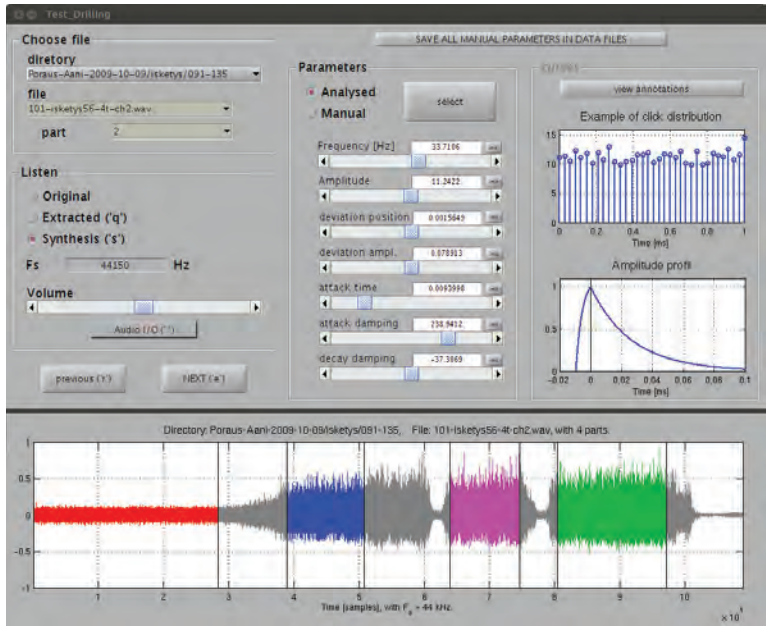

Figure 5.12. Screenshot of the Graphical User Interface for testing and for the parameter modification.

5.3.4 Drilling Data Exportation

All analyzed parameters, and eventually manually modified, are stored into a data file using the Matlab format. To package all the parameters into data files readable by the real-time synthesizer, the exportation program just converts the useful information into new formatted files. Figure 5.13 shows the printed text during the exportation.

Additionally, for the real-time software, a "global" information file is used. This text file gives the sampling rate, the path of the folder which contains all exported files, and a list of the used sounds. The exportation makes a template information file, that the users can modify before the real-time synthesis. All analyzed parts are

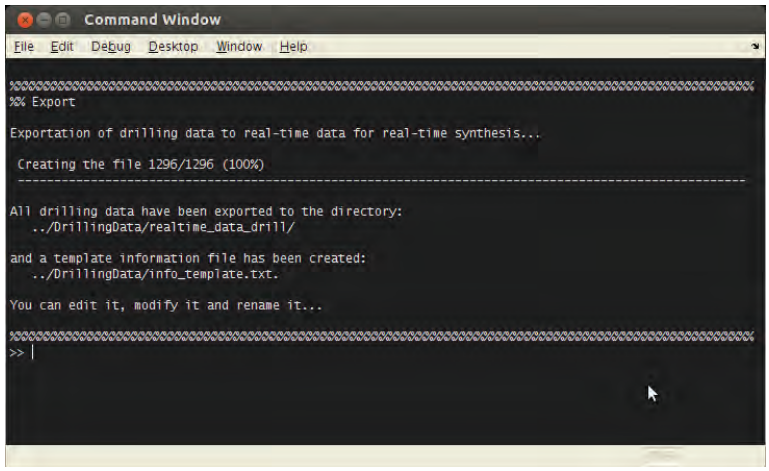

Figure 5.13. Printing text of the exportator.

listed there. This file permits to change the path of the directory of the stored data, and also to modify the list of the used files.

The directory path is given by the field “DATA FOLDER”, cf. fig. 5.14, and can be modified. The sampling rate is given by the field “FS” and must not be modified. Then the list of sounds is given. Three pieces of information are given for all sounds:

- First, the name of the data file contained in the directory. This field should not be modified except if the name of the stored file is also modified. Note that all the names contain: the name of the sub-directory containing the original sound file, the name of the sound file itself, and the number of the annotated part, with the keywords: “DIR”, “FILE” and “PART”.
- Second, a flag is given, set to 0 or 1. On the one hand, the loading of all data files may use a big amount of internal memory, but on the other hand the disk access to load the data each time a sound is played may be slow, and may imply a delay. Then, this “persistent” flag informs if the data files must be loaded once at the launch, or if they must be loaded each time the sound is played.
- Third, because the file name previously described is not always understandable, the user has the possibility to give a new name more understandable. For example, if the sound corresponds to an open rattling sound with 3 rods, this new name can be “Open_Rattling_3”. In the example of fig. 5.14, we have chosen the names of cartoon characters, Dingo, Titi, Casimir...

```
%-------------------------------------------------------------------------------------------
"DATA FOLDER" = '.\drilling_data_export\'

"FS" = '44100'

"DIR_Poraus_Aani_2009-10-07_FILE_03_norm_test_1_2t_PART_1"  = '1' = 'Dingo'
"DIR_Poraus_Aani_2009-10-07_FILE_03_norm_test_1_2t_PART_2"  = '0' = 'Titi'
"DIR_Poraus_Aani_2009-10-07_FILE_04_alis_test_1_2t_PART_1"  = '0' = 'Rominet'
"DIR_Poraus_Aani_2009-10-07_FILE_05_norm_test_1_2t_PART_1"  = '1' = 'Rémy'
"DIR_Poraus_Aani_2009-10-07_FILE_05_iske_test_1_2t_PART_3"  = '0' = 'Casimir'

"DIR_Poraus_Aani_2009-10-07_FILE_05_iske_test_1_2t_PART_3"  = '0' = 'Casimir2'
  = 'F0: 25.39; Gdb: 1; stdT: -1; stdA: 0; attFac: .5; relFac: 2'

"DIR_Poraus_Aani_2009-10-07_FILE_06_norm_test_1_3t_PART_1"  = '0' = 'Pollux'
"DIR_Poraus_Aani_2009-10-07_FILE_06_iske_test_1_3t_PART_2"  = '0' = 'Choubaka'
"DIR_Poraus_Aani_2009-10-07_FILE_07_norm_test_1_4t_PART_1"  = '0' = 'Minus_et_Cortex'
"DIR_Poraus_Aani_2009-10-07_FILE_09_norm_test_1_4t_PART_1"  = '0' = 'Spongebob'
"DIR_Poraus_Aani_2009-10-07_FILE_09_alis_test_1_4t_PART_2"  = '0' = 'Garfield'
"DIR_Poraus_Aani_2009-10-07_FILE_10_syv__test_1_4t_PART_3"  = '0' = 'Milou'
"DIR_Poraus_Aani_2009-10-07_FILE_11_norm_test_1_5t_PART_1"  = '0' = 'Kermit'
"DIR_Poraus_Aani_2009-10-07_FILE_11_norm_test_1_5t_PART_2"  = '0' = 'Peggy'
"DIR_Poraus_Aani_2009-10-07_FILE_11_iske_test_1_5t_PART_3"  = '0' = 'Taz'
"DIR_Poraus_Aani_2009-10-07_FILE_13_ylis_test_1_5t_PART_1"  = '0' = 'Bugs'
"DIR_Poraus_Aani_2009-10-07_FILE_13_alis_test_1_5t_PART_2"  = '0' = 'Winnie_l_ourson'
"DIR_Poraus_Aani_2009-10-07_FILE_14_iske_test_1_5t_PART_1"  = '0' = 'Gizmo'
"DIR_Poraus_Aani_2009-10-07_FILE_14_taip_test_1_5t_PART_2"  = '0' = 'Nounours'
%-------------------------------------------------------------------------------------------
```

