Time-Domain Alignment of Non-Stationary Signals

Jason FILOS

Final Year Project Report 2007

Digital Signal Processing
Department Electrical and Electronic Engineering
Imperial College London
England
June 2007

Dedicated to

My parents

Abstract

Time-Domain Alignment problems arise in various fields of Digital Signal Processing such as Audio Engineering and Speech Processing. Replacing a segment of audio data from a large continuous waveform with an alternative passage intrinsically holding near to perfectly similar data, is not as straightforward as it may appear, and is often associated with cumbersome manual labor carried out by an audio editor or recording engineer. For a large collection of audio data where multiple edits are needed manual segmentation and time alignment is impractical and expensive. This report outlines research into an automated software implementation aiming to optimally time align musically related audio data. The motivation for such an implementation is that optimal solutions may be sought for more efficiently for a large set of data minimizing the decisional requirements of an external operator. Through time and frequency domain feature extraction and analysis significant performance improvements can be achieved over the traditional manual segmentation.

Acknowledgements

I would like to thank my supervisor Dr. P. A. Naylor for his excellent guidance and insightful criticisms throughout the course of this project. Furthermore I would like to thank all my family, for showing me never to give up, but especially my parents for their loving care and support. To each of the above, I extend my deepest appreciation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ever since the introduction of computers in our world, their task has been to improve on the way we carry out our work, helping us to complete even the most mundane of tasks. The predicament of time-domain alignment of non-stationary signals, such as audio data, is a problem which occurs in various working environments such as music and TV/Radio broadcasting studios. It is part of a subset of problems which have not been sufficiently addressed by today's standards. The creation and performance of music is thought to be a pleasant experience. However not many of us realize that it is associated with a lot of cumbersome and tedious work, such as archiving the audio material and segmenting it. In this project we will outline a methodology aimed at finding the ideal alignment point between two sequences of related audio data in an automatic and autonomous way, lifting the burden from the tape operator (or audio editor) and transferring it the computer. This way not only can the operator focus on more important things, but ultimately the cost and overhead for the studio, and the audio industry in general, can be greatly reduced.

1.1 Motivation

The motivation for this project arises from the fact that no automatic method for the procedure of time alignment of audio sequences has been presented yet. Drawing on concepts encompassing areas such as digital signal processing in general, and speech processing in particular, we wish to outline a methodology by which this problem

can be solved in an objective and precise way.

1.2 Objectives

We will set the following objectives for this project:

- Outline a method for reading-in a variety of audio signals in a meaningful way.
- Provide a measure of similarity between two sequences of audio data.
- Realize a mapping of one sequence onto the other, based on which regions coincide between the two.
- Force an alignment between the two if no linear mapping pattern is observable.
- Estimate the optimal alignment point at which both these sequences can be interchanged without observing an audible difference.
- Provide an objective measure of the success of this alignment.

Chapter 2

Background

The idea of audio alignment and its context is perhaps best illustrated by an example. When a piece of music is recorded in a digital audio studio environment several passages of the score are performed and recorded numerous times. From this collection of so called audio-takes a subset is selected as the resulting piece of music. Traditionally each segment is auditioned by the recording engineer and manually arranged (i.e. time aligned and placed in context with respect to the other audio-takes) on a software/hardware host sequencer. For modern Pop/Rock productions and especially electronic music of recent days this is a relatively straight forward process since the music is recorded with a click-track or metronome, i.e. all independent segments are synced or time aligned to a master tempo track. Classical music, along with some other styles such as free-form Jazz, do not follow this pattern and are of variable speed throughout. Consequently no clear time segmentation exists and a manual patching and fixing method is employed to align each audio part into context.

Thus in order to automate the procedure outlined above one would choose a suitable segment from the set of audio-takes and compare it to the whole piece of music, locating the time instant where the most similar segment (or even the same albeit in a different form) resides.

2.1 Introduction to Audio Alignment

Consider Figure 2.1 depicting a musical audio file. Within this file a certain region has been selected, containing an erroneous sequence. We wish to replace this erroneous passage with an alternative segment of audio data which was recorded at some other time. One way to do this would be to manually listen to all existing recordings of that passage and choose the best version. One would then proceed by "cutting" out the region of interest from the optimal version obtained earlier, resulting in our example in an audio snippet of approximately 2.5 seconds of length. The highlighted passage depicted in figure would then be removed (or muted) and replaced by the new audio snippet. Traditionally this is done by creating a linear amplitude fade between the two files, called a *crossfade*, in order to prevent digital distortion or clipping. Thus in order to summarize the above procedure the following steps need to be made:

1. Choosing a region to be replaced by inserting a left and right (L+R) marker in the audio track.
2. Auditioning the alternative audio takes.
3. Choosing the correct segment from the alternative take by placing new L+R markers and effectively cutting out that region.
4. Replacing the old region with the new snippet using a linear crossfade.

As one can see from the above procedure four markers need to be manually placed in two different audio files. As one can observe a lot of time and effort is associated with this methodology. Furthermore the criteria for correctly placing these markers are based on a subjective evaluation of browsing through the data and auditioning the result. A more efficient way of doing this would be to manually place the first two L+R markers in the target file and have an algorithmic representation aimed at finding the suitable snippet automatically and time aligning the result, effectively making steps 3-4 obsolete. In order for this to work the various audio features or semantics of the data need to be analyzed in an automated and autonomous process.

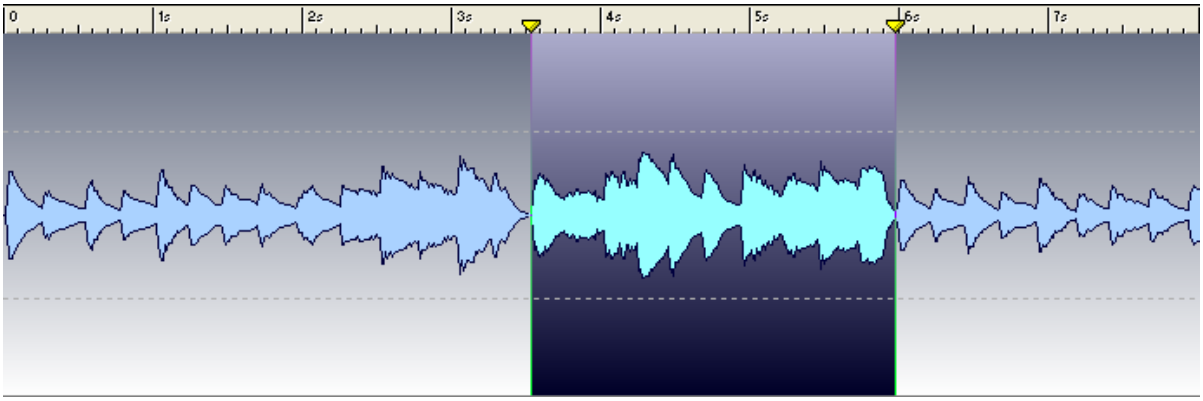

Figure 2.1: Introductory passage to Beethoven's "Für Elise."

In other words, one needs to extract discrete features from the input data and store them in an appropriate data structure. A suitable algorithm then employs this information to automatically time-align a set of audio files.

2.2 Extracting Audio Features

In order to find the means of comparing individual audio segments, a suitable global feature vector needs to be implemented. This feature vector implemented at block precision level of overlapping frames computed once every 10 ms over a window of 20 ms contains time and frequency domain information relative to the input waveform. The temporal features contained reveal information about a note *Onset*, or other events incurring a considerable rate of change in the input data. The time localization of these events is build upon on a method involving the Short-term Energy of the signal and its first time derivative. The spectral feature extraction is based on a mel-cepstral analysis where a variable number of Mel Frequency Cepstrum Coefficients (MFCCs) are computed, and on a more traditional frequency analysis involving the Fast Fourier Transform (FFT). Moreover other *static* features, such as the A-weighting of the spectrum (as defined in IEC/CD 1672), along with *dynamic* features, such as the local time derivatives of the short-term spectrum or cepstrum (delta coefficients), are included. In addition to that a scheme for a polyphonic pitch detection function employing the Short-Time Fourier Transform (STFT) is outlined.

2.2.1 Time Domain Analysis

Time-Domain Analysis is conducted in multiple stages which are broken down into the following categories (ref. Figure 2.7):

- Pre-processing of the Signal (Optional)
- Reduction based on the pre-processed Signal
- Peak-Picking by estimating the local maxima from the Detection Function
- Onset localization

Using a reduction method of the original audio signal based on temporal features such as the Short-term energy (2.1) and its first time derivative, one is effectively implementing an envelope follower revealing clear peaks in the detection function at note onsets. According to [1] by rectifying and smoothing (2.2), where $w(m)$ is an N-point window or smoothing kernel, centered at $m = 0$, further enhancements can be made to the detection function. Additionally according to [4] we define the logarithm of the detection function based on the Short-term energy, such that $\widetilde{d} = \log[d(n)]$, where $d(n)$ is the detection function.

$$E(n) = \frac{1}{N} \sum_{n} x^2(n) \tag{2.1}$$

$$E_0(n) = \frac{1}{N} \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} \left[x(n+m)^2\right] w(m) \tag{2.2}$$

After a reliable reduction stage resulting in a suitable detection function the peak-picking algorithm aims at localizing note onsets as discrete events. This peak-picking stage is broken down into three stages itself:

- Post-Processing : DC Removal / Normalization
- Thresholding: Fixed or Median Thresholding
- Peak-Picking: Fixed Thresholding

Fixed thresholding methods define onsets as peaks where the detection function exceeds the threshold: $d(n) \geq \delta$, where δ is a positive constant and $d(n)$ is the detection function. Since we are dealing with non-stationary signals which might evolve drastically different over time we cannot simply rely on a fixed thresholding stage. Hence we define a thresholding function based on a median threshold (2.3).

$$\tilde{\delta}(n) = \delta + \lambda \, median \left\{ |d(n-M|, \ldots, |d(n+M)| \right\} \tag{2.3}$$

Additionally to a median thresholding function one could check for local minima in the reduction function.

As soon as the regions of interest have been defined, i.e. the *onsets* detected and localized the algorithm proceeds to the final peak-picking stage. A fixed threshold is employed here to define the precision at which our model progresses through the input data. According to the input data a variable order of magnitude can be selected; a sensible value for a piano recording for instance would be between 10 - 12 dB. The resulting output data from our analysis now contains the discrete events for which there is a considerable rate of change in our input audio, such as a new note onset, a pause in the recording or the introduction of a further musical layer such as a new voice or even a new instrumental layer. The methods outlined in this section summarize the time domain information extracted from the input audio data as one part of the global feature vector.

2.2.2 Frequency Domain Analysis

Spectral features are an important element to consider for a successful implementation of the audio alignment procedure. Previous commercial methods explored [6] rely solely on a time-domain representation, however for a more general solution to our alignment problem, a range of frequency domain based methods need to be explored. Frequency dependent analysis is achieved by means of a suitable transformation applied to the input data, such as the Fourier Transform and expansions on the former including the A-weighting of the obtained spectrum and a parametric representation of the acoustic signal such as the Mel-Frequency Cepstrum Coeffi-

cients. These extensions take into account the auditory perception based on ear physiology and other psychoacoustical phenomena.

2.2.2.1 Spectral Analysis

The concepts of the Fourier Transform (FT) and the Discrete Fourier Transform (DFT) are widely known and recognized[1]. Followed is a quick overview as shown by [8]: Consider a finite duration signal

$$\{x(n)\} \quad n = 0..N-1$$

Its z-transform is

$$X(z) = \sum_{n=0}^{N-1} x(n)\, z^{-n}$$

Evaluate at points on z-plane as

$$X(k) = X(z)|_{z_k} = \sum_{n=0}^{N-1} x(n)\, e^{-j\frac{2\pi}{N}kn}$$

We can evaluate N independent points

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j\frac{2\pi}{N}kn}$$

The above is known as the Discrete Fourier Transform (DFT) of $\{x(n)\}$ which is periodic in k, i.e. $X(k+pN) = X(k)$. Multiply both sides of the DFT by $e^{j\frac{2\pi}{N}km}$and add over frequency index k

$$\sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}km} = \sum_{n=0}^{N-1} x(n) \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}k(m-n)}$$

From which

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, e^{j\frac{2\pi}{N}kn}$$

[1] Refer to [7] for an in-depth analysis.

This is the inverse DFT

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, e^{j\frac{2\pi}{N}kn} \quad x(n+qN) = x(n)$$

That is, on the one hand the DFT assumes that we deal with periodic signals in the time domain, and on the other hand sampling in one domain produces periodic behavior in the other domain. Computation of the DFT requires for every sample N multiplications. There are N samples to be computed, i.e. N^2 time consuming operations. There are efficient methods available for computing the DFT such as the Fast Fourier Transform (FFT) which reduce the total computational effort to $\frac{N^2}{2}$and beyond.

By computing the N-point FFT we obtain spectral information of the input data with a normalized spacing of $\frac{1}{N}$Hz. It is important to point out however, that, as we shall see at a subsequent section of the report, there is a trade-off between a good time and a good frequency resolution, as dictated by the *Uncertainty Principle*. Another important aspect to point out is that a spectral analysis based on the raw linearly-spaced FFT-bins does not take into account important physiological properties of the human ear. As outlined by [9] "sensations (hearing, seeing, smelling, etc.) increase logarithmically as the intensity of the stimulus increases. Many experiment have (...) verified (at least approximately) this law and have led to the use of the decibel scale." In general the subjective listener response of loudness is a function of the intensity, frequency, and quality of sound. The overall frequency response of the human ear is highly non-linear, and has better frequency resolution at low frequencies [10]. This frequency-dependent sensitivity lead to the development of differing standards for sound-pressure-level meters such as A weightings (Figure 2.2), which deemphasize low-frequency sounds.

2.2.2.2 Cepstral Analysis

For acoustic signals there are many different parametric representation available, among them the Mel-Frequency Cepstrum Coefficients (MFCC) is the most widely used. MFCCs are coefficients that represent audio. They are derived from a type of

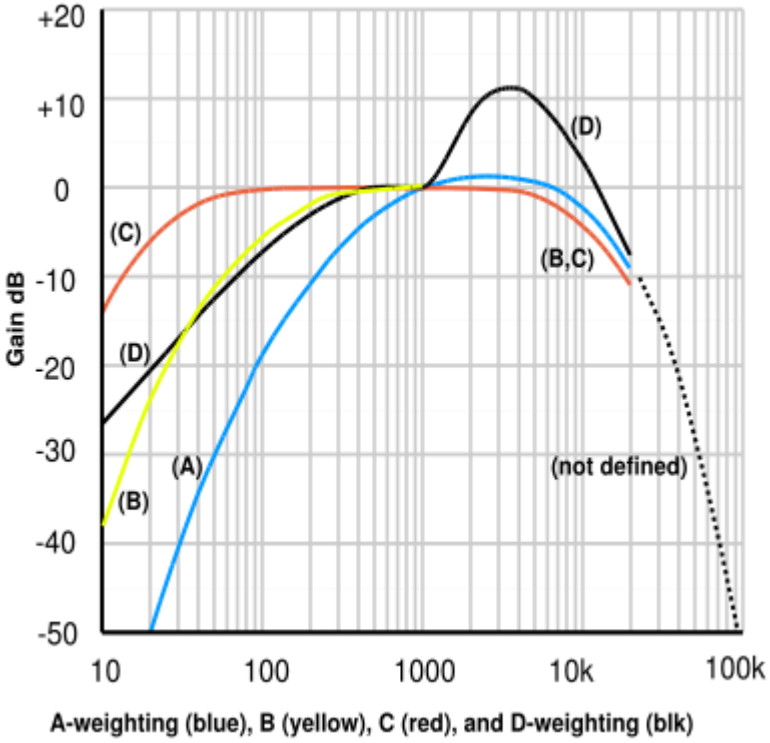

Figure 2.2: A, B,C and D-weighted SPL measurements.

cepstral representation. The cepstrum is defined as the Inverse Fourier Transform (IFT) of the log spectrum (Bogert et al.). The *mel* scale relates perceived pitch to frequency: it is linear at low f, and logarithmic at high f: $mel(f) = 2595 \log_{10}(1+\frac{f}{700})$ where f is in Hz. According to [11] the MFCCs are extracted in the following way (Figure 2.3):

1. First the audio data is separated into short segments (frames) of length 20 ms with 10 ms overlap (i.e. 50% overlap) between adjacent frames (ref. Figure

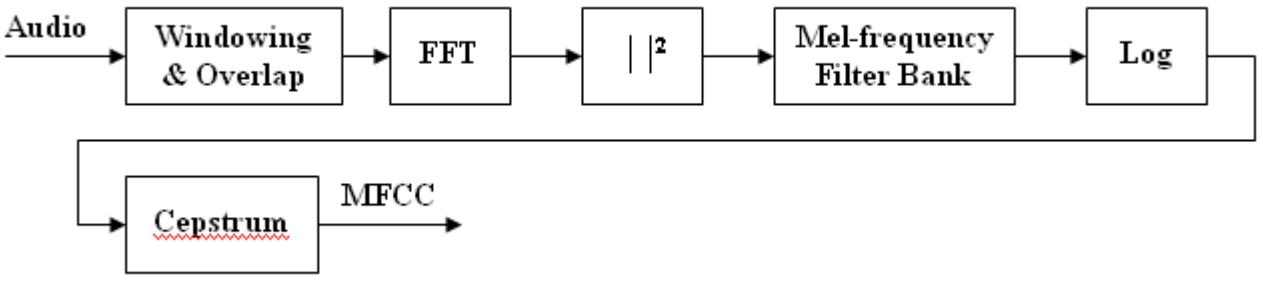

Figure 2.3: Block Diagram of a typical MFCC extraction algorithm.