Figure 5.14. Example of an information file.

Additionally, it is possible to modify the parameter values at the data loading, as in the line 8 of fig. 5.14, for “Casimir2”. Cf. sec. 5.3.5 for more details about the modifiable parameters.

5.3.5 Real-Time Synthesis Library

For the implementation of the real-time synthesis into the rig simulator, we developed a library which consists in a C++ Class, named `Drilling_Synthesizor`. After the creation of an instance of this class, the simulator can communicate with it using the different methods of the class. For example, the constructor of the class receives the name of the information file, and creates a structure with a table of all used sounds. A method is dedicated to load a new sound, some methods get or set the values of some parameters, and a special method computes the synthesis.

Section 5.3.6 describes a Graphical User Interface which is an example of use of the library, in order to facilitate the implementation of developers. Additionally, this GUI is useful to test the synthesis and to modify the parameters.

In this section, first the internal architecture is briefly described, second the new defined parameters are explained, and finally all the public methods are listed and detailed

Synthesizer architecture: To facilitate the loading and the manipulation of the sounds, a class `Drill_Sound` has been developed for a single sound. The main class `Drilling_Synthesizor` contains a table of `Drill_Sound` instances corresponding to all sounds listed in the information file. All the data of the sound can be loaded at the creation of the object if the persistent flag is "true", otherwise they are loaded using the method `Drill_Sound::load()` when the sound is played. Note that this table is declared as follows:

```
vector<Drill_Sound*> sound_table;
```

To make possible a smooth change between two different sounds, an interpolation between them is possible. For that, two boxes `current_sounds` are created for the simultaneous synthesis of both, and a modifiable variable, between 0 and 1, sets the mix between them. When a new sound must be played, the main `Drilling_Synthesizor` object copies the `Drill_Sound` object from the table to the chosen box. Note that, to reduce the amount of data and the time of the copy, the copy constructor of the class `Drill_Sound` does not copy the table of the click signals. The new object shares the same memory and an internal variable counts the number of objects which share the memory, in order to know when it can be deleted.

Because of the simultaneous synthesis of two different sounds, the random choice of the click position and amplitude must be done by the main `Drilling_Synthesizor` object in order to perfectly synchronize the clicks of the two played sounds. Then, the `Drilling_Synthesizor` object knows some of the synthesis parameters of the sounds objects. They are: the mean frequency F_0, the mean amplitude B and the deviations σ_T and σ_B. An interpolation of this parameters is done between the two sounds.

Moreover, when a newly created sound replaces a former sound in a "box", `current_sounds`, the currently played clicks should not be suddenly stopped. Then, the former sound object is moved from this box to a temporary "box", and the new sound is copy in the chosen box. This temporary box is dedicated to finish the played clicks.

To summarize, the table `sound_table` contains all the available sounds, the two

boxes `current_sounds` contains the currently played sounds, with a possible interpolation, and the temporary box named `thrash_sound` contains the former sound before its deletion. Figure 5.15 illustrates this.

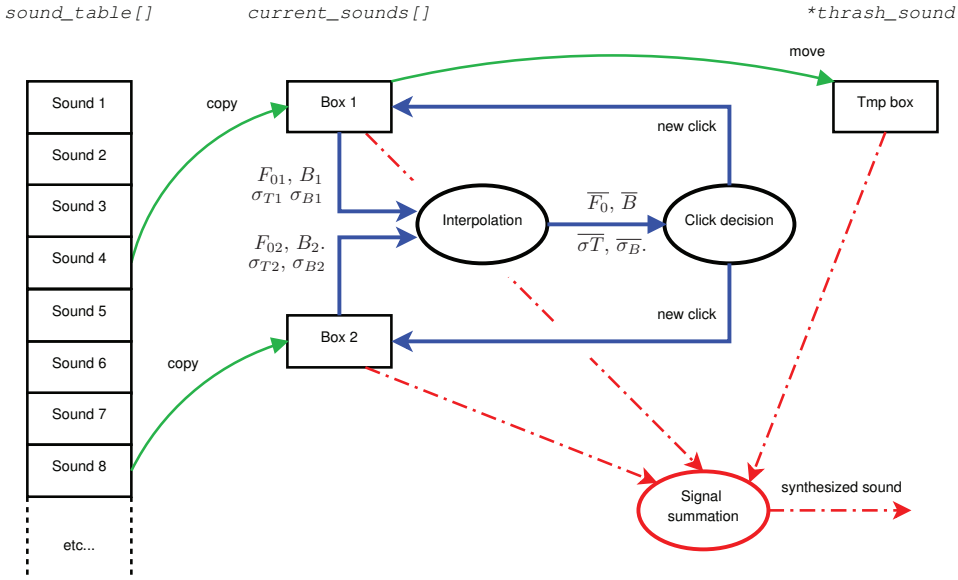

Figure 5.15. Overview of the synthesizer architecture.

New defined parameters: To make possible an easier modification of the parameters during the real-time synthesis, some new parameters have been defined. The aim is not to affect a given new value, but to change it around the analyzed value. For example, instead of giving the absolute linear gain, we can give the relative gain in decibel. Then, because these new parameters modifies the real-time synthesis parameters, we speak about parameter *modifiers*.

Moreover, not only it is possible to change all the parameters of the two currently played sounds, separately, but also it is possible to use "global" parameters to globally change the synthesis.

These modifiers can be set using the following methods: `set_F0_param`, `set_gain`, `set_std_time`, `set_std_amplitude`, `set_attack_factor`, `set_release_factor`. The arguments are the value, and an integer giving the target: 0 for the sound of the first box, 1 for the second sound, and 2 for the global synthesis. Note that the attack time N_a cannot be changed.

Here is the list of all the six parameter modifiers:

- Frequency (`set_F0_param`): for the global synthesis, the modifier value is the translation in half-octave. For the sounds 1 and 2, the given value is the real frequency in [Hz]. note that in this case, it is not a modifier in a strict sense. With $\gamma \in [0, 1]$ the mixing value, F_1 and F_2 the frequencies of the sounds, and m_F the global modifiers, the used interpolated frequency is:

$$f^* = ((1-\gamma)F_1 + \gamma F_2)\ 2^{m_F/2}.$$

- Gain (`set_gain`): the mean amplitude can be modified using the modifiers of the gain. The gains of the individual sounds are given in decibels, and the global gain is linearly given. With g_1 and g_2 the gain modifiers in [dB] of the individual sounds, B_1 and B_2 the analyzed mean amplitudes, and G the global gain, the real mean amplitudes of the clicks of the sounds are respectively:

$$b_1^* = G\ (1-\gamma)\ B_1\ 10^{g_1/20}$$

$$b_2^* = G\ \gamma\ B_2\ 10^{g_2/20}$$

- Deviation in time (`set_std_time`): using the same principle for the standard deviation in time, the modifiers change the used value around the analyzed values. With μ_{T1}, μ_{T2} and μ_{TG}, the modifiers of the first sound, the second sound, and the global synthesis, and with σ_{T1} and σ_{T2} the analyzed value of the sounds 1 and 2 respectively, the used value is then:

$$\sigma_T^* = \left((1-\gamma)\ \sigma_{T1}\ 2^{\mu_{T1}/2} + \gamma\ \sigma_{T2}\ 2^{\mu_{T2}/2}\right)\ 2^{\mu_{TG}/2}.$$

- Deviation in amplitude (`set_std_amplitude`): again, the modifiers of the standard deviation in amplitude change the used value. With μ_{B1}, μ_{B2} and μ_{BG}, the modifiers of the first sound, the second sound, and the global synthesis, and with σ_{B1} and σ_{B2} the analyzed value of the sounds 1 and 2 respectively, the used value is then:

$$\sigma_B^* = \left((1-\gamma)\ \sigma_{B1}\ 2^{\mu_{B1}/2} + \gamma\ \sigma_{B2}\ 2^{\mu_{B2}/2}\right)\ 2^{\mu_{BG}/2}.$$

- Attack factor (`set_attack_factor`): for the envelope shape of the attack, it is possible to separately modify the attack factor of the sounds using a modifier. Note that in this case, no global modifier is defined. With n the number of the

considered sound, 1 or 2, $\alpha_{a,n}$ the analyzed value of the attack factor, and $\beta_{a,n}$ its modifier, during the synthesis the used attack factor of the sound n is:

$$\alpha^*_{a,n} = -\beta_{d,n}/N_a + \alpha_{a,n}.$$

- Decay factor (`set_release_factor`): for the envelope shape of the decay, it is also possible to separately modify the decay factor of the sounds using a modifier. Also in this case, no global modifier is defined. With n the number of the considered sound, 1 or 2, $\alpha_{d,n}$ the analyzed value of the decay factor, and $\beta_{d,n}$ its modifier, during the synthesis the used decay factor of the sound n is:

$$\alpha^*_{d,n} = \alpha_{d,n} 2^{\beta_{d,n}/2}.$$

Excepted for F_1, F_2 and G, note that with $\gamma = 0$ (or $\gamma = 1$ resp.), if the modifier values are 0, then the used parameters are the analyzed values of the sound 1 (or sound 2 resp.). Setting them at 0 makes no effect, 0 is then a "neutral value". For the global gain, the neutral value is 1, because it is linearly defined.

As seen previously, the information file is useful to set the persistent flags and to name the sounds. It is also useful to manually set the modifiers of sounds individually (cf. the sound Casimir2 of the fig. 5.14, line 8). Not only it is useful to refine the parameters by the user, but also it is possible to copy a line with a different new name and different modifiers, consequently the data file will be shared by two sounds with different timbres. The fields `F0`, `Gdb`, `stdT`, `stdA`, `attFac` and `relFac`, respectively initialize the modifiers of F_0, B, σ_T, σ_B, α_a and α_d. Then, when a sound is loaded into one of the two boxes, the analyzed parameters are loaded and are modified by these modifiers; except for the frequency which is replaced by the value of the information file. Note that, if the modifiers are not given in the information file, the original analyzed values are used.

Remark 1: the stored clicks in the data file already have the shape of the analyzed envelope. Consequently, if the envelope is modified, by β_a or β_d, we need to compensate the envelope by doing a division by the original envelope. Note that for the decay, we just need to compute an exponential with the following decay factor $\alpha^*_d - \alpha_d$. Finally, remark that, instead of computing some exponential functions $u_n = u_0 \, \mathrm{e}^{-\alpha n}$ at each sample, we use the recurrence equation $u_n = u_{n-1} \, \mathrm{e}^{-\alpha}$, with properly chosen initial value of u_0.

Remark 2: the modifiers β_a and β_d are defined so that the use of positive values

makes a smoother click envelope, and the use of negative values makes a sharper click envelope.

Class methods:

`Drilling_Synthesizor( string info_file_name );`

Standard constructor of the Class. This constructor receives the name of the information file, and loads all useful data into the internal memory of the instance.

`Drilling_Synthesizor( Drilling_Synthesizor &source_synth );`

Copy constructor. In the case where the object has to be "cloned" to another instance, this method receives the source object and returns the newly created object. Note that, a part of the internal allocated memory is not duplicated, it is the case of the loaded click signal which are in read only mode. Using a count of objects sharing the memory, we can know when this memory has to be deleted.

`~Drilling_Synthesizor( void );`

Destructor of the class. This method deletes the allocated memory, and decrements the count for the sharing memory, or deletes it if it is the last instance that uses it.