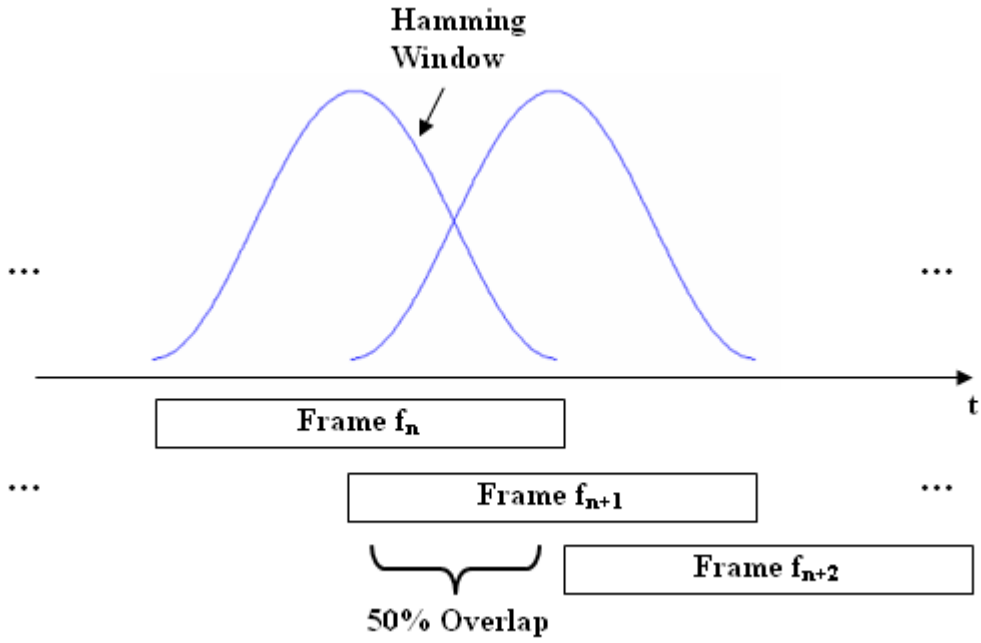

Figure 2.4: Frame blocking in a typical MFCC extraction algorithm.

2.4). For a sampling rate of 44100 Hz this would result in a frame length of 882 samples with 442 samples overlap. From a signal processing perspective a frame can be seen as the result of a waveform (speech or audio), multiplied by a rectangular pulse whose width is equal to the frame length. This will therefore introduce significant high frequency auxiliary components (noise) at the beginning and end points of the frame because of the sudden level changes from zero to the amplitude of the signal and vice versa. This edge effect can be reduced by applying a suitable window to each frame. The length of the window is set to equal the frame length. Typically a Hamming window is used as shown in equation (2.4).

$$Ham(N) = 0.54 - 0.46\cos\left(2\pi\frac{n-1}{N-1}\right) \tag{2.4}$$

where N is equal to 882 for a sampling rate of 44100 Hz and n is from 1 to N.

2. The FFT is then computed for each frame. To obtain a good frequency resolution, a 256-point FFT is used [11]. Because of the symmetry property only the first 128 coefficients need to be calculated.

3. The resulting spectrum of each frame is then filtered (smoothed) by a set of triangular shaped band-pass filters, and the power of each band is calculated. This is referred to as a mel filterbank which concentrates data values in the more significant part of the spectrum.

4. Finally the log of the mel spectrum is computed, along with the logged energy of each frame as one of the coefficients. Furthermore a range of dynamic parameters such as the first and second time derivatives can be calculated, to provide additional information about how the spectrum is changing with time. These delta coefficients are obtained from the following formula:

$$\triangle C_i(n) = \frac{\sum_{k=-N}^{N} kC_i(n+k)}{\sum_{k=-N}^{N} k^2}$$

The result of the above computation, is a variable sized feature vector, typically containing 39 elements in the context of speech processing [10]. Since MFCCs are usually used for speech recognition a pre-emphasis filter $1 - az^{-1}$ is employed at the beginning of the chain, where "a" is between 0.9 and 1. In this case the speech is first pre-emphasized to spectrally flatten the signal. However when dealing with audio signals, such as music, this stage is omitted. Alternatively when dealing with music, a high-pass filter can be employed in order to eliminate the 50/60 Hz hum originating from the AC electric power source.

2.2.2.3 Polyphonic Pitch Detection

Pitch Detection has been major subject of ongoing academic research throughout the years. The following section illustrates the methodology used for this project, while the suitability of a pitch detection algorithm for the time-alignment problem is discussed at a later chapter.

The transformation used for the input audio in the context of pitch detection is based on the Short-Time Fourier Transform (STFT) as shown in equation (2.5) where $x[n]$ denotes the signal and $w[n]$ the window. The preferred window when working with audio data is a Blackman-Harris window (2.6). By computing the STFT over blocks of length between 20-100ms over the length of the whole input

data one can gain valuable insights into the variation of the spectral components and their correlation. Using the STFT the global pitch of the various voices inherent in a polyphonic instrumentation can be analyzed over time; discrete events can then be localized as musical notes, opening the possibility of an additional interpretation of the audio signal into a musical score or a direct Audio-to-MIDI translation.

$$X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \tag{2.5}$$

$$w(n) = a_0 - a_1 \cos\Big(\frac{2\pi n}{N-1}\Big) + a_2 \cos\Big(\frac{4\pi n}{N-1}\Big) - a_3 \cos\Big(\frac{6\pi n}{N-1}\Big) \tag{2.6}$$

$$a_0 = 0.35875;\ a_1 = 0.48829;\ a_2 = 0.14128;\ a_3 = 0.01168$$

The pitch of the audio data is calculated in the following way according to [3]:

1. Select a sampling window from the incoming data.
2. Apply window for each block and perform Fast Fourier Transform (FFT).
3. Identify principal frequencies.
4. Identify the fundamental as a sub-multiple of the frequency of greatest amplitude.
5. If necessary, refine window and repeat.

After identifying the most important frequency components for each block, by the use of a local maximum search function a fixed threshold is employed to pick out the frequencies, f_i, with the largest amplitudes. This is almost certainly a true harmonic of the fundamental we are seeking. In other words, the fundamental frequency is $F = f_i/n$ where n is an integer. For each value of n from one to five we examine the spectrum to see how many frequencies are potential harmonics of F. A frequency is a potential harmonic if it is close to the ideal frequency of the harmonic. A pitch deviation of $\pm$5Hz is allowed for each candidate frequency. The resulting feature vector contains the values (in Hz) of the determined pitch(es) for each block-frame.

2.3 Alignment through Dynamic Programming

In the above section we have analyzed the process of extracting suitable parameters from the incoming audio data. It can be shown that two sequences of audio data can be represented in the following way [12]: Let X and Y be two sequences of elements x_m and y_m and of length M and N respectively:

$$X = x_1, x_2, \ldots, x_m, \ldots, x_M = \{x_m, m = 1, \ldots, M\} \tag{2.7}$$

$$Y = y_1, y_2, \ldots, y_n, \ldots, y_M = \{y_n, n = 1, \ldots, N\} \tag{2.8}$$

The alignment of an element x_{mk} with an element y_{nk} is defined by a couple $a_k = (m_k, n_k)$, $1 \leq m_k \leq M$ and $1 \leq n_k \leq N$. An alignment of one sequence with the other is defined by a sequence

$$A = a_1, a_2, \ldots, a_k, \ldots, a_K \tag{2.9}$$

such that the sequences $\{m_k, k = 1, \ldots, K\}$ and $\{n_k, k = 1, \ldots, K\}$ are non decreasing ($m_{k-1} \leq m_k, n_{k-1} \leq n_k$). If one considers a plane with indices m on the abscissa and n on the ordinate, A also defines a 'path' in this plane (ref. Figure 2.5).

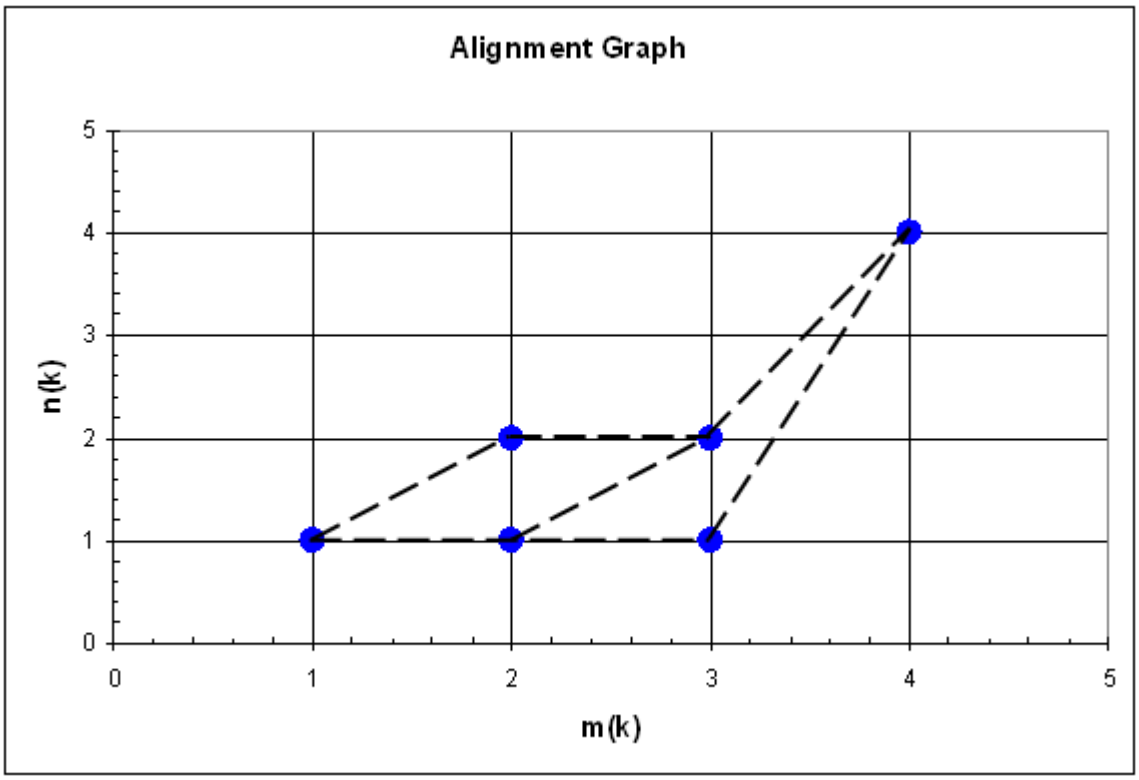

Figure 2.5: Possible Alignment of four segments betwen two sequences

As outlined in the early stages of the report we wish to estimate the time instant at which two portions of audio data are (near) equivalent so that we can interpolate smoothly between them. It is well known however that variation in the playing rate (in the case of a musical performance) or variation in the speaking rate (in the context of speech processing) causes a nonlinear fluctuation in a music/speech pattern time axis. We wish to eliminate this fluctuation, referred to as time-normalization. As outlined by [13] at an early stage, some linear normalization techniques were examined, in which timing differences between (...) patterns were eliminated by linear transformation of the time axis. Reports on these efforts indicated that any linear transformation is inherently insufficient for dealing with highly complicated fluctuation nonlinearity. In order to overcome these problems various algorithms have been presented over the years which make use of dynamic programming (DP). These algorithms aim at matching two sequences of audio data with a nonlinear time-normalization effect. By [13] timing differences (...) are eliminated by warping the time axis of one pattern so that the maximum coincidence is attained with the

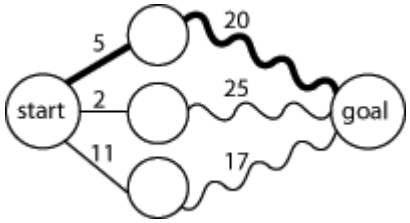

Figure 2.6: Finding the shortest path in a graph

other. Then, the time-normalized distance is calculated as the minimized residual distance between them. This minimization process is very efficiently carried out by use of the dynamic programming (DP) technique. By [14] the pattern matching problem can be formulated as an optimal path finding problem (optimal in the sense of minimum path dissimilarity) over a finite two dimensional grid (Figure 2.5).

The problem of finding an optimal path or optimal trajectory and its solution through DP, has been studied for decades[2] (Bellman et al.). Figure 2.6 depicts an example of the shortest path problem in a graph. A straight line indicates a single edge; a wavy line indicates a shortest path between the two vertices it connects (other nodes on these paths are not shown); the bold line is the overall shortest path from start to goal[3]. In the context of speech processing the DP technique outlined in this section is commonly referred to as Dynamic Time Warping (DTW). It will be shown throughout the report that DTW can be applied to traditional music signals as well as speech signals, albeit with a few modifications. The exact constrains of the DTW and the algorithm itself will be presented in the next chapter. Furthermore two distinct solutions will be presented over the course of the report. On the one hand a solution to the alignment problem based on a *static* time shift and on the other hand on a *dynamic* time warp. It will be shown that the time shift method is NP-complete providing however a very fine coarsed solution to the alignment problem with minimal error, while the time warp method can be solved in polynomial time. It will be shown that there is a trade-off between computational complexity and accuracy, and as a solution to that problem a *hybrid* model between a *static* time shift and a *dynamic* time warp will be presented.

[2]Please refer to [15] (especially chapters IV, V and VI).
[3]Source: http://en.wikipedia.org/wiki/Dynamic_programming

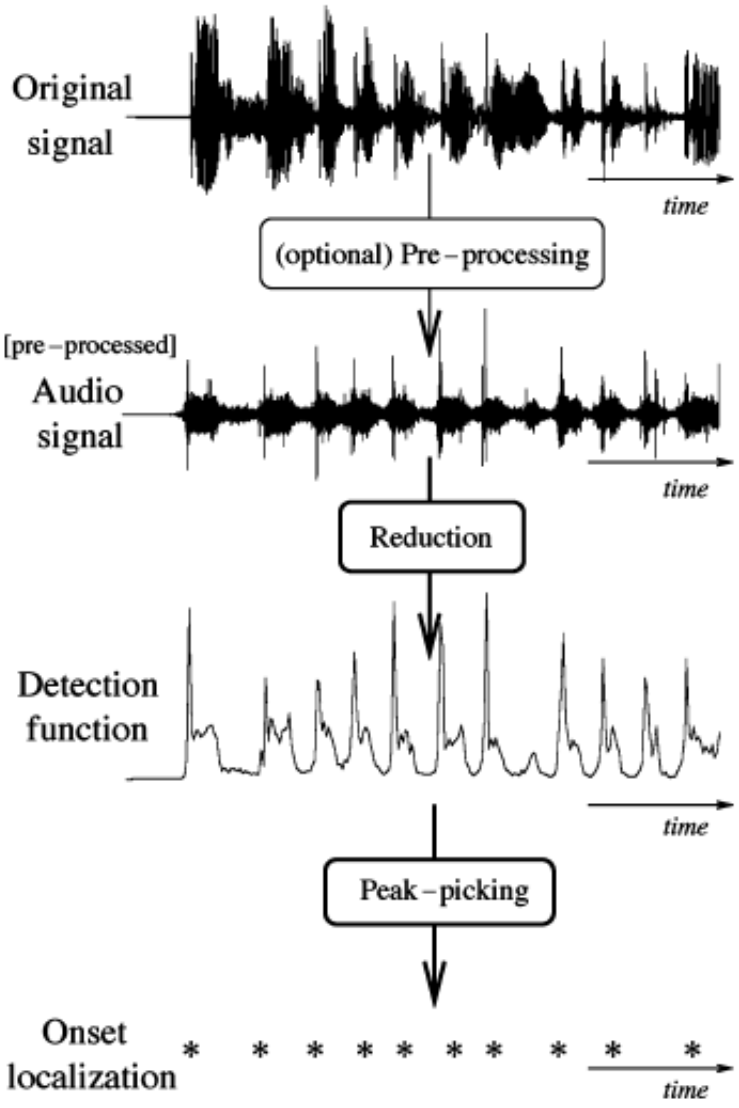

Figure 2.7: Time-Domain Analysis Flowchart

2.4 Overview of VST Framework

A few words on the Software Development Kit (SDK) on which the implementation of the above parts was initially specified. The open-source Steinberg VST Plug-ins SDK is a freely available (but subject to certain licensing conditions) software environment aimed at working with audio data, through providing a suitable layer of abstraction so as to facilitate development and deployment of an audio processing algorithm. It is based on C/C++ but is architecture and compiler independent. The preferred machine architecture chosen for the implementation of the project is an i386 compatible architecture and can be compiled in a Microsoft Windows or GNU Linux environment using an free/open-source compiler such as Microsoft Visual C++ or the GNU Compiler Collection (GCC) respectively. The simulation of the algorithm can be conducted on an open-source VST host, however it is preferably evaluated using commercially available software, based on the VST architecture such as Steinberg's own Cubase/Nuendo platform or any other 3rd party host.

2.5 Applications of Audio Alignment

What is outlined in this report can be applicable to many different scenarios where a time-domain alignment of various signals (such as audio) is sought-after. This project aims to find the optimal time-domain alignment point for two (or more) audio sequences specifically. It is thus aimed for the audio signal processing industry such as recording studios or TV/Radio-broadcast stations. Through the introduction of a novel way of automatically estimating the correct alignment point between various audio sequences, we wish to reduce the burden associated with the manual method as it is used today.

2.6 Summary

In this chapter we have introduced the notion of audio alignment and the various problems associated with it. We have presented how time domain alignment problems are dealt with today, and introduced the fundamental concepts needed in order

to automate this process. In the following chapter we will establish a mathematical framework for our alignment model, which will then be presented in greater detail in chapter 4.

Chapter 3

Audio Alignment

3.1 Notations

Symbol	Definition
$\mathbf{X}$	Signal measurement matrix
$\mathbf{Y}$	Signal measurement matrix
N_x	Total number of signal measurements for each frame
N_y	Total number of signal measurements for each frame
x	Signal parameter coefficient
X	Sequence of elements x_m
Y	Sequence of elements y_n
A	Warping function
D	Weighted summation of distances
N_c	Normalization coefficient

Table 3.1: Nomenclature.

3.2 Overview

Since the input audio data can be classified as a non-deterministic and non-stationary signal, we need to obtain a parametric depiction of the different properties encapsulated within the data. As shown in the previous chapter we analyse different parameters ranging from time domain to frequency domain representations with the ultimate target of finding a suitable classification of our signal at discrete time in-

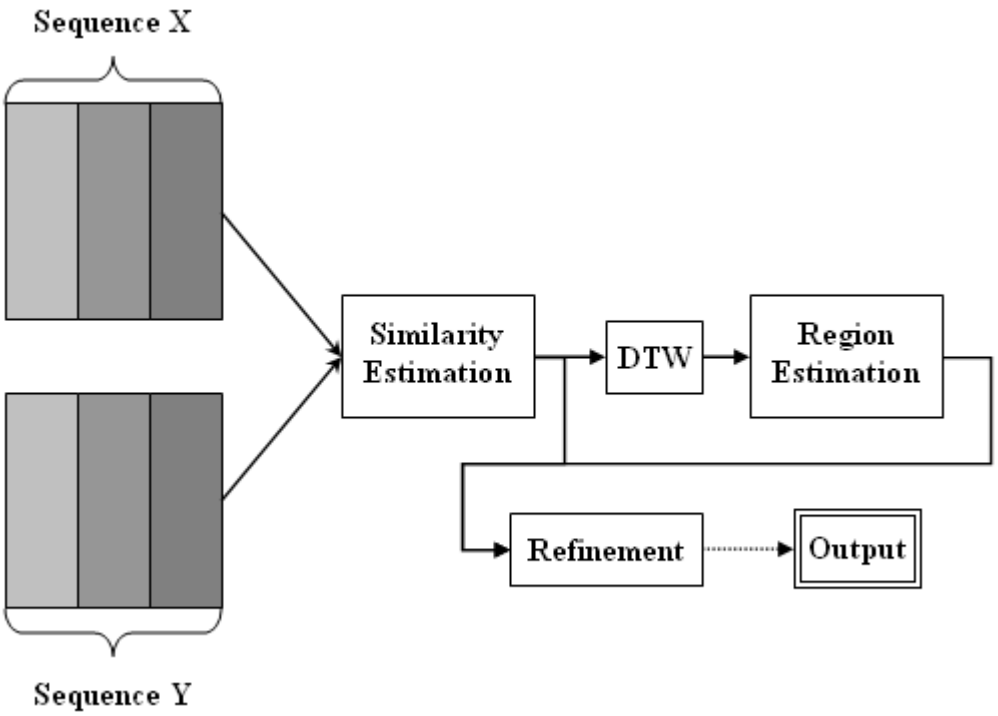

Figure 3.1: Overview of the processing blocks for audio alignment.

stances; i.e. we represent the input data as a sequence of data points at discrete time intervals. The creation of such a global feature vector will form the core data block of the input signal on which a variety of processing can take place. An overview of the different elements performing computation on the input data are depicted in Figure 3.1. Sequence X and Y represent the feature vector of two different audio files. In the following section we will analyze the inner workings of a variety of these processing blocks. We will start by mathematically defining the creation of the global feature vector for both sequences and then turn our attention to the problem of statistical transformation models for the signal parameters such as decorrelation and normalization. A distance measure will then be explored and the principles of Dynamic Time Warping (DTW) analyzed.

3.3 Feature Vector Extraction

By [16] let us define a signal measurement matrix for signal X as follows:

$$\mathbf{X} = \begin{bmatrix} x(0,0) & x(0,1) & \cdots & x(0,N_x-1) \\ x(1,0) & x(1,1) & \cdots & x(1,N_x-1) \\ \cdots & \cdots & \cdots & \cdots \\ x(N_f-1,0) & x(N_f-1) & \cdots & x(N_f-1,N_x-1) \end{bmatrix} \tag{3.1}$$

where $x(n,m)$ denotes the mth signal measurement at frame n, N_f denotes the total number of frames in the signal, and N_x denotes the total number of signal measurements for each frame. Similarly for signal Y:

$$\mathbf{Y} = \begin{bmatrix} y(0,0) & y(0,1) & \cdots & y(0,N_y-1) \\ y(1,0) & y(1,1) & \cdots & y(1,N_y-1) \\ \cdots & \cdots & \cdots & \cdots \\ y(N_f-1,0) & y(N_f-1) & \cdots & y(N_f-1,N_y-1) \end{bmatrix} \tag{3.2}$$

The signal measurement matrices $\mathbf{X}$ and $\mathbf{Y}$ contain all measurements of the two input signals for all time. N_x and N_y are of the same size for our analysis. Note that the signal measurement matrix usually contains a mixture of measurements: The short-term energy and its first time derivative, FFT coefficients and a set of cepstral coefficients along with delta coefficients $\frac{d}{dt}$. Again as shown by [16] N_x represents the dimension of the vector that is the composite of these measurements. From this point on we will consider these measurements as a group, rather than individually, and not refer to specific types of measurements. In some analyses, it is nevertheless useful that common measurements be grouped together in adjacent columns in $\mathbf{X}$ and $\mathbf{Y}$ (...). Additionally note that at this stage we mix quantities such as energy/power and spectral/cepstral coefficients together in the same feature vector. These parameters all exhibit a different numerical scale. If we compare two parameter vectors using a simple operator such as the Euclidean distance, the result will likely be dominated by the terms with large amplitude and variances, even though the true information may lie in the smaller amplitude parameters. As shown by Picone et al. the range and variance of the power term will be much larger than the range and variance of a cepstral coefficient. We will continue with a discussion on how we can achieve normalization as a remedy to this observation.

3.4 Distance Measures

Let us first define the term distance measure. According to [17] a distance measure should obey the following properties:

1. Nonnegativity:

$$D(\bar{x}_1, \bar{x}_2) > 0, \quad x_1 \neq x_2$$

$$D(\bar{x}_1, \bar{x}_2) = 0, \quad x_1 = x_2$$

2. Symmetry:

$$D(\bar{x}_1, \bar{x}_2) = D(\bar{x}_2, \bar{x}_1)$$

3. Triangle Inequality:

$$D(\bar{x}_1, \bar{x}_3) \leq D(\bar{x}_1, \bar{x}_2) + D(\bar{x}_2, \bar{x}_3)$$

The Euclidean distance measure is a metric which satisfies the above relations. For our application we make use of the factored form of the Euclidean distance:

$$D(\bar{x}_1, \bar{x}_2) = \quad \|\bar{x}_1 - \bar{x}_2\|^2 = \quad (\bar{x}_1 - \bar{x}_2)(\bar{x}_1 - \bar{x}_2)^\dagger = \quad \|\bar{x}_1\|^2 + \|\bar{x}_2\|^2 - 2(\bar{x}_1 \cdot \bar{x}_2) \quad (3.3)$$

The Euclidean distance thus is the sum of the magnitudes of the vectors minus twice the dot product. This representation is particularly important when devising a fast and efficient implementation in order to create a similarity matrix between two feature vectors.

As emphasized in the end of last section we need to provide for normalization and decorrelation between the different elements of the feature vector when choosing a simple distance metric such as the Euclidean distance. Thus for each element in the feature vector we derive the following expression for normalization:

$$\tilde{x}(n) = \frac{x(n) - \mu}{\sigma} \quad (3.4)$$

where the standard deviation is defined as:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2 - \bar{x}^2}$$

and the mean average μ as:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

There are other methods available for decorrelating and normalizing parameters used extensively in the context of speech processing and recognition. By [16] there is a straightforward method of decorrelating parameters in a statistically optimal sense for a multivariate Gaussian process. This method is outlined below but is not addressed in this project. Let us define a multivariate Gaussian probability distribution as

$$p(\bar{v}) = N[\bar{v}, \bar{u}_v, \mathbf{C}_v] = \frac{1}{\sqrt{(2\pi)^{N_v}\,|\mathbf{C}_v|}} e^{(-1/2)(\bar{x}-\bar{u}_v)\mathbf{C}_v^{-1}(\bar{x}-\bar{u}_v)^\dagger}$$

We will assume that our parameters obey this type of statistical model. We can compute a linear transformation that will simultaneously normalize and decorrelate the parameters. Let us define a transformed vector $\bar{y}$ as

$$\bar{y} = \mathbf{\Psi}(\bar{v} - \bar{\mu}_v)$$

where $\bar{v}$ denotes the input parameter vector, and $\bar{\mu_v}$ denotes the mean value of the input parameter vector. We define $\mathbf{\Psi}$ as a prewhitening transformation. (...) It can be shown that $\mathbf{\Psi}$ is given by

$$\mathbf{\Psi} = \mathbf{\Lambda^{-1/2}\Phi}^\dagger$$

where Λ denotes a diagonal matrix of eigenvalues, and Φ denotes a matrix of eigenvectors of the covariance matrix of $\bar{v}$. The eigenvalue and eigenvectors can be shown to satisfy the following relation:

$$\mathbf{C}_v = \mathbf{\Phi\Lambda\Phi}^\dagger$$

where $\mathbf{C}_v$ is the covariance matrix for v. Each element in $\mathbf{C}_v, C_v(i,j)$, can be computed as follows:

$$C_v(i,j) = \frac{1}{N_f} \sum_{m=0}^{N_f-1} (v_m(i) - \mu_v(i))(v_m(j) - \mu_v(j))$$

The above method [16] has been applied in the fields of speech processing. However the creation of the transformation matrix Ψ is associated with an additional cost, since it must be trained. This is done by collecting mean and covariance statistics across a large amount of (...) data. The implications of this will be discussed in the final stage of the report.

3.5 Dynamic Time Warping

Let us now focus on the mathematical framework of dynamic time warping as outlined by [13]. We will define a sequence of points referred to as a *warping* function which approximately realizes a mapping from the time axis of pattern $\mathbf{X}$ onto that of pattern $\mathbf{Y}$. For this we shall first simplify the notation of 3.1 and 3.2 as two sequences of elements x_m and y_n of length M and N respectively:

$$X = x_1, x_2, \ldots, x_M = \{x_m, m = 1, \ldots M\} \tag{3.5}$$

$$Y = y_1, y_2, \ldots, y_N = \{y_n, n = 1, \ldots N\} \tag{3.6}$$

The alignment of one sequence (warping function) with the other is defined by a sequence of points $a = (x, y)$:

$$A = a_1, a_2, \ldots, a_k, \ldots a_K \tag{3.7}$$

where by [12] the alignment of an element x_{mk} with an element y_{nk} is defined as a couple $a_k = (m_k, n_k)$, $1 \leq m_k \leq M$ and $1 \leq n_k \leq N$. A graphical interpretation would see patterns X (3.5) and Y (3.6) developed along a $m - n$ plane with indices m on the abscissa and n on the ordinate (ref. Figure 2.5). Thus sequence A (3.7)

can be seen as defining a *path* in this plane, i.e. a mapping from the time axis of pattern X onto pattern Y . This warping function coincides with the diagonal line $n = m$ if no timing difference between the two sequences exists. It deviates further from the diagonal line as the timing difference grows. We denote a measure of the difference between two feature vectors x_m and y_m as $d(m_k, n_k)$. Then, the global distance along an alignment between x_m and y_m is the weighted summation of distances along warping function A such that:

$$D(X, Y, A) = \sum_{k=1}^{K} d(m_k, n_k) \cdot w_k \tag{3.8}$$

where w_k is a nonnegative weighting coefficient. 3.8 is a reasonable measure for the goodness of warping function A and attains its minimum value when 3.7 is determined so as to optimally adjust the timing difference. The time-normalized distance between the two patterns X and Y is defined as follows:

$$\tilde{D}(X, Y, A) = \min_{F} \left[\frac{\sum_{k=1}^{K} d(m_k, n_k) \cdot w_k}{\sum_{k=1}^{K} w_k} \right] \tag{3.9}$$

where the denominator of 3.9 is employed as to compensate for the effect of K, i.e. the number of points on the warping function.

We have now established a mathematical framework for the definition of an alignment between two sequences. There are some restrictions to consider for the warping function however.