`void DSP_Process( int N, float *x );`

Callback of the DSP process of the class. This method returns the new `N` samples of the synthesized sound, into the buffer pointed by `x`. Remark that the number format is the single precision floating points, using 32 bits, which is the usual format for plugins of sound synthesis and audio effects in music. Nevertheless, for Audio APIs, the format is usually the output format which is usually signed integers with 16 bits, then a conversion must be done.

```
void init_sound( int option, int number );
void init_sound( int option, string name );
```

These methods make the loading of sound, and prepares it for synthesis. With the first version, the number of the sound to load is given, this number corresponds to the sorted list of the information file, starting from 1. With the second version, the new name is given, corresponding to the given new name in the information file. Note that if the number 0 is given, then an empty sound

is loaded. Additionally, the integer `option` is the ID of the box where the sound must be loaded: 0 for the first box, and 1 for the second.

```
int get_current_sound_number( int option );
```

Gives the number of the sound currently loaded in the box given by the ID `option`.

```
int get_Fs( void );
```

Returns the sampling rate in Hz, which is the value given in the information file.

```
int get_n_sounds( );
```

Returns the number of available sounds, which is the number of listed sounds in the information file.

```
void switch_sounds( );
```

Switches the currently loaded sounds. The sound of the box 1 goes to the box 2, and reciprocally. Also the mixing value is inversed $\gamma \leftarrow 1 - \gamma$, which provides no artifact in the synthesis. This method is useful to successively play and interpolate some sounds, cf. sec. 5.3.6.

```
string get_file_name( int isound );
string get_new_name( int isound );
```

Returns the name of the data file corresponding to the number `isound`, or its given new name, as in the information file.

```
string get_data_folder( void );
```

Returns the directory path of the data files, as in the information file.

```
void set_mix( float new_mix_val );
float get_mix( void );
```

Sets or gets the current mix value γ. This synthesis parameter is useful to make a smooth interpolation between 2 sounds.

```
void set_F0_param( float value, int option );
void set_gain( float value, int option );
void set_std_time( float value, int option );
void set_std_amplitude( float value, int option );
void set_attack_factor( float value, int option );
```

```
void set_release_factor( float value, int option );
```

These methods set all the different synthesis parameters. The integer `option` is 0 or 1 to set the parameters of the current sound in the boxes 1 or 2, and it is 2 to set the "global" parameters.

```
float get_F0_param( int option );
float get_gain( int option );
float get_std_time( int option );
float get_std_amplitude( int option );
float get_attack_factor( int option );
float get_release_factor( int option );
```

These methods return the current values of the synthesis parameters. Also the integer `option` is 0 or 1 to set the parameters of the current sound in the boxes 1 or 2, and it is 2 to set the "global" parameters.

```
vector<string> get_errors( );
bool is_new_errors( );
```

The first method returns a list of the error messages, and the second returns a flag to know if some errors have occurred since the last call.

5.3.6 Demonstration Application

The graphical user interface, developed in C++, is mainly used as an example of implementation of the C++ Library. Nevertheless, it can also be used to choose the synthesized sounds among the complete list of available sound, and to manually modify the parameters (modifiers).

This application uses standard Graphical APIs dedicated for Windows 32 bits, and the Bass library for audio playback. It is not the best choice, but it has been the most convenient for us. Figure 5.16 illustrates it.

The principle of this application is simple: to associate most of the Class methods with one graphical control. Then the user and the developer can easily understand the role of every method, and the effect of every parameter: the top panel allows to load a sound into the boxes 1 and 2, using the file name, the new defined name or the number. A slider makes possible the interpolation between the two loaded sounds. Also, a button *switch* executes the method `switch_sounds()`. The main panel allows the modification of all parameters. The different parameters can be modified using the corresponding slider and the currently used value is printed above it. Also, a

scrolling menu “Edit Parameters” changes the value of the integer `option`. Finally, the top right panel is useful to set the global volume, to activate or to stop the audio playback, and to compare the synthesis and the original sounds. Additionally, the button *Sequence* launches a sequence of automatic changes of parameters, cf. below.

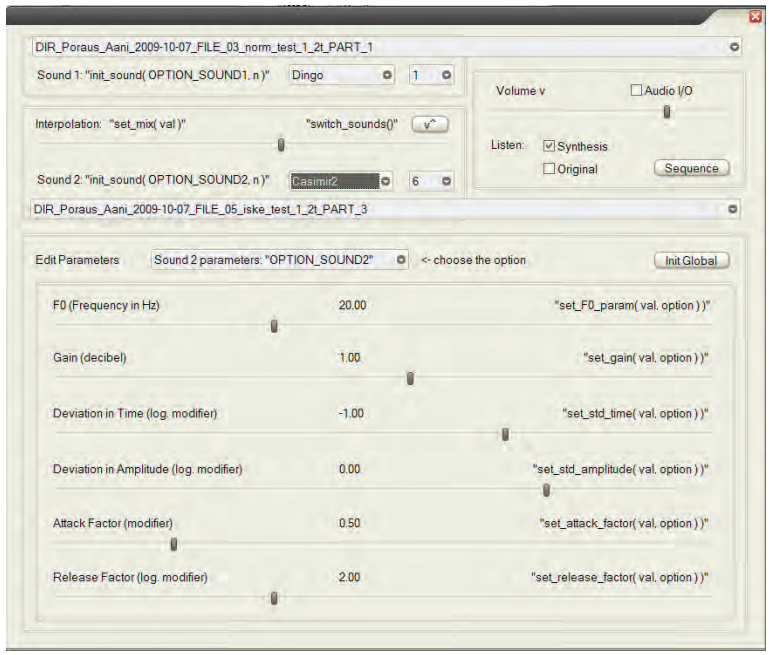

Figure 5.16. Screenshot the Graphical User Interface. Example of implementation of the C++ real-time library.

To illustrate how to use the methods of the class in real-time, we also developed a “sequencer”. Pressing the button “sequence”, a text file containing the sequence is read and played.

In this text file, cf. an example in fig. 5.17, the time synchronization is given by the command `sleep` n, where n is the time to wait in hundredths of seconds before the execution of the next command. The command `load1` *name* loads the box 1 by the sound with name *name*. In the same way the command `load2` *name* loads the second box. The command `switch` interchanges the sounds of the boxes and reverses the mixing value $\gamma \leftarrow 1-\gamma$. The commands `mF0, Gain, stdT, stdA, mix` set the global modifiers and the mix value between 0 and 1. Finally, the command `interpol` *param v1 v2 time*, makes the interpolation of the modifier *param*, cf. previously, from the value *v1* to *v2* during *time* hundredths of seconds.