1. Endpoint constraints (also referred to as boundary conditions) force the points a_k to start and finish in the opposite diagonal corner of the rectangle (m, n):

$$a_1 = (1, 1)$$

$$a_K = (M, N) \tag{3.10}$$

2. Monotonic conditions:

$$m_{k-1} \leq m_k \quad and \quad n_{k-1} \leq n_k \tag{3.11}$$

3. Local Continuity Constraints force the possible steps in the warping path to adjacent cells:

$$m_k - m_{k-1} \leq 1 \quad and \quad n_k - n_{k-1} \leq 1 \tag{3.12}$$

As a result of these two restrictions, the following relation holds between two consecutive points:

$$a_{k-1} = \begin{cases} (m_k, & n_{k-1}), \\ (m_{k-1}, & n_{k-1}), \\ (m_{k-1}, & n_k). \end{cases} \tag{3.13}$$

4. Slope constraint condition: Neither too steep nor too gentle a gradient should be allowed for the warping function A because such deviations may cause undesirable time-axis warping. Too steep a gradient, for example, causes an unrealistic correspondence between a very short pattern X segment and a relatively long pattern Y segment. (...) Therefore, a restriction called a slope constraint condition is set upon the warping function, so that its first derivative is of discrete form. The slope constraint condition is realized as a restriction on the possible relation among several consecutive points on the warping function. The larger the effective intensity of the slope constraint P , the more rigidly the warping function slope is restricted. When $P = 0$ there are no restrictions on the warping function slope. When $P = \infty$ the warping function is restricted to the diagonal line $n = m$.

Since the criterion function in 3.9 is a rational expression, its maximization is an unwieldy problem [13]. If the denominator in 3.9

$$N_c = \sum_{k=1}^{K} w_k$$

(called normalization coefficient) is independent of warping function A, it can be put out of the bracket, while simplifying the equation as follows:

$$\tilde{D}(X, Y, A) = \frac{1}{N_c} \min_{F} \left[\sum_{k=1}^{K} d(m_k, n_k) \cdot w_k \right] \tag{3.14}$$

This simplified problem can be effectively solved by use of the dynamic programming technique. We will now consider two typical weighting coefficient definitions which enable this simplification. Note that for this project only the first form (symmetric) is considered.

1. Symmetric form:

$$w_k = (m_k - m_{k-1}) + (n_k - n_{k-1}), \tag{3.15}$$

   then $N_c = M + N$, where M and N are lengths of patterns X and Y, respectively [see 3.5 and 3.6].

2. Asymmetric form:

$$w_k = (m_k - m_{k-1}), \tag{3.16}$$

   then $N_c = M$. (Or equivalently, $w_k = (n_k - n_{k-1})$, then $N_c = N$.)

3.6 Summary

In this chapter we have provided a mathematical framework for the instantiation of a feature vector (3.1 and 3.2) representing a sequence of coefficients over time. An appropriate distance measure was derived (3.3) and the problem of normalization and decorrelation of the parameters adressed. Furthermore a model for the time-alignment of two series (3.5 and 3.6) was outlined and simplified so that it can be solved by use of dynamic programming. In the following chapter we will show how these different models can be implemented and simulated.

Chapter 4

Implementation

4.1 Overview

In this chapter we will turn our attention to the individual processing blocks of the algorithm, aimed at finding the optimal time-alignment point between two audio files. We will outline the implementation of each block, starting by the parsing of the input data, i.e. the creation of the global feature vector. We will then show how a similarity matrix between two sequences is obtained and finally derive two implementations of the dynamic time warping procedure. Additionally we will briefly outline how the pitch of a sequence of audio can be estimated. Note that the algorithm has been optimized for musical data, however because of a modular design and a range of different parameters explored, the implemented model can be just as well used for speech data. The algorithm has been devised in MATLAB with special emphasis placed on an efficient and fast implementation within the restrictions of the environment. We will start by going through each stage iteratively providing pseudo-code and graphical output wherever applicable.

4.2 Creating the Feature Vector

The feature vector forms the core data block and contains various coefficients representing the audio signal. The total number of signal measurements (that is the coefficients we compute) varies throughout our analysis, and is dependant on the

Elements	Type
1	Short-term energy
1	First time derivative of above
1..512	FFT/A-weighted coefficients
1..64	MFCC coefficients
1..64	Delta coefficients

Table 4.1: Coefficients used in the feature vector.

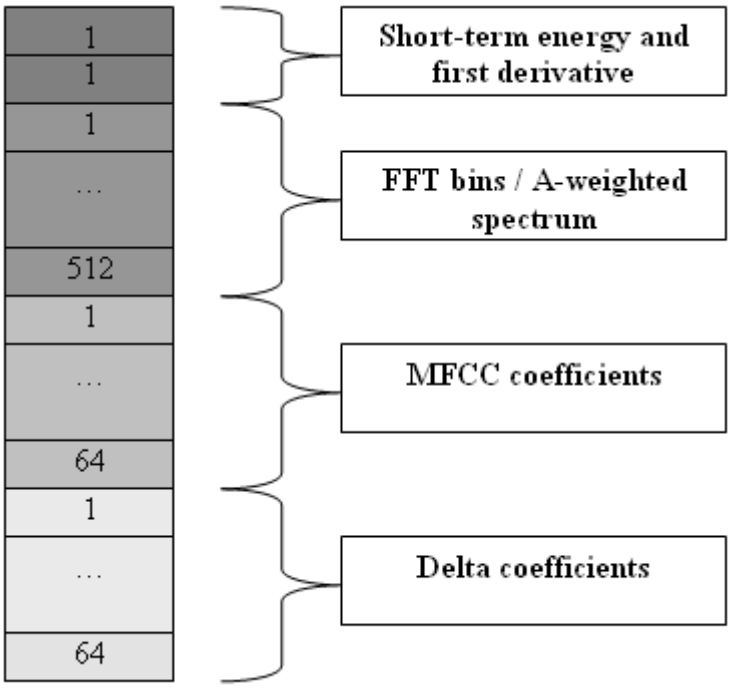

Figure 4.1: Snapshot of the elements of the feature vector.

input signal. It is sometimes useful to omit FFT coefficients completely in favor of a more dominant cepstral analysis (or vice versa). The size of the vector consists of typically around 62 elements: Two elements for the short-term energy and its derivative, around 32 FFT/A-weighted coefficients and another 13 MFCCs along with 13 delta coefficients. However as can be seen from Figure 4.1 the maximum size of the feature vector amounts to around 642 elements. Nonetheless a more realistic (and computationally feasible) figure of 62 elements can be used for most input data. The evaluation of the parameter count is conducted in the final stages of the report. We will begin by outlining the different stages in order to create the feature vector using Beethoven's "Fuer Elise" for Piano as our example input file.

The routine devised for creating the feature vector has the following parameters:

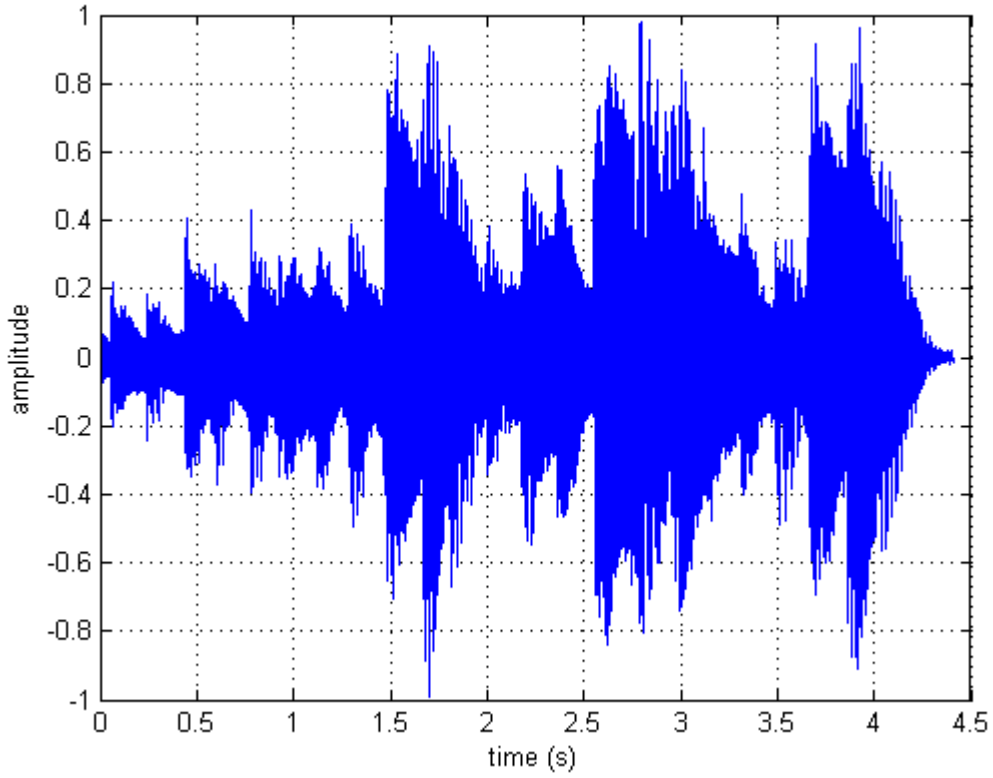

Figure 4.2: Example waveform of a *Beethoven* music file.

- The input filename.
- The cut-off frequencies for the initial filtering stage.
- The choice of a wide range of window types used for splitting the frames into overlapping blocks.
- The frame length (in ms).
- A switch type to either use the logged energy or the default case.
- The number of FFT/A-weighted coefficients to compute.
- The amount of MFCCs to compute.

We will now go through the different stages of the algorithm and provide graphical output wherever appropriate.

1. Read-in the audio file (typically in WAV format). A snapshot of the waveform is depicted in figure 4.2.

2. Normalization and filtering stage: Here we optionally pre-emphasize the audio file with an FIR filter $1 - az^{-1}$ in the case of a speech waveform or a simple high-pass filter in order to eliminate the 50/60 Hz hum originating from the AC electric power source. Additionally we can choose to *normalize*[1] the waveform to a maximum range.

3. Frame-splitting stage: In this part of the algorithm we split the input data into overlapping frames and window each frame appropriately.

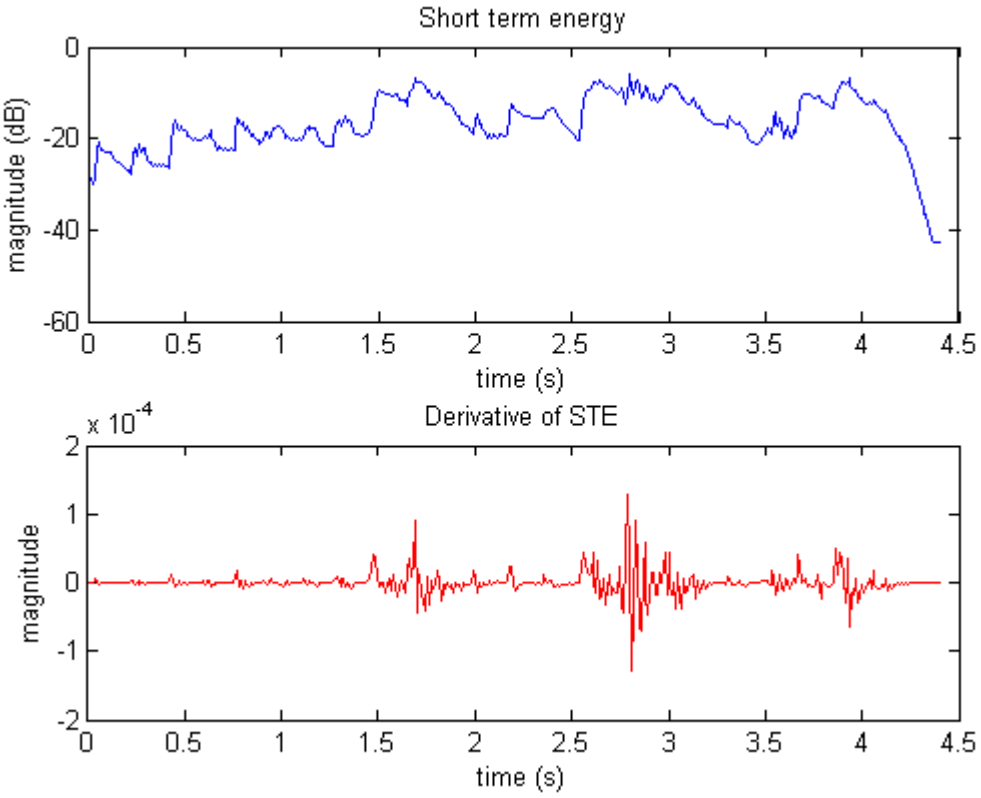

Figure 4.3: Short-term energy and first derivative of the waveform.

4. Next we compute the time-domain features of the audio file: The short-term energy and its first derivative as shown in figure 4.3.

[1] Audio normalization is the process of increasing the amplitude of a digital audio recording to a maximum peak level of 100% (0 dB).

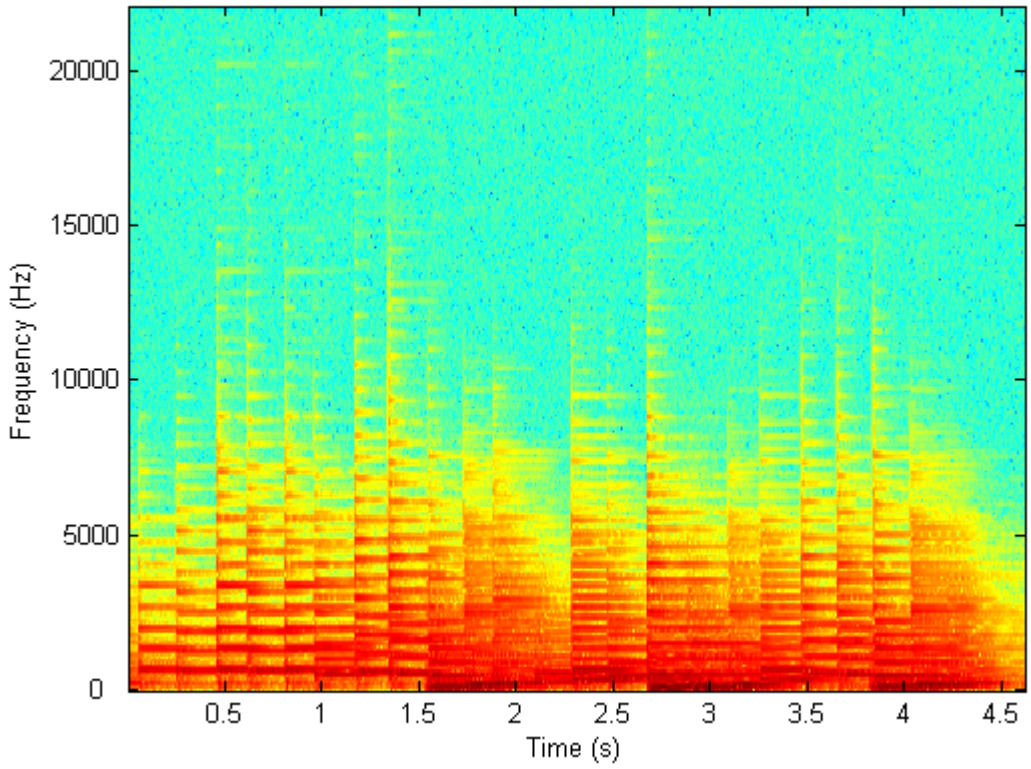

Figure 4.4: Spectogram of the input signal.

5. Embarking on the frequency-domain feature extraction we proceed with calculating the FFT/A-weighted coefficients (figure 4.4).

6. Finally we compute the MFCC coefficients and merge the results into one feature vector[2].

Having created the feature vector we can then optionally prefer to normalize or decorrelate the different parameters, since as emphasized in chapter 3 the parameters all exhibit a different numerical scale.

4.3 Estimating the Alignment

In this section we will outline the routine which aims at detecting the optimal time-alignment point between two sequences. This procedure takes the feature vectors of

[2]This routine is available from [18].