To successively play the list of sounds, with smooth transitions, the principle is to follow this iterative procedure:

(1) loading of the box 1, with the first sound,

(2) loading of the box 2, with a new sound,

(3) interpolation of the mixing value between 0 and 1,

(4) switch of the sounds, it automatically reverses γ,

(5) return in (2).

```
% ------------------------------------------------------------------------
load1   21-part3
mF0 -1
Gain  0
stdT  0
stdA  0
mix 0
sleep   100

load1    21-part3
interpol Gain 0 .7 10
interpol mF0 -1 0 100
sleep   150

load2 21-part1
interpol mix 0 1 50
interpol Gain .7 1 10
sleep   300

load2 21-part2
interpol mix 0 1 10
sleep   70

switch
interpol Gain 1 .7 10
load2 21-part3
interpol mix 0 1 30
sleep   150

switch
load2 21-part4
interpol Gain .7 1 10
interpol mix 0 1 10
sleep   10
switch
load2 21-part5
interpol mix 0 1 300
sleep   300

interpol mF0 0 -3 30
sleep   30

interpol Gain 1 0 30
sleep   100
% ------------------------------------------------------------------------
```

Figure 5.17. Example of a sequence.

6. Evaluation

6.1 Listening Test at Creanex

6.1.1 Procedure

The hydraulic sounds and the harvester and forwarder sounds created in this project were evaluated in an informal listening test. Two experts were asked to first determine whether the sound sample sounds realistic or not, after which they could rate the sound on a scale of 0 to 5 (0 = poor, 5 = excellent). The expert listeners were both employees of the simulator company and both had excellent knowledge regarding the forwarder and harvester simulators. In addition, one expert had real life experience in operating the actual machines. Some of the sounds presented were played along with video of the actual event to more closely simulate how the sound would work in a simulator. The listening test itself was a very informal event where the experts could give comments and feedback orally and in writing at any time. The experts were also allowed to hear the samples as many times as required.

The idea behind the listening test was to receive feedback from people with real life experience with the machines and simulators in question. Their feedback is essential in determining whether the sounds created for the simulators actually correspond to the actual machines they are simulating. The feedback and results were fairly positive, although some sounds were deemed unrealistic and requiring improvements. Though some sounds failed in sounding realistic enough, the listening test as a whole was very successful as it lead to some excellent discussion and improvement ideas for the sounds. The results and improvements will be discussed below.

6.1.2 Results and Improvements

Table 6.1 shows the results of the listening test. The test Experts are labeled as A and B, where A is the test Expert with experience in operating the actual machines.

Table 6.1. The results of the listening test.

	Does the sound sample sound like the sound event in question?		How realistic is the sound?	
Sound Event	**Expert A**	**Expert B**	**Expert A**	**Expert B**
Feeding	Yes	Yes	3	3
Feeding & Delimbing	Yes	No	2	0
Hydraulic Sounds	Yes	Yes	4	5
Hydraulic Pump	Yes	Yes	4	4
Contact Bunks	Yes	Yes	4	5
Contact Screen	Yes	No	3	2
Contact Grapple	Yes	Yes	4	4
Contact Logs	Yes	Yes	4	3
Load Brake	No	No	0	1

For the first question regarding whether the sound samples actually sound like the sounds they are simulating, Expert A answered "Yes" 8 out of 9 times (89 %) and Expert B 6 out 9 times (67 %). The load brake sound was the one sound both experts agreed did not sound realistic. Both experts still considered a majority of the sounds to correlate to their corresponding sound events. The calculated average scores regarding the authenticity of the sounds were very similar for both experts. Expert A rated the sounds with an average of 3.1 and Expert B with an average rating of 3.0.

6.1.3 Feeding

The feeding sound was played along with a video clip of feeding and it received fairly positive feedback from both experts. Expert A suggested amplifying the sound of the feed rollers, which can apparently be a very loud sound due to the rollers scraping against the bark of a tree. Expert B noted that the authentic characteristics of the sound had been found, although the result could have been much better with higher quality source material. Expert B's point is a very valid one, as the source material

was littered with engine and wind noise, which however was suppressed by filtering, will surely cause loss of quality in the final synthetic result. With a cleaner recording of a feeding event, the loud sound of the feed rollers could possibly be more prominent in the final result.

6.1.4 Feeding and Delimbing

The delimbing sound was played combined with the feeding sound (along with a video clip), but it did not perform well in the listening test. The single crackling sounds were also played separately to the experts. Expert A gave the sound a rating of 2.0 and Expert B did not consider it realistic at all and rated it a 0.0. According to the discussion and feedback on the delimbing sound, the synthesis method itself is fine, but the "crack" or "snap" sounds created are too short and clean sounding. This is mostly due to the source material used, meaning more appropriate sounds of branches breaking would need to be either recorded or obtained. Expert A commented that the sounds more closely resembled the sound of an ax chopping wood, instead of the delimbing knives cutting branches.

6.1.5 Hydraulic Sounds

The hydraulic sounds were played back in a longer sound clip with a combination of the different hydraulic sound types presented in Sec. 3. This longer sound clip aimed to simulate the sound of hydraulic cylinders at work. The single sounds were also played separately and compared to the real hydraulic sounds. The feedback for the sounds was excellent, as Expert A gave the sounds a 4.0 rating and Expert A a rating of 5.0. Expert A especially praised the high frequency variable hydraulic sound, which sounded extremely realistic, cf. Sec. 3.3.4.

6.1.6 Hydraulic Pump

The hydraulic pump sound of the harvester also received positive feedback, which is not surprising as it was synthesized in the same fashion as the other hydraulic sounds, which also received excellent reviews. The sound was played along with a video clip similar to the other harvester and forwarder sounds. Most of the discussion on this particular sound was in determining the difference between the squeal of the turbo and the whistling sound created by the hydraulic pump. According to both

experts, both sounds are very similar as they are both high frequency "squeals" or "whistling sounds". In the end, both experts agreed the sound of the turbo is an even higher frequency sound and this particular synthetic sound is that of the hydraulic pump.

6.1.7 Forwarder Contact Sounds

Four different contact sounds were played back individually to the experts and each sound was rated separately. The sounds were also played back with a short video clip where they were all utilized to present the sounds in the correct context. The four contact sounds reviewed were: logs hitting the bunks (vertical poles), logs hitting the screen (the back of the carriage), sound of the grapple opening, and the sound of logs dropping. As shown in Table 6.1, the contact sounds scored fairly well with both experts. The screen sound received some critique as the sound of the grapple opening can apparently be heard in the sound sample. A lower frequency sound was also requested for the sound of the logs dropping, as they currently sounded a bit too synthetic.

6.1.8 Load Brake Sound

The synthetic load brake sound scored very poorly in the test (A: 0, B: 1). This is mostly due to a lack of source material, as the load brake is extremely hard to hear in the forwarder videos (practically inaudible). The sound was synthesized on the basis of the sound sounding like a squeaky bicycle saddle. The synthesis method itself is correct, but an actual recording of a load brake would be required to achieve a realistic result.

6.2 Listening Test at Sandvik

6.2.1 Procedure

In a similar way, the sound synthesis of drilling sounds has been evaluated using a listening test with employes of Sandvik. Four experts who have an excellent experience in drilling machines, were asked to determine whether the sound samples are realistic or not, after which they could rate the sounds on a scale from 0 to 5 (0 = poor, 5 = excellent). Again, this listening test was informal, the subjects could give oral or written comments and feedbacks, and they were allowed to listen every sound as many times as wanted.

In this test, some situations have been tested: normal drilling, under feeding, over feeding, deep drilling, bending, closed rattling and open rattling; cf. sec. 5 for a description of these situations. For most of the situations, we also tested different frequencies and different values of the deviations in time and amplitude, σ_T and σ_B. Indeed, according to preliminary informal tests, we expected than the analyzed values of the deviations were over-estimated, then we decided to test the original obtained values, and also reduced values.

Remark that the original denoised sound has been played; the subjects were informed and they were asked to rate the denoising. When the original sound was played, it was always in second position, and the played first sound was always the apparently best result according to our preliminary informal test.

In the next section, all situations are presented, with the descrition of the tested sounds, the results, and comments. The last section gives some general comments about the synthesis.

6.2.2 Normal Drilling

For normal drilling, we tested first the apparent best deviations, half the analyzed values; second the original denoised sound; third the synthesis with the analyzed frequency and the analyzed deviation; then the synthesis with half deviations and modified frequency: 30Hz and 40Hz. Remark that the length of all played sounds was 5 seconds, as with the other situations, cf. sections below, but the original denoised sound has a length of 2 seconds, then the sound had to be looped. Even if this looping effect was audible, it was not disturbing because of the relative stability of

the sound. Remind that this problem never occurs with synthesized sounds. They are summarized here:

- Sound 1: Analyzed frequency (34Hz), deviations divided by 2.
- Sound 2: Original sound.
- Sound 3: Analyzed frequency (34Hz), and analyzed deviations.
- Sound 4: Modified frequency $F_0 = 30$Hz, and deviations divided by 2.
- Sound 5: Modified frequency $F_0 = 40$Hz, and deviations divided by 2.

All the results for normal drilling are collected in next table. For all sounds, the subjects were asked to say if the sound is realistic, using the letter "y" for *yes*, or not using "n" for *no*. Also they were asked to rate the realism from 0 to 5. Note that in some cases, for unknown reasons, some subjects did not rate some sounds. The missing answers are noted using "-".

	Operator 1		**Operator 2**		**Operator 3**		**Operator 4**	
sound 1	y	5	y	3	y	3	y	3
sound 2	y	5	y	2	y	3	y	-
sound 3	y	4	y	3	y	2	y	1
sound 4	n	2	y	2	y	1	n	-
sound 5	y	3	y	3	y	3	y	4

As a first trend, the sound 1 is the preferred one for most of the subjects, as expected. Compared to the sound 3 which has the analyzed deviations, the rates are slightly better, which confirms that the deviations were over-estimated. Note that the original sound, number 2, has similar rates, and the subjects judged that the sound 1 is really close to the original. Also, all the sounds 1, 2, 3, and 5 are judged realistic, and the worse sound is the number 4 with lower frequency which has been marked unrealistic by two experts.

Consequently, this results shows the accuracy of the analysis and the quality of the synthesis. Nevertheless, as expected, the deviations were slightly over-estimated, and we modified the analysis in order to automatically choose refined deviations.

Two additional and interesting comments were given: first a subject said that the sound 4, with lower frequency, seems to be from a small machine, and the sound 3 with higher deviations seems to be from an under-feeding. The first comment implies that we can reduce the audibly perceived size of the machine by reducing the fre-

quency, and the second comment suggests that the under-feeding situation is characterized by an irregular drilling. These remarks may have also physical justifications.

6.2.3 Under Feeding

For the under-feeding situation, the tested sounds are descibed below, and the results are given by the next table.

- Sound 1: Analyzed frequency (36Hz), and deviations divided by 2.
- Sound 2: Original sound.
- Sound 3: Analyzed frequency (36Hz), and analyzed deviations.

	Operator 1		**Operator 2**		**Operator 3**		**Operator 4**	
sound 1	y	5	y	4	y	3	y	4
sound 2	y	5	y	4	y	3	y	-
sound 3	-	3	n	2	n	1	n	-

Again, the preferred synthesized sound is the first sound, with half the value of the analyzed deviations; and again, the sound 1 and the original sound, number 2, were judged similar. Note that, a subject said that the synthesized sound 1 is even slightly better than the original sound.

6.2.4 Over Feeding

For the over-feeding situation, the tested sounds are descibed below, and the results are given by the next table.

- Sound 1: Analyzed frequency (27Hz), and deviations divided by 4.
- Sound 2: Original sound.
- Sound 3: Analyzed frequency (27Hz), and analyzed deviations.

	Operator 1		**Operator 2**		**Operator 3**		**Operator 4**	
sound 1	y	5	y	2	y	3	y	3
sound 2	y	5	y	3	y	4	y	-
sound 3	y	3	y	1	y	1	n	-

We obtain similar results as with the under-feeding situation.

6.2.5 Deep

For deep drilling, the tested sounds are descibed below, and the results are given by the next table. Note that, here we only tested the synthesis with original analyzed value, and the original denoised sound.

- Sound 1: Analyzed frequency (34Hz), and analyzed deviations.
- Sound 2: Original sound.