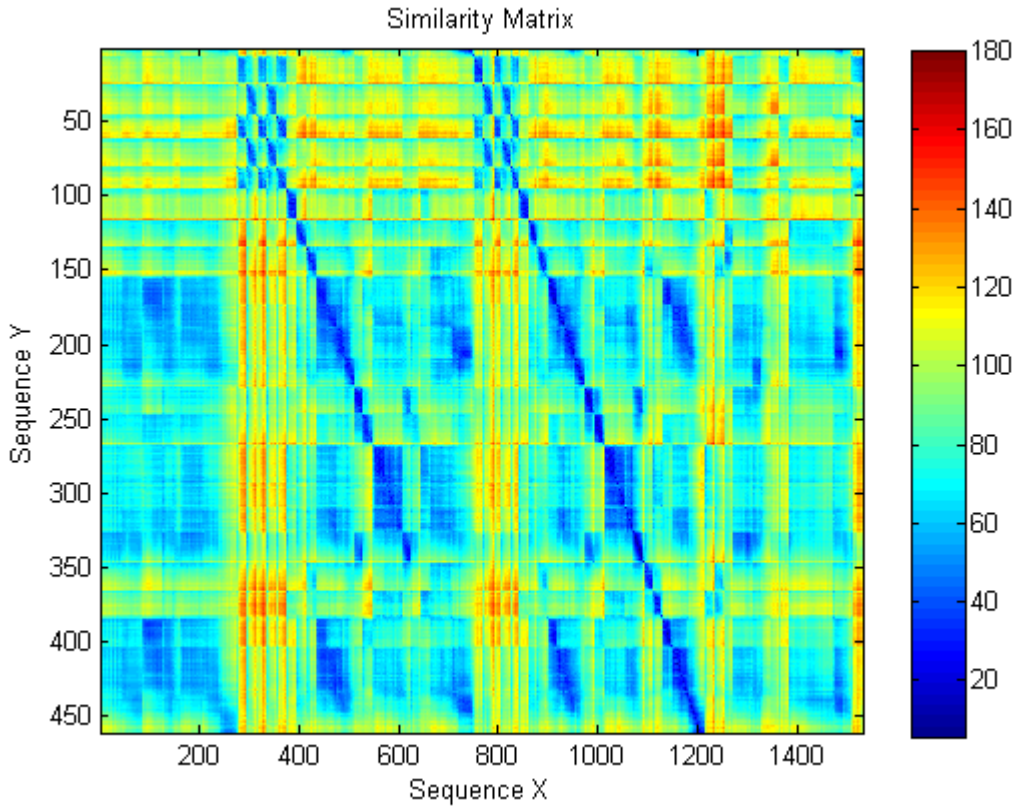

Figure 4.5: Similarity Matrix between two audio sequences.

two audio files and a marker region as input and returns the estimated alignment point. The different steps are outlined in the sections below.

4.3.1 Similarity Matrix

We compute the similarity matrix between the two files using the Euclidean distance. We make use of the factored form (expression 3.3) for an efficient and fast implementation. Figure 4.5 shows the *similarity* between two audio sequences. The similarity metric is proportional to the similarity of the frames. Values which approach 0 (dark blue region) depict similar frames while values which have a high value (dark red region) can be identified as dissimilar frames. As shown by [19] the resulting checkerboard pattern corresponds to segment boundaries, and off-diagonal lines signifying repeated phrases. The presence of the strong diagonal line indicates a good alignment between the two audio files. The next stage in our procedure attempts to find this alignment, on the one hand through dynamic time warping

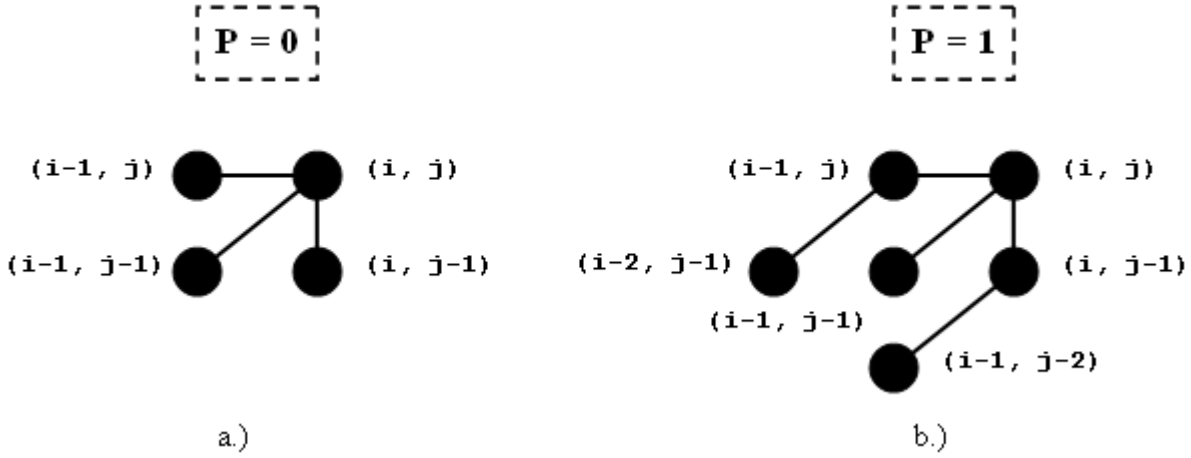

Figure 4.6: Path Direction (Slope Constraint) in DTW.

the two sequences and on the other hand through a linear time shift (exhaustive search).

4.3.2 Dynamic Time Warping

As mentioned above the aim of this stage is to evaluate the shortest path through the similarity matrix. By use of the dynamic programming (DP) technique we can efficiently compute this path. Having established different slope constraints in chapter 3 we will derive two models based on the symmetric form (expression 3.15) weighting coefficient definition. Let us first present these different slope constraint conditions.

First consider the case where no slope constraint is employed (i.e. $P = 0$). As can be seen in figure 4.6 a. the path through which the DP-matching algorithm progresses is not restricted. By [13] the basic algorithm for calculating the distance between two patterns X and Y is written as follows:

Initial condition:

$$g_1(a_k) = d(a_k) \cdot w_1 \tag{4.1}$$

DP-equation:

$$g_k(a_k) = \min_{a_{k-1}} [g_{k-1}(a_{k-1}) + d(a_k) \cdot w_k] \tag{4.2}$$

Such that the time-normalized distance is defined as:

$$D(X,Y) = \frac{1}{N} g_K(a_K) \tag{4.3}$$

It is implicitly assumed here that $a_o = (0,0)$. Accordingly, $w_1 = 2$ in the symmetric form (and likewise $w_1 = 1$ for the asymmetric form). We can thus derive an algorithmic representation for slope constraint ($P = 0$). Note that we will deviate from the nomenclature of chapter 3 and use i and j as our indices instead of n and m respectively.

Initial condition:

$$g(1,1) = 2d(1,1) \tag{4.4}$$

DP-equation:

$$g(i,j) = \min \begin{bmatrix} g(i,j-1) + d(i,j) \\ g(i-1,j-1) + 2d(i,j) \\ g(i-1,j) + d(i,j) \end{bmatrix} \tag{4.5}$$

Thus the time-normalized distance is given as:

$$D(X,Y) = \frac{1}{N} g(I,J) \tag{4.6}$$

where $N = I + J$. The DP-equation 4.5 must be recurrently calculated in ascending order with respect to coordinates i and j, starting from the initial condition at $(1,1)$ up to (I,J). The domain in which the DP-equation must be calculated is specified by $1 \leq i \leq I$, $1 \leq j \leq J$.

For the case where slope constraint $P = 1$ (figure 4.6 b.) is employed, the same basic conditions as outlined above hold. However now the second derivative of the warping function is restricted, so that the point a_k path does not orthogonally change its direction. This new constraint reduces the number of paths to be searched and

Algorithm 1 Pseudocode for the DP-matching algorithm for P = 0.

```
[I,J] = size(similarity_matrix);
for i = 1:I
 for j = 1:J
  d = g(i,j);
  [gmax, index] = min([g(i,j-1)+d, g(i-1,j-1)+2d, g(i-1,j)+d]);
  g(i,j) = gmax;
  slope(i,j) = index;
 end;
end;
//Commence Traceback
i = I;
j = J;
while i > 1 & j > 1
 if (index(i,j) == 1)
  j = j - 1;
 else if (index(i,j) == 2)
  i = i - 1; j = j - 1;
 else if (index(i,j) == 3)
  i = i - 1;
 X = [i,X];
 Y = [j,Y];
end.
```

thus reduces the computational complexity. The DP-equation $g(i, j)$ now becomes:

$$g(i,j) = \min \begin{bmatrix} g(i-1, j-2) + 2d(i, j-1) + d(i,j) \\ g(i-1, j-1) + 2d(i,j) \\ g(i-2, j-1) + d2(i-1, j) + d(i,j) \end{bmatrix} \tag{4.7}$$

The outcome of this section can be implemented in even the most basic programming environment. An example *pseudocode* similar to the one used in this project is shown in Algorithm 1. Having established the DTW procedure we obtain the alignment or warping path between two sequences. This path is depicted in figure 4.7. As can be seen, two regions of maximum coincidence between the two audio sequences are evaluated. These regions coincide with the strong diagonals lines of the similarity matrix of figure 4.5. We can formally define these regions as a sequence of points with near constant and maximum slope, i.e. a sequence of points or a path along the graph where the derivative approaches its maximum value. In order to find the alignment point between the two sequences it would be sufficient to trace back along

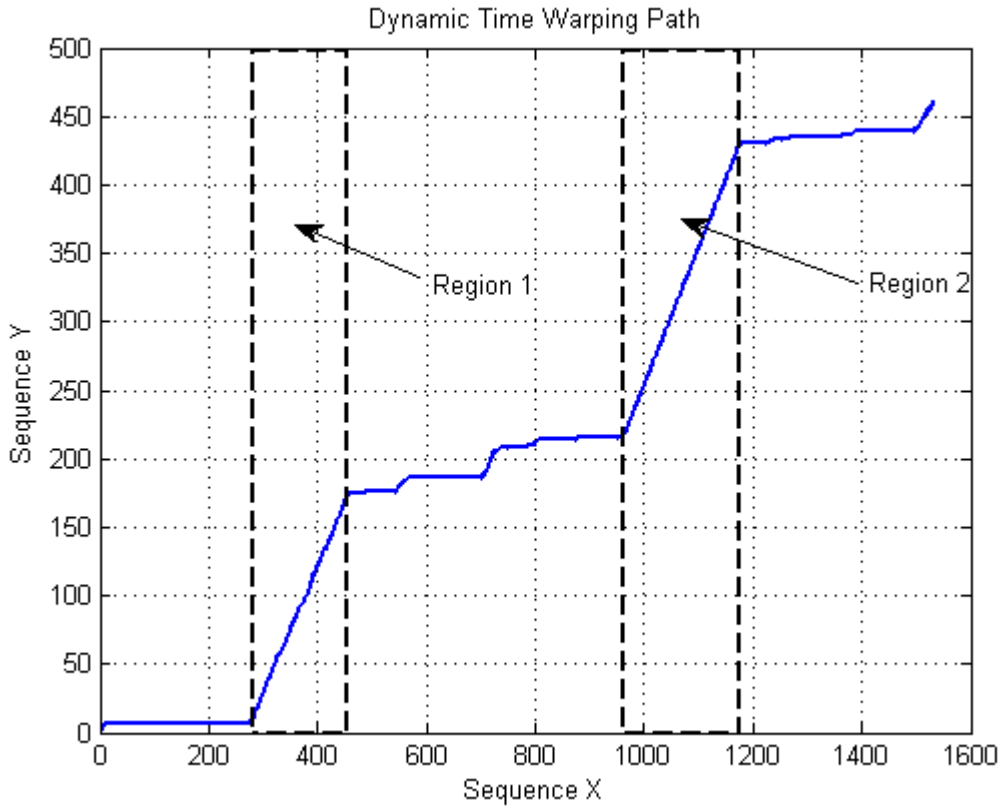

Figure 4.7: Alignment or warping path between two *Beethoven* audio-takes.

this slope and find the point where the path crosses the x-axis, however as pointed out in chapter 3 this methodology can provide unsatisfactory results, especially when dealing with large and complex data which is additionally corrupted by noise.

4.3.3 Estimating the regions of interest

In this section we will present how we can make use of the information obtained through DTW analysis to refine the regions of interest (ref. figure 3.1) by means of a gradient detection function. This gradient detection procedure locates the maxima of the first derivative of the slope in figure 4.7 and returns the regions of interest, i.e. the possible alignment points. As can be seen in figure 4.8 two global maxima are estimated from our example data. This is in accordance with what we have previously established (ref. figures 4.5 and 4.7). The sequence on the abscissa of the graph represents the *master* audio file sequence X on which we wish to align *slave* sequence Y. Note that sequence Y is in fact a subset of sequence X, i.e. it contains

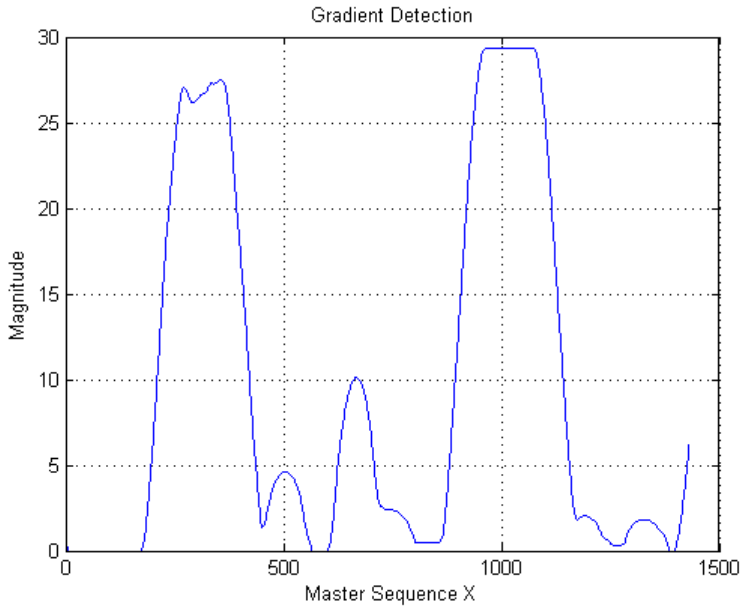

Figure 4.8: Gradient Detection function: Estimating the regions of interest.

a portion of the same musical data recorded at a different time. The reason why two global maxima are estimated is because the first few seconds of the introductory passage to Beethoven's "Fuer Elise" (which we have chosen in our example) are in fact repeated. Please note however that these passages are not *exactly* repeated twice, i.e. the second passage contains a variation towards the end. Audio file sequence Y contains the passage with this variation, and therefore as can be seen from figure 4.8 the algorithm correctly classifies the second global maximum as the one with the higher magnitude. Additionally note that this information is not known *a priori*, and that it is possible that when dealing with larger instrumentations or orchestrations there might be many sequences which are identically repeated. For this reason the algorithm demands an estimate, i.e. a marker region, to roughly locate itself into context. This marker is set by the user before running the algorithm, nonetheless this approximate region estimate needs to be only accurate within 2-4 seconds of the target alignment point. Having formally estimated the regions of interest for our alignment algorithm we can proceed to the final stage of the implementation.

4.3.4 Finding the target alignment point

As mentioned briefly above we could naively trace-back the gradient of the warping function to find the ultimate alignment point. In practice however this does not appear as straightforward as it may appear. We have mentioned in chapter 3 (ref. figure 3.1) that through DTW we obtain a region estimate, i.e. an interval of confidence localizing the target alignment point. This region estimate is used in the final block of the alignment algorithm where a time-consuming albeit local exhaustive search is employed to accurately determine the correct alignment point between the two audio sequences. The procedure is outlined below:

1. Initialize bounds of time shift to the regions estimated by DTW.
2. Perform local exhaustive search on the similarity matrix.
3. Compare estimated sequence with the *ideal* alignment sequence.
4. Return point of maximum coincidence.

Using this *hybrid* model of a global dynamic time warp and local exhaustive search we can locate the ultimate alignment point with great precision. The *ideal* alignment sequence is in fact the diagonal line $i = j$. The maximum coincidence between our estimate and the ideal path formally guarantees that we can give an estimate of the correct *ideal* alignment point between two sequences.

4.4 Pitch Detection

In this section we will briefly outline how to implement a pitch detection function for an arbitrary input audio file. The method has been formally defined in chapter 2.

1. First split data into overlapping frames of about 100ms with 50% overlap and window each individual frame. A Blackman-Harris is employed here preferably. We use the same routine for this part as for the feature vector extraction and therefore many different window types can be used.

Pitch (Hz) estimated every 50 ms.														
90	100	260	260	260	260	260	180	180	260	590	260	260	260	260
	260						260	220	590		590	590	590	590
							450	260						

Table 4.2: Polyphonic Pitch Estimation.