	Operator 1		**Operator 2**		**Operator 3**		**Operator 4**	
sound 1	y	5	y	2	y	3	y	4
sound 2	-	5	y	4	y	4	y	4

Obviously, even if the rates of the synthesis are mainly slightly lower than the original sound, the synthesis is judged realistic. Note that as with the under-feeding situation, a subject judged that the synthesized is better than the original sound.

6.2.6 Bending

For bending rod string, the tested sounds are descibed below, and the results are given by the next table.

- Sound 1: Analyzed frequency (40Hz), and deviations divided by 2.
- Sound 2: Original sound.
- Sound 3: Analyzed frequency (40Hz), and analyzed deviations.

	Operator 1		**Operator 2**		**Operator 3**		**Operator 4**	
sound 1	y	5	-	-	y	2	y	4
sound 2	y	5	-	-	y	2	y	1
sound 3	y	4	-	-	y	1	y	2

We obtain similar results as with the under-feeding situation. Note that most of the subjects agreed to say that the sound 1 and the original sound, number 2, are very similar in this example. They also added that the bending situation is very hard to distinguish from normal drilling.

6.2.7 Rattling (Closed)

For closed rattling, the tested sounds are descibed below, and the results are given by the next table. Note that in this case, the original sound sample is very short, almost half a second, and the timbre changes a lot. This produced a very disturbing looping effect, and that's the reason why we decided to not play the original sound. Here, we only compared two different frequencies: the original analyzed frequency, 41Hz, and a modified frequency, 30Hz.

- Sound 1: Analyzed frequency (41Hz), and deviations divided by 2.
- Sound 2: Modified frequency $F_0 = 30$Hz, and deviations divided by 2.

	Operator 1		**Operator 2**		**Operator 3**		**Operator 4**	
sound 1	y	3	y	4	y	3	y	2
sound 2	y	4	y	2	y	1	y	1

Because the preferred synthesized sound has the original frequency, this test reveals again that the frequency is also an important property for realistic sounds. Even if we had not planned to test the playback volume, the subjects remarked that an important property of rattling sounds is the level. Indeed, rattling always occurs with very high level and very high frequency components. Then, it is obviously important to adjust the gain of rattling sounds with higher level than with other situations.

6.2.8 Rattling (Open)

For open rattling, the tested sounds are descibed below, and the results are given by the next table. For the same reason than with closed rattling, the original denoised sound is not tested here. Again, we only tested the frequency.

- Sound 1: Analyzed frequency (30Hz), and deviations divided by 2.
- Sound 2: Modified frequency (25Hz), and deviations divided by 2.
- Sound 3: Modified frequency (40Hz), and deviations divided by 2.

	Operator 1		**Operator 2**		**Operator 3**		**Operator 4**	
sound 1	y	4	y	3	y	3	y	4
sound 2	y	3	y	2	y	1	y	2
sound 3	y	3	y	4	y	4	y	4

Again, the original analyzed frequency is the preferred one, and the subjects did the same remarks as with open rattling about the sound level.

6.2.9 General Comments, Conclusion and Improvements

In a general conclusion of these listening tests: first, the original analyzed frequency is always the preferred one; second, the ratting sounds, closed and open, need a higher level; and third, the original analyzed deviations were over-estimated in most of the cases.

As a consequence of the given rates, these tests revealed that the developed analysis is very accurate and the proposed synthesis is realistic with a high quality. Moreover, in many cases some subjects judged the synthesized sounds, with adjusted deviations, better and clearer than the original sounds. Nevertheless, as seen, the deviation were over-estimated, thereafter we adjusted the analysis parameters such that the good deviations were automatically chosen by the analysis procedure.

7. Conclusions and Future Work

This report presented different sound synthesis methods for working machine simulators. Sounds were synthesized for three different simulators and their corresponding machines: the forest harvester and forwarder, a drill rig, and a truck-mounted hydraulic platform. Several different signal processing techniques and synthesis approaches were employed, including filtering, spectral subtraction, linear predictive coding, non-negative matrix factorization, envelope estimation, and peak detection. The work was divided into four main parts: hydraulic sounds, forwarder/harvester sounds, drilling sounds, and evaluation.

Hydraulic sounds were synthesized using LPC and a white noise excitation signal. Several different hydraulic sounds were first analyzed and extracted from the source material and then synthesized. These hydraulic sounds included: the basic sound, a fading sound, a high frequency sound, a variable high frequency sound, and three different piston contact sounds. The benefits of the synthetic versions of these sounds include the ability to create sound samples of any length required without a looping sound effect and the advantage of being completely free of background noise. The hydraulic sounds created in this project can be utilized in any of the three simulators, as the machines themselves rely heavily on hydraulic operations. The simulators did not include existing hydraulic sounds, thus these newly created sounds can add a new element of reality to the training process.

The main sounds synthesized for the forest harvester and forwarder were feeding, delimbing, and basic contact sounds. LPC and sliding average filters were employed in the synthesis of the feeding and delimbing sounds. LPC was used to extract the spectral features of the sounds and sliding average filters isolated the amplitude envelopes, which combined with a white noise excitation formed a synthetic result. Contact sound samples were simply created by applying spectral subtraction to noisy real

life sound samples of the different contact situations.

A simple drilling sound already existed in the drill rig simulator which utilized pitch shifting to alter the drilling frequency. This project aimed to improve the drilling sound by removing the need for pitch shifting, which can often lead to very unrealistic sounding results. This was achieved by separating single click sounds from the original drilling sound sample and playing them back at the correct drilling frequency. As seen, first the original sounds are denoised using a NMF based method, then an optimization procedure estimates the time envelope of all clicks. Finally the envelope is used to extract some single clicks, with a significantly reduced contribution of the neighbouring clicks. The synthesis is then really low-cost because it only consists in playing back the extracted clicks in real-time, using the wanted frequency, and with some fluctuations in frequency and in amplitude to improve the realism. A software package has been developed in order to facilitate the work of developers of rig simulators.

The last section presented the two listening tests which were performed regarding the hydraulic, forest machine, and drilling sounds. Some experts were employed in evaluating the sounds created in a very informal listening test, which consisted of simple questions and oral feedbacks. The sounds scored fairly well in the evaluation with some sounds performing better than others. The sessions was an extremely fruitful one with excellent discussion and ideas on how to improve the sounds.