2. Compute FFT. Here the size of the FFT is set to equal the framesize or the next integer power of two.

3. Find peaks in the spectrum and sort them in descending magnitude order.

4. Discard FFT-bins which are below a predefined threshold.

5. Estimate the pitch of the remaining components. Here we can specify how many harmonics should be considered for each estimate. A suitable value has been empirically estimated to amount to five harmonic components for each fundamental frequency. Furthermore we accept each candidate pitch estimate as a harmonic component of the fundamental frequency if it deviates not more than 10Hz from the predicted value.

6. Sort the result by pitch (in Hz) and output the result as a matrix.

An example input file in which first the note C4 (260Hz) and then the note D5 (590Hz) is played on a Piano, produces the following result (ref. table 4.2).

4.5 Summary

In this chapter we have seen how the individual methods outlined in chapter 2 and formally defined in chapter 3 have been implemented. We have shown how to create a global feature vector from an input audio file, encompassing various parameters and coefficients. A similarity matrix based on the Euclidean distance between two points was then instantiated. We then showed how to obtain an estimate for the region of interest by use of the dynamic programming technique and more specifically through dynamic time warping. Additionally it was pointed out how to refine this region and obtain a more accurate target alignment point by use of an exhaustive search function. This *hybrid* model will be evaluated in the following chapter.

Chapter 5

Evaluation

5.1 Overview

As outlined in chapter 2 we wish to provide an automatic alignment method so as to reduce the decisional requirements of an external operator. This means that traditionally the time-domain alignment problems have been resolved through manual auditioning of the audio data and a subjective evaluation of the optimal alignment point. The importance of this observation leads us to the main problem when evaluating the success of this project. It means that we will evaluate an objective measure of the ideal time-alignment point and compare it against a subjective ideal value. We will therefore assume the manual estimation of the different time-alignment points as our ideal target value - although this is not entirely true. However the manual segmentation has been done by industry professionals and can be subjectively considered as perfect[1] or ideal.

In the following we will present the results of the evaluation of the algorithm established in chapter 4, conducted on various different types of input audio data. We will start the evaluation on two solo acoustic instruments, namely Piano and classical guitar and an example speech file. We will then increase the complexity of the data by considering a range of live recordings of acoustic (orchestral) instruments made in a concert-hall. On the one hand we will consider an arrangement of Baroque

[1] Perfect in this context means that we cannot distinguish any audible difference.

Error Range (ms)	Meaning
≤ 5	Perfect
≤ 10	Inaudible
≤ 30	Good
≤ 100	Useful

Table 5.1: A metric of success.

music for recorders by Henry Purcell, and on the other hand a classical arrangement for a string-section with horn by Wolfgang Amadeus Mozart. We will then present the different parameter variations considered. These are namely the size of the MFCC and FFT coefficients considered, along with observations about the effect of statistical normalization. First however we will give an overview of the metric of success and provide the appropriate ideal target values for each example data.

5.2 Target Values

Consider table 5.1 depicting the permissible error range for each alignment point obtained. An alignment estimation that deviates not more than 5ms from the ideal target value can effectively be considered perfect. Anything within 10ms is considered virtually inaudible and thus almost perfect. An alignment which deviates at most 30ms from the ideal value can be considered good and in some cases not audible. If the error is within 100ms then the result can still be considered useful however manual 'tweaking' by the user would be needed.

Source Data	Target Value (s)
Piano	7.483
Guitar	5.400
Speech	4.577

Table 5.2: First stage analysis target values.

Having established the constraints within which we can classify a successful alignment we can pursue with our final analysis. Table 5.2 shows the ideal target alignment values for the first stage analysis that we will consider, while tables 5.3 and 5.4 depict the ideal target values for the second stage *Purcell* and *Mozart* analysis respectively.

Purcell Take #	Target Value (s)
Take 1	16.753
Take 2	16.852
Take 3	20.958
Take 4	28.300

Table 5.3: Second stage *Purcell* analysis target values.

Mozart Take #	Target Value (s)
Take 1	16.912
Take 2	31.038
Take 3	7.637

Table 5.4: Second stage *Mozart* analysis target values.

5.3 Model Evaluation

In this section we will evaluate the behavior of our model for three distinctly different data-sets. We will examine the effect of a normalized feature vector as opposed to an un-normalized one, and additionally vary our parameters (namely FFT/MFCC coefficients and the effects of the logged energy evaluation) in order to obtain better accuracy with respect to the ideal values. We will provide five different *presets* for our first stage data and conduct a more thorough analysis for the second stage data.

5.3.1 First stage analysis

We will go through each of the preset files for our three different first stage audio files. Note that we always use a framesize of 20ms (with 50% overlap), since this is the lowest value we can use for achieving satisfactory results while permitting theoretically the most precise alignment.

First we will consider the effect of the different presets for the piano data. As

Preset #	Window type	FFT coefficients	MFCC coefficients	Framesize (ms)
1	Hamming	128	13	20
2	Hamming	64	39	20
3	Hamming	128	39	20
4	Hamming	0	256	20
5	Hamming	512	0	20

Table 5.5: Preset Table for first-stage analysis.

Preset #	Result (s)	Error (ms)	Normalized Result (s)	Normalized Error (ms)
1	7.460	23	7.460	23
2	7.460	23	7.460	23
3	7.460	23	7.460	23
4	7.460	23	7.460	23
5	7.470	13	7.450	33

Table 5.6: Piano audio data results.

Preset #	Result (s)	Error (ms)	Normalized Result (s)	Normalized Error (ms)
1	5.430	30	5.480	80
2	5.440	40	5.380	20
3	5.440	40	5.400	0
4	5.420	20	5.400	0
5	5.460	60	5.400	0

Table 5.7: Guitar audio data results.

can be seen from table 5.6 there is not much variance between the output. For most presets the result is the same. This is due to the simple nature of the input audio file, where only a limited amount of notes are played. Furthermore this example file is recorded in a studio environment with a very low noise floor. The overall minimum error for this type of input data is established with preset 5 (without statistical normalization).

Next we shall estimate the effect of our presets on the guitar data. The results are shown in table 5.7. For this type of data we can observe that preset 4 performs best. However the question that arises here is as to why this is the case. In fact the guitar data was recorded in a noisy environment. This explains on the one hand that the results vary throughout for each preset and on the other hand that the MFCC based analysis (as employed in preset 4) performs better here, since this method is more robust to noisy data. We can additionally observe that in presets 3-5, with statistical normalization of the feature vector, the error reduces to 0 ms. This would lead us to think that statistical normalization of the feature vector is preferable before computing the optimal alignment path. We shall leave this discussion however for a later stage before jumping to quick conclusions.

Finally we will compute the optimal alignment for the speech data. The results for this analysis are shown in table 5.8. In the case of an unnormalized feature vector

Preset #	Result (s)	Error (ms)	Normalized Result (s)	Normalized Error (ms)
1	4.520	57	4.500	77
2	4.480	97	4.530	47
3	4.480	97	4.530	47
4	4.580	3	4.530	47
5	4.560	17	4.530	47

Table 5.8: Speech data results.

we can observe how closely the parameter count is tied with obtaining a good result. The larger feature vectors obtained from presets 4 and 5 far outperform the other three cases. Especially the MFCC only analysis (preset 4) strikes out with an error of only 3 ms for the case of an unnormalized feature vector. This was expected, since the MFCC analysis was introduced in the context of speech processing and thus verifies the choice of this representation for speech signals.

In this section we gave a quick overview into the variation of the alignment results based on how many and more importantly *which* parameters to consider. Leaving out any considerations about statistical normalization for the moment, we can observe that the minimum error for each case of the example audio data, namely piano, guitar and speech reaches reasonable values (as dictated by table 5.1) of 13 ms, 20 ms and 3 ms respectively. It can be shown (at the expense of generality) that this error can be reduced further by adapting the parameters of the algorithm for different types of data. Accordingly this is why the analysis of this section is presented on the basis of five *preset* mixtures. Since we are dealing with a wide range of signals, no one combination of parameters is perfect for every type of data. Nonetheless for the three distinct types of test data preset 4 (MFCC only approach) gives on average the best results. It is important to note however, that as we shall also see in the next section, that opting for an only FFT or only MFCC (and not a combination of the two) based feature vector extraction seems to give better results.

5.3.2 Second stage analysis

In this section we will examine a range of live orchestral recordings with many instruments playing at the same time. We will provide an error analysis in graphical form for a wider range of parameters than explored in the first stage analysis. We will

begin with an investigation using a MFCC coefficient only approach and then present the results for an FFT only analysis. Finally we will combine both approaches and discuss the results. Note that since there is a plethora of data for this stage, we will only select a subset for our analysis, the complete data can be found nonetheless in the appendix.

5.3.2.1 Purcell Analysis

In this section we will give an overview of the results accumulated by means of an analysis on the different recordings of the arrangement by Purcell.

- MFCC Analysis

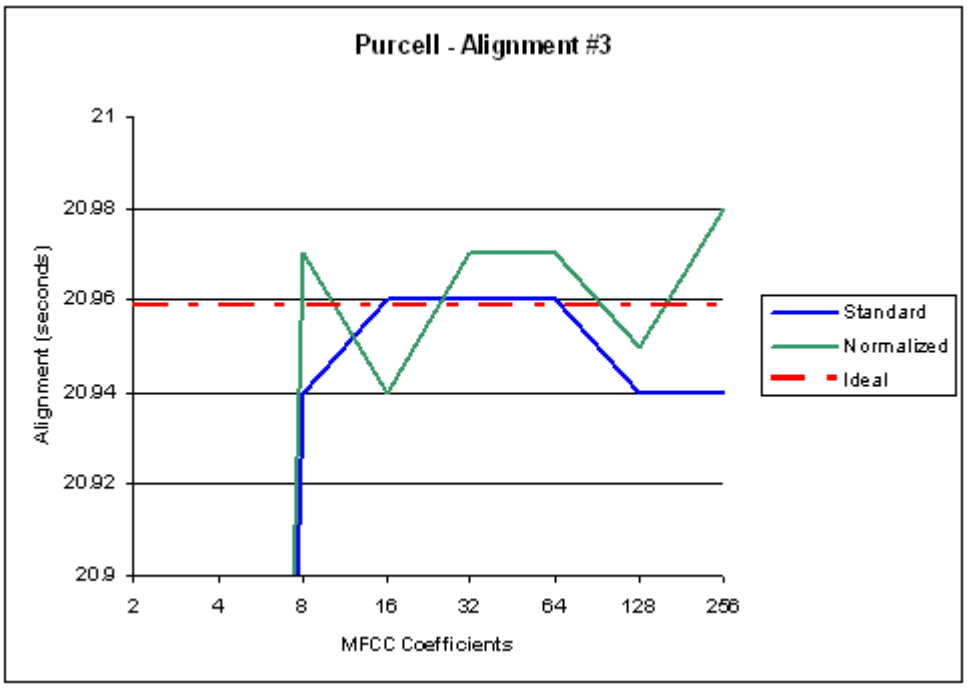

Figure 5.1: MFCC Alignment graph of *Purcell* take 3.

As can be seen in figure 5.1 we evaluate the optimal alignment point over a range of different MFCC coefficients. The blue line depicts the standard case, i.e. an alignment point estimation based on an unnormalized feature vector, while the green line shows the effect on the alignment by normalizing the feature vector. The dashed red line shows the value of the ideal case as shown in table 5.3. We can see that the blue line approaches the ideal value when more than 16 MFCCs are used for

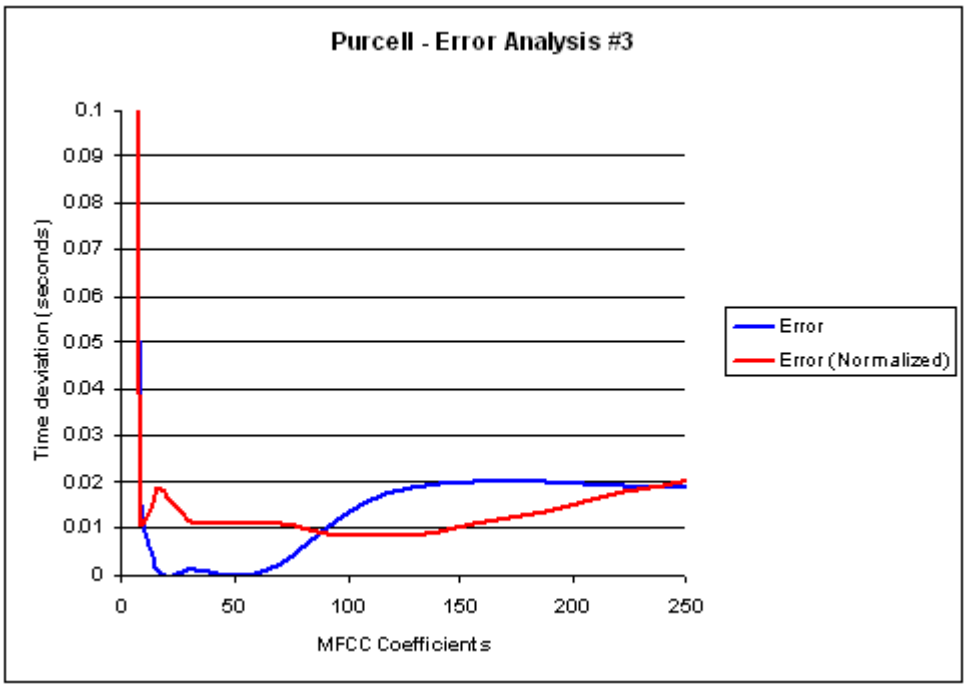

Figure 5.2: MFCC Error analysis for *Purcell* take 3.

Take #	Ideal (s)	Result (s)	Error (ms)
1	16.753	16.760	7
2	16.852	16.910	58 (8)
3	20.958	20.960	2
4	28.300	28.290	10

Table 5.9: *Purcell* - Error Table (64 MFCCs).

the feature vector. It deviates however from the ideal estimate if more than 64-128 MFCCs are computed. We can plot the error with respect to the ideal function as shown in figure 5.2. From this figure it is also clear that the error converges towards zero but then increases again. As we can see from table 5.9 the deviation from the ideal value is within very reasonable bounds when using 64 MFCC coefficients. There is one exception however, take 2 seems to have a much higher error than the other three takes. We can plot the alignment graph for this particular take (ref. figure 5.3). In this particular case we can observe a similar pattern than in figure 5.1. However here 128 MFCCs need to be evaluated to reduce the error within reasonable limits, i.e. 8 ms. Taking this new value as our result we can deduce that the average error for the four audio takes using 64 MFCCs amounts to 6.75 ms, which according to 5.1 is inaudible and an *almost* perfect alignment.

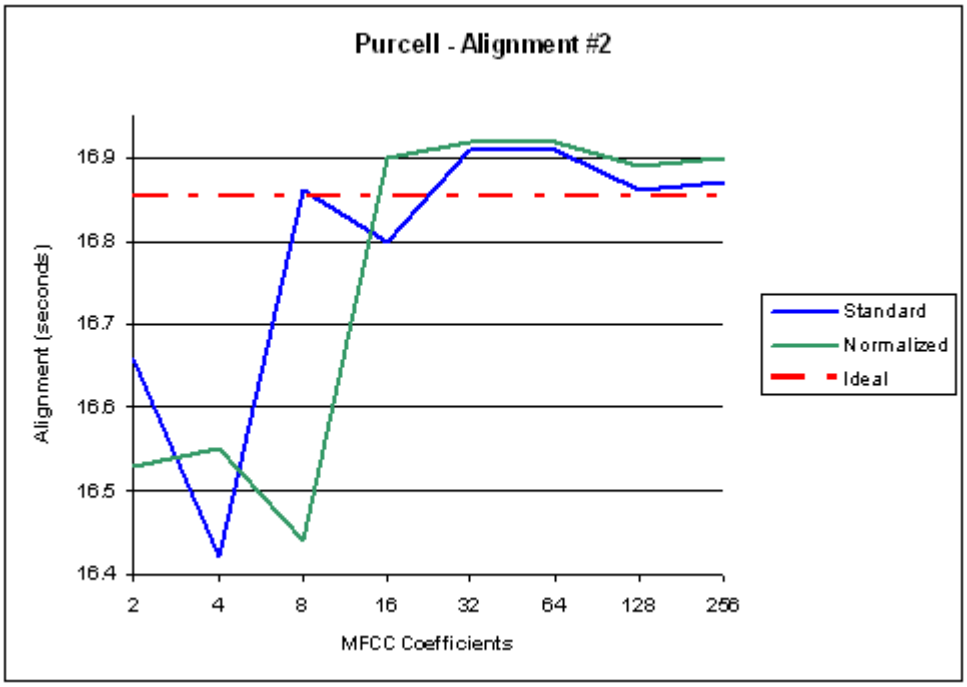

Figure 5.3: MFCC Alignment graph of *Purcell* take 2.

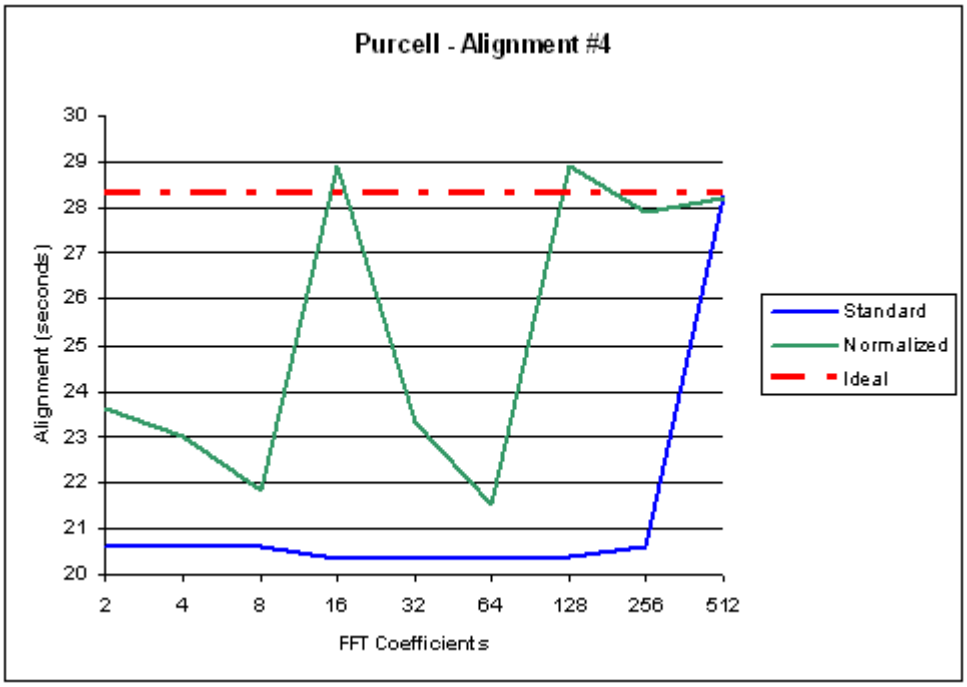

Figure 5.4: FFT Alignment graph of *Purcell* take 4.

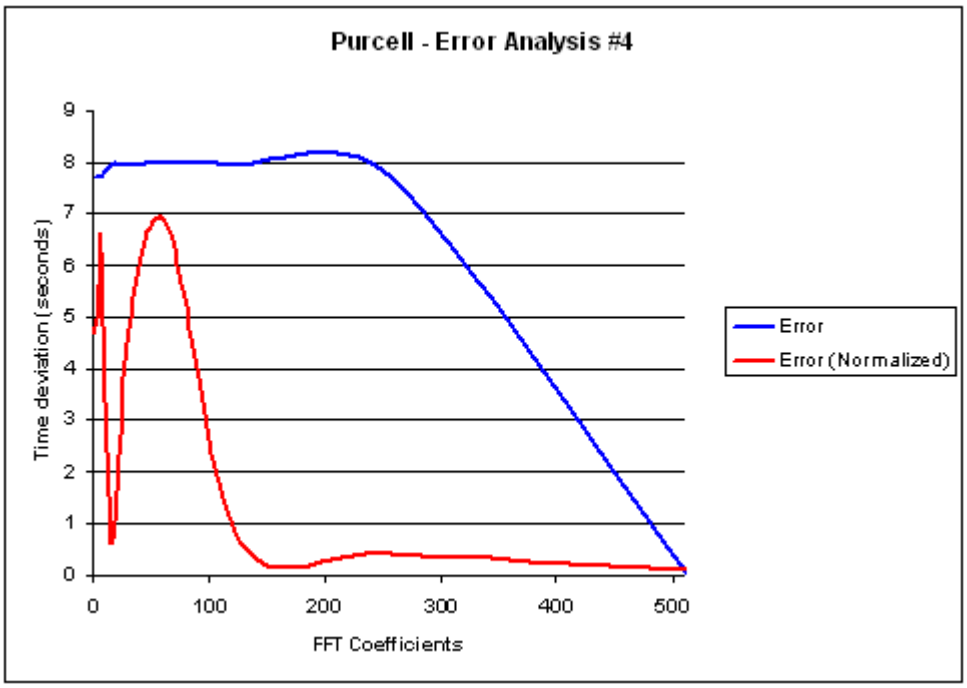

Figure 5.5: FFT Error analysis for *Purcell* take 3.

- FFT Analysis

Take #	Ideal (s)	Result (s)	Error (ms)
1	16.753	16.780	27
2	16.852	16.850	2
3	20.958	20.950	8
4	28.300	28.270	30

Table 5.10: *Purcell* - Error Table (512-point FFT).

Next we will focus on an FFT based analysis in order to see if a similar pattern to the analysis above is observable. As can be seen in figures 5.4 and 5.5 the estimated alignment point converges towards the ideal value when the size of the FFT is set to 512. This example is a bit of an extreme case, and as can be seen in the appendix, for other takes the alignment can converge earlier towards the ideal value. However for all four takes the ideal estimate is reached when the size of the FFT is 512 (table 5.10). The alignment results given when a 512-point FFT is used can be deemed as good (table 5.1) and in some cases literally inaudible (takes 2 and 3). The average error in this analysis amounts to 16.75 ms.

- MFCC/FFT Analysis

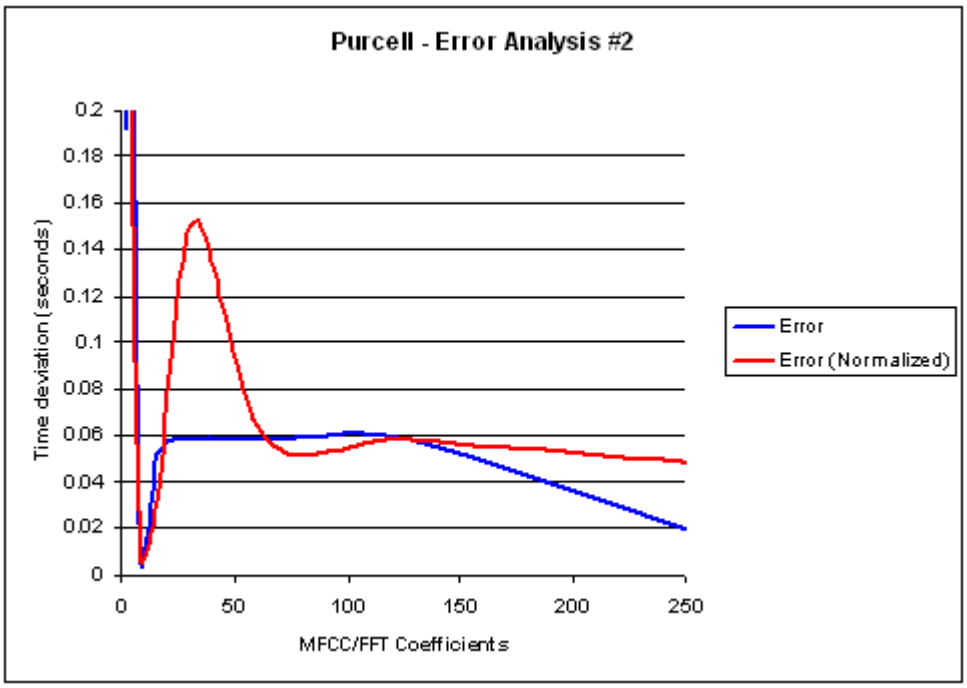

Figure 5.6: MFCC/FFT Error analysis for *Purcell* take 3.

Take #	Min. Error (ms)	MFCC/FFT coefficients
1	100	32
2	2	32
3	18	256
4	7	64

Table 5.11: *Purcell* - Minimum error table for MFCC/FFT coefficients.

We will now briefly turn our attention towards an alignment estimation based on both MFCC and FFT coefficients. As we can see from figure 5.6 the error does seem to converge towards a minimum value, however it stays well above 50 ms for the most part and as can be seen from figure 5.7 well above 150 ms. It is nonetheless true that for some takes the error can amount to much smaller values. As can be seen from table 5.11 the minimum deviation from the ideal value is achieved with a different number of MFCC/FFT coefficients throughout. Based on the results of this analysis we can say that using a mixture of MFCC and FFT coefficients seems to give unpredictable results. This is easily explained by the fact that MFCC and FFT coefficients virtually represent the same frequency domain information. It is therefore important

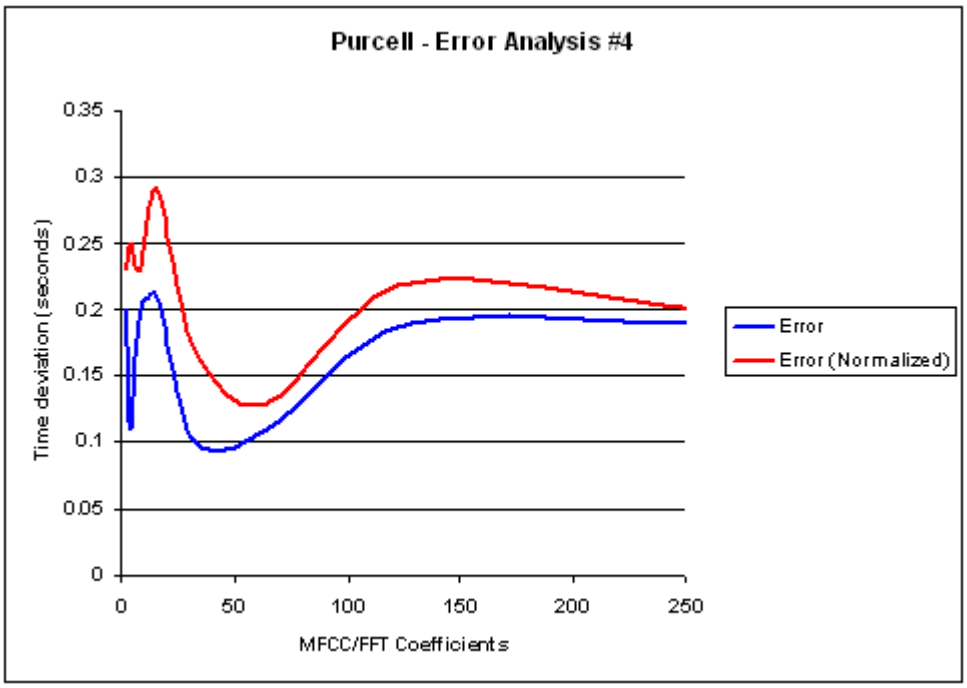

Figure 5.7: MFCC/FFT Error analysis for *Purcell* take 4.

to use only one representation of the frequency domain and not both methods simultaneously.

5.3.2.2 Mozart Analysis

In this section we will consider three audio takes from the performance based on the arrangement by Mozart.

- MFCC Analysis

As can be seen in figure5.8 the estimate of the alignment point quickly converges towards the ideal value. The results of the MFCC based analysis exhibit a similar behavior for this test data as in the previous case. However note that for the Mozart arrangement a bigger parameter value needs to be estimated. Instead of 64 MFCCs as shown in the case of the Purcell data. we need to compute around 128 MFCCs to obtain an optimal estimate. The explanation for this is based on the fact that this data exhibits a larger spectral range. The recorders used for the Purcell interpretation are more limited from a spectral point of view. For our test data in this case however more instruments (and of different type) are used. This explains

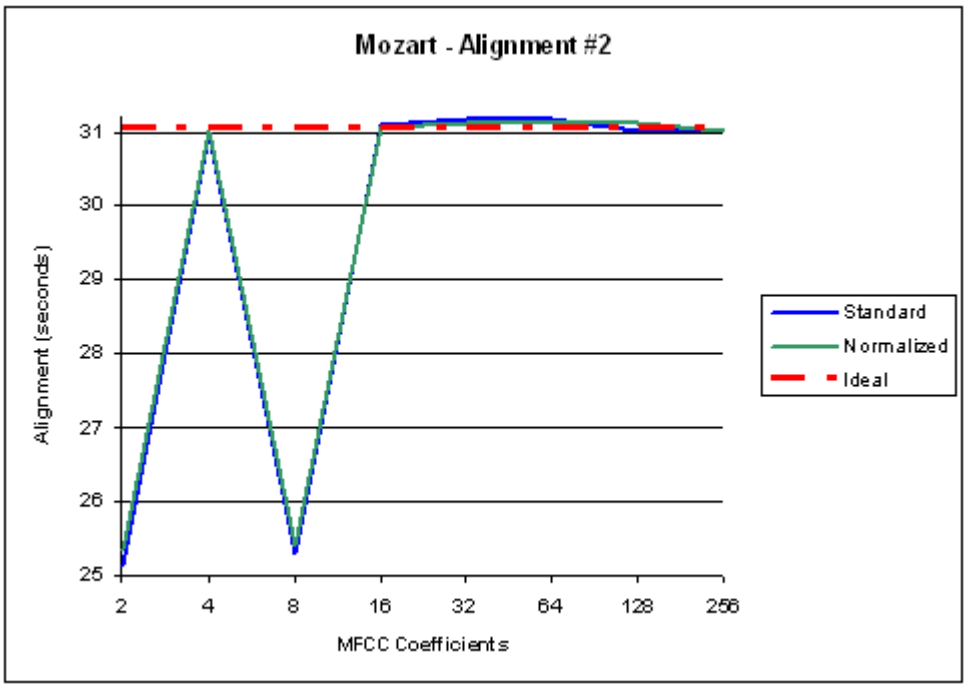

Figure 5.8: MFCC Alignment graph of *Mozart* take 2.

Take #	Ideal (s)	Result (s)	Error (ms)
1	16.912	16.920	8
2	31.038	31.010	27
3	7.637	7.640	3

Table 5.12: *Mozart* - Error Table (128 MFCCs).

why we need to increase the parameters of our model. Table 5.12 shows the error estimates for each individual audio take when employing 128 MFCCs. The average error amounts to around 12.6 ms for this configuration. This is an almost inaudible timing difference, and proves to be a satisfactory result for this kind of complex data.

- FFT Analysis

From figures 5.9 and 5.10 we can deduce that we need to compute 256-512-point FFT coefficients to obtain a good alignment. This is in accordance with what has been established earlier for the Purcell data. Table 5.13 shows the error estimates for a 512-point FFT evaluation. Take 2 stands out as larger than the other two. This is similar to the result obtained from table 5.12. It is difficult to give an explanation for this, but it is either a passage in the musical data which the algorithm cannot

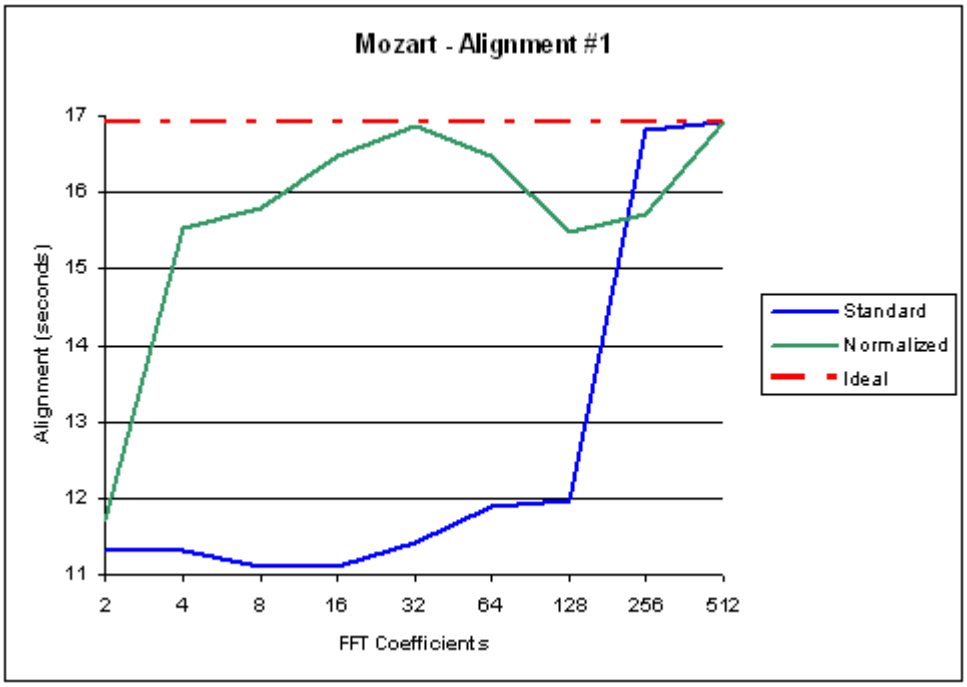

Figure 5.9: FFT Alignment graph of *Mozart* take 2.

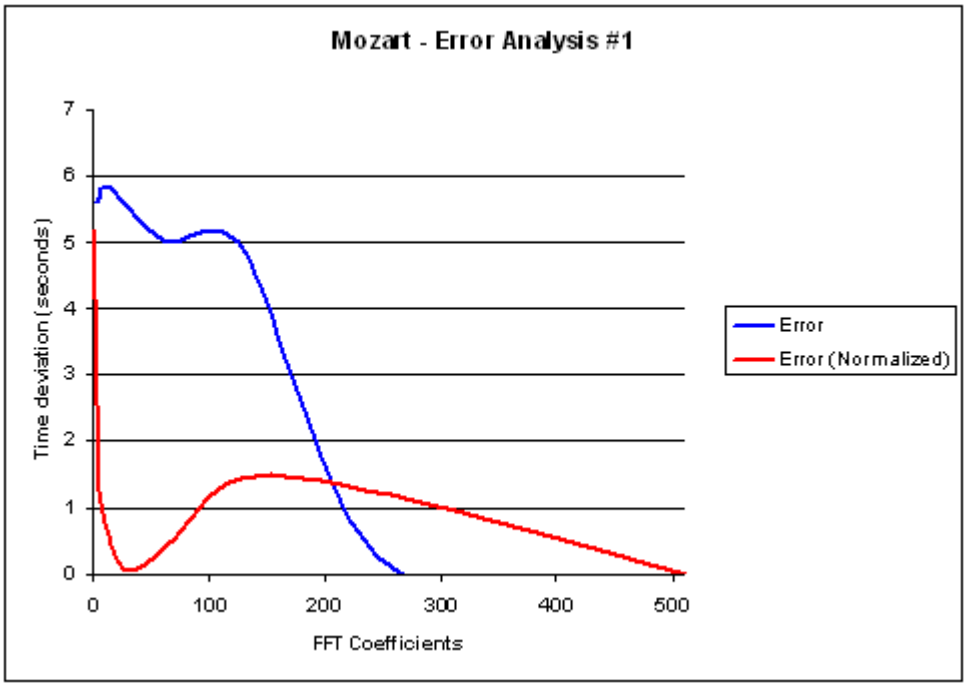

Figure 5.10: FFT Error analysis for *Mozart* take 1.

Take #	Ideal (s)	Result (s)	Error (ms)
1	16.912	16.920	8
2	31.038	31.130	92
3	7.637	7.640	3

Table 5.13: *Mozart* - Error Table (512-point FFT).

track properly (i.e. a boundary segment or a point at which a substantially low frequency sound is occurring), or a marker which has been manually placed in the wrong place to start with. By looking at the musical score we cannot detect any anomalies, thus we can deduce that it is either a wrongfully placed initial marker or just a local inconsistency with our evaluation. Nonetheless the analysis on the three regions of the Mozart arrangement, using an FFT based analysis, seems acceptable (with the exception of alignment #2) producing an average error of about 34.3 ms.

Having established the superiority of an exclusive FFT or MFCC evaluation in the previous section we will not include data for the mixed case.

5.4 Overview of Results

We will now give an overview of the results so far and rank each estimation obtained according to table 5.1.

Starting by the first stage analysis we can observe that using only a limited amount of different features (5 presets) can provide satisfactory results. However as pointed out earlier, we need to adapt the parameters of the algorithm to the different audio data. Detailed a priori knowledge about the type of data used, needs to be considered before the optimal alignment point can be evaluated. We showed that in the case of Piano data the error reduced to 13 ms when using preset 5. This depicts an almost inaudible timing difference. On the other hand the noise corrupted classical guitar data had a minimum error of 20 ms when using preset 4 which we can objectively rank as a good alignment. Using an MFCC based analysis we showed that for the speech data the error amounted to only 3 ms, which is a perfect alignment.

In the second stage analysis we first established the superiority of the MFCC and

Audio Data	Ranking
Piano	Good/Inaudible
Guitar	Good
Speech	Perfect
Purcell #1	Inaudible
Purcell #2	Inaudible
Purcell #3	Perfect
Purcell #4	Inaudible
Mozart #1	Inaudible
Mozart #2	Good
Mozart #3	Perfect

Table 5.14: Ranking of each test data.

FFT based evaluations respectively and deemed the combined MFCC/FFT method as inferior. For the Purcell test data employing 64-128 MFCCs we evaluated an error of 7, 8, 2 and 10 ms for takes 1, 2, 3 and 4, ranking the timing difference for each case as virtually inaudible. Using similar parameters for the Mozart data we evaluated an error of 8, 27 and 3 ms for takes 1,2 and 3 respectively. Table 5.14 depicts the ranking for each test data considered. Out of the ten different alignments four can be classified as *good,* three as *inaudible* and another three as *perfect.*

5.5 Additional Considerations

Additional considerations which have not been fully addressed in the above sections include the following:

- Effects of using different frame-lengths.
- Computing the logged energy as opposed to the *normal* energy.
- Statistical Normalization of the feature vector.
- Window types used.
- Run-time behavior (execution speed etc.)

We will now go through each of these different considerations starting with the frame-length (framesize). In the evaluation of the algorithm we used a framesize of 20 ms

(with 50% overlap) throughout. Ultimately one should evaluate the effects observed when changing the frame-length, however this was not done in this evaluation. The reason for this is that using a frame-length of more than 20 ms increases the minimum theoretical value of the maximum error estimation. Because we are working on a frame-precision level, and not on a sample precision level we can only provide a certain theoretical precision range. For example if an optimal alignment point is defined at time instant 4.005 s and we are computing an alignment every 10 ms (as used in this project) then we can either choose to provide an estimate at time 4.000 or 4.010. The minimal value of maximum error thus becomes 5ms. If we increase our framesize then this theoretical min-max error increases as well.

Computing the logged energy as opposed to the normal energy didn't prove to make any difference. The alignment results are exactly the same for each case. The reason for this being that we are focusing on a more frequency-domain based method. However as will be shown at a later stage if we were to weight each type of parameter differently, we could emphasize the time-domain features. In this case the logged energy could theoretically play a more central role.

We have included the estimation of the time-alignment point based on statistical normalization in our plots in this chapter. However the results for statistical normalization are inconclusive. Sometimes they give similar results than when not using any normalization at all, rarely better, and most times worse and unpredictable results. This is possibly due to the fact that the normalization was either not applicable in our case or that it was not properly implemented.

The different window types used can theoretically prove to enhance our results. However during the implementation and evaluation stages throughout the course of this project their effect has been deemed negligible. Therefore we have outlined results based only on the *Hamming* window.

Run-time behavior is an important aspect to consider when developing an algorithm for signal processing. However the environment used for the implementation (MATLAB) is not a good measure of performance, and therefore we have excluded any run-time considerations in this evaluation.

5.6 Summary

In this chapter we have evaluated the programming model outlined in chapter 4. We first defined a measurement metric by which we could objectively define and interpret our results. We then derived an optimal parameter set for each different input data. We verified this parameter set and then classified our results according to the measurement metric developed. Finally additional considerations were enumerated and possible short-comings noted. In the next chapter we will summarize what has been done and where future enhancements can be made.

Chapter 6

Summary

This chapter summarizes the material presented in this report and the results of this investigation. First we will outline the results we have obtained throughout the course of this project and analyze potential topics and areas of further research.

6.1 Summary of Results

We embarked on this project with the aim of finding an automatic algorithmic representation for the solution of the time-domain alignment problem. First the problem was defined and the existing manual solution presented and analyzed. A mathematical framework was then devised based on academic work in related areas. This mathematical background was used in order to obtain a parametric representation of non-stationary audio signals. We then derived a framework for the alignment of two distinct sequences and showed that the resulting alignment problem could be solved by use of the dynamic programming technique. We devised two forms for computing the dynamic time warping path between these two audio sequences and implemented a two stage model within the MATLAB programming environment. The outcome of this model was then estimated and its correctness confirmed by evaluating it against existing manually segmented data. In the following section we will describe the potential weaknesses and short-comings of the model and provide an overview on possible areas of further research.

6.2 Suggestion for Further Research

First and foremost it was initially assumed that the obtained model would be implemented as a VST plugin. However because of a recent upgrade to the VST SDK, support for a few vital components seemed to have been dropped. In particular the Offline-VST Interface, which was present in version 2.3 but removed in 2.4 (current version), would greatly facilitate the development of the algorithm outlined in this report. By coding a new host or by modifying the source of the VST SDK one could effectively implement an algorithm similar to the one used in this project yielding direct deployment on a vast variety of target computer architectures and operating systems.

One of the problems not thoroughly addressed in this project is that of statistical normalization and decorrelation of the feature vector. The success of the alignment procedure is sensitive to the parameters and coefficients in the feature vector, which in turn are dependent on the input data. By providing a rigorous statistical normalization, decorrelation and weighting of each individual coefficient the alignment procedure can be greatly improved. Various methods can be explored here similar to those incorporated in Hidden Markov Models (HMM), since the main strength in HMMs lies in their ability to integrate, through training, multiple example instances into a single probabilistic model. Since we are not dealing with a real-time realization a priori statistical information can be obtained for the given input signal.

Furthermore in this project a two stage model was devised for finding the correct target alignment. This simplification in the last stage of the algorithm is prone to errors (and especially noise). By using an appropriate DTW model incorporating a cost function penalizing certain movements in the warping function one can greatly improve the alignment estimation of two sequences.

6.3 Concluding Remarks

The investigation outlined in this report has illustrated that an automatic time-domain alignment of two audio sequences is possible. This type of algorithm (when implemented in a suitable environment) could drastically improve on the methodol-

ogy used in audio and TV/Radio-broadcasting studios throughout the world today.

Bibliography

[1] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies and M. B. Sandler, "A Tutorial on Onset Detection in Music Signals," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 13, no. 5, 2005.

[2] P. McLeod and G. Wyvill, "Visualization of Musical Pitch," *IEEE Proceedings of the Computer Graphics International,* 2003.

[3] G. Peeters, "Music Pitch Representation By Periodicity Measures Based On Combined Temporal And Spectral Representations," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP-06)*, 2006.

[4] T. Jehan, "Musical Signal Parameter Estimation," M.S. thesis, Univ. of California, Berkeley, CA, 1997.

[5] O. Deshmukh, C. Y. Espy-Wilson, A. Salomon, and J. Singh, "Use of Temporal Information: Detection of Periodicity, Aperiodicity, and Pitch in Speech," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 13, no. 5, 2005.

[6] Synchro Arts Limited VocALign, Unique Automatic Audio Alignment, *http://www.synchroarts.com/products/vocalign/vocalign.asp.*

[7] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Prentice-Hall International, Inc., 1996.

[8] A. G. Constantinides, *Digital Signal Processing and Digital Filters,* Course Notes, Imperial College London, UK, 2007.

[9] B. Gold and N. Morgan, *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*, John Wiley & Sons, Inc., 2000.

[10] P. A. Naylor, *Speech Processing*, Course Notes, Imperial College London, UK, 2007.

[11] W. Han, C.-F. Chan, C.-S. Choy and K.-P. Pun, "An Efficient MFCC Extraction Method in Speech Recognition," *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS 2006),* 2006.

[12] H. Kaprykowsky and X. Rodet, "Globally Optimal Short-Time Dynamic Time Warping Application to Score to Audio Alignment," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP-06)*, 2006.

[13] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", *IEEE, Trans. ASSP-26*, 1, pp.43-49, 1978.

[14] R. Yaniv and D. Burshtein, "An Enhanced Dynamic Time Warping Model for Improved Estimation of DTW Parameters", *IEEE Trans. Speech and Audio Process.,* vol.11, no. 3, 2003.

[15] R. Bellman and S. E. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, 1962.

[16] J. W. Picone, "Signal Modeling Techniques in Speech Recognition", *Proc. IEEE,* vol. 81, no. 9, 1993.

[17] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis,* New York: Academic Press, 1973.

[18] D. P. W. Ellis, Online web resource, *PLP and RASTA (and MFCC, and inversion) in MATLAB, http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/*, 2005.

[19] R. J. Turetski and D. P. W. Ellis, "Ground-Truth Transcriptions of Real Music from Force-Aligned MIDI Syntheses", LabROSA, Columbia University, USA, 2003.

Appendix A

Basic and Auxiliary Results

A.1 Matlab Code

A.1.1 Creating the feature vector (buildvec.m)

```
function [ output ] = buildvec( data, varargin )
%BUILDVEC builds a feature vector of an input audio file.
%F=BUILDVEC(DATA,VARARGIN)
%   data: Input data ('filename.wav')
%   varargin parameters (default):
%    'len' (20): Window length in ms
%    'premph'(0): Pre-emphasis filter (default off)
%    'logged'(0): Turn on logged computation (default off)
%    'wtype'('hamming'): Window Type
%    'nsize'(64): FFT size (note: half of the values are discarded)
%    'msize'(13): Number of cepstra to return
%
%   Jason FILOS, Imperial College London.

[len,wtype,premph,logged,nsize,msize] = ...
    process_options(varargin, 'len', 20, 'wtype', 'hamming', 'premph', 0, ...
    'logged', 0, 'nsize', 64);
```

```
%% Read in the file
[X,Fs]=wavread(data);
X_len = length(X);
disp(sprintf('### WAV File Read ###\n'));

%% Pre-emphasis (optional)
if premph ~= 0
    X = preemph(X,premph);
    disp(sprintf('### Pre-emphasis Completed ###\n'));
end

%% Split signal up into overlapping frames
frames = splitframe(X,Fs,len,wtype);
disp(sprintf('\n### Frame-Splitting Completed ###\n'));
disp(sprintf('Window Used: %s', wtype));
disp(sprintf('Block length: %d ms\n', len));

%% Calculate Short-term Energy and 1st derivative
[ste,ste_deriv]= ste7(frames,logged);
if (logged==1)
    disp(sprintf('### Logged Short-term Energy Computed ###\n'));
else
    disp(sprintf('### Short-term Energy Computed ###\n'));
end

%% Calculate FFT bins
[fftbins] = abs(spectralfft(frames,nsize));
disp(sprintf('### %d-point FFT bins Computed ###\n',nsize/2));

%% Calculate MFCC and 1st derivative
```

```
%% Merge results into one Feature Vector (FV)
ste_rows        = size(ste,1);
ste_deriv_rows  = size(ste_deriv,1);
fft_rows        = size(fftbins,1);

total_cols      = size(frames,2);
total_rows      = ste_rows+ste_deriv_rows+fft_rows;

fv = zeros(total_rows,total_cols);



% fv(1,:) = ste;
% fv(2,1:total_cols-1) = ste_deriv;
% fv(3:fft_rows+2,:) = fftbins;
% % fv(35:ceps_rows+34,:) = cepstra;
% % fv(48:60,1:total_cols-1) = cepstra_deriv;
% fv(67:