Better source material would also help in improving some of the harvester and forwarder sounds. Delimbing would require better and more suitable recordings of branches breaking to correctly mimic the sound of a delimbing event. The sound of a load brake could also be successfully synthesized with an actual recording of the sound. Although in some cases the lack of proper source material decreased the quality of the synthetic results, the methods presented in this project should still prove to be valuable and useful in improving the sound environment in working machine simulators.

Bibliography

[1] A. Okapuu-von Veh, R. Marceau, A. Malowany, P. Desbiens, A. Daigle, R. Gauthier, A. Shaikh, and J. Rizzi, "Design and operation of a virtual reality operator-training system," *IEEE Transactions on Power Systems*, vol. 11, no. 3, 1996.

[2] H. Ploner-Bernard, A. Sontacchi, G. Lichtenegger, and S. Vössner, "Sound-system design for a professional full-flight simulator," in *Proc. of Int. Conference on Digital Audio Effects (DAFx'05)*, 2005.

[3] S. Vössner, R. Braunstingl, H. Ploner-Bernard, and A. Sontacchi, "A new functional framework for a sound system for realtime flight simulation," in *Proc. of Int. Conference on Digital Audio Effects (DAFx'05)*, 2005.

[4] S. Oksanen, J. Parker, and V. Välimäki, "Physically informed synthesis of jackhammer tool impact sounds," in *Proc. of Int. Conference on Digital Audio Effects (DAFx-13)*, 2013.

[5] P. R. Cook, *Real Sound Synthesis for Interactive Applications*. AK Peters, Ltd., 2002.

[6] D. O'Shaughnessy, "Linear predictive coding," *IEEE Potentials*, vol. 7, 1988.

[7] J. Makhoul, "Linear prediction: A tutorial review," *Proceedings of the IEEE*, vol. 63, 1975.

[8] S. Oksanen, J. Parker, and V. Välimäki, "Vibroacoustic analysis and synthesis of struck metal bars using musical instrument modeling techniques," in *Akustiikkapäivät (Finnish Acoustics Days)*, 2013.

[9] J. Smith, *Viewpoints on the History of Digital Synthesis*, 1991, online access June 26, 2014. [Online]. Available: https://ccrma.stanford.edu/~jos/kna/

[10] ——, *Introduction to Digital Filters*, 2007, online access July 14, 2014. [Online]. Available: https://ccrma.stanford.edu/~jos/filters/

[11] R. Moog, "A voltage-controlled low-pass high-pass filter for audio signal processing," in *17th AES Convention*, 1965.

[12] A. Huovilainen, "Non-linear digital implementation of the Moog ladder filter," in *Proc. of Int. Conference on Digital Audio Effects (DAFx-04)*, 2004.

[13] S. Golestan, M. Ramezani, J. Guerrero, F. Freijedo, and M. Monfared, "Moving average filter based phase-locked loops: Performance analysis and design guidelines," *IEEE Transactions on Power Electronics*, vol. 29, no. 6, 2014.

[14] R. Lyons, *Understanding Digital Signal Processing*. Pearson Education, 2011.

[15] J. Kaiser, "On the use of the i_0-sinh window for spectrum analysis," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-28, no. 1, 1980.

[16] E. Weisstein, *Modified Bessel Function of the First Kind, MathWorld–A Wolfram Web Resource*, online book, accessed October 16, 2014. [Online]. Available: http://mathworld.wolfram.com/ModifiedBesselFunctionoftheFirstKind.html

[17] V. Välimäki and A. Huovilainen, "Oscillator and filter algorithms for virtual analog synthesis," *Computer Music Journal*, vol. 30, no. 2, 2006.

[18] U. Zölzer, *DAFX - Digital Audio Effects*. Wiley, 2002.

[19] S. Boll, "Suppression of acoustic noise in speech using spectral subtraction," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-27, no. 2, 1979.

[20] J. Allen, "Short time spectral analysis, synthesis, and modification by discrete Fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-25, no. 3, 1977.

[21] H. Nia and H. Hu, "Applying Bayesian decision theory to peak detection of stochastic signals," in *4th Computer Science and Electronic Engineering Conference (CEEC)*, 2012.

[22] M. Heiniö, *Rock Excavation Handbook*. Sandvik Tamrock Corp., 1999.

[23] C. Févotte, B. Torrésani, L. Daudet, and S. Godsill, "Sparse linear regression with structured priors and application to denoising of musical audio," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 1, 2008.

[24] D. Lee and H. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, 1999.

[25] P. Smaragdis and J. Brown, "Non-negative matrix factorization for polyphonic music transcription," in *IEEE Workshop on Appl. of Signal Proc. to Audio and Acoust. (WASPAA03)*, 2003.

[26] D. Lee and H. Seung, "Algorithms for non-negative matrix factorization," *Advances in Neural Information Process. Systems (NIPS). MIT Press*, 2000.

[27] E. Walter and L. Pronzato, *Identification of Parametric Models*, ser. Communications and Control Engineering, 1997.

[28] J. Lagarias, J. Reeds, M. Wright, and P. Wright, "Convergence properties of the Nelder-Mead simplex method in low dimensions," *SIAM Journal of Optimization*, vol. 9, no. 1, 1998.

[29] NIST/SEMATECH, *e-Handbook of Statistical Methods*, online book, accessed September 26, 2014. [Online]. Available: http://www.itl.nist.gov/div898/handbook/

This report summarizes the results of the REMES project, ''Realistic Machine and Environmental Sounds for a Training Simulator to Improve Safety at Work''. Its aim is to improve the sound environment in working machine simulators.

Modern simulators are visually and operationally extremely advanced and realistic, but the sound environment is still limited. By improving the sounds in these simulators, the simulators can grow into exceptionally realistic training tools with the ability to fully educate future operators in a completely safe environment.

Existing sounds are improved and new sounds are created for three different simulator types: a forest harvester and forwarder simulator, a drill rig simulator, and a truck-mounted hydraulic platform simulator. The main sound types synthesized are hydraulic sounds, drilling sounds, feeding and delimbing sounds (forest machines), and basic contact sounds.

Aalto-ST 16/2014

ISBN 978-952-60-5963-1 (printed)
ISBN 978-952-60-5964-8 (pdf)
ISSN-L 1799-4896
ISSN 1799-4896 (printed)
ISSN 1799-490X (pdf)

Aalto University
School of Electrical Engineering
Department of Signal Processing and Acoustics
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS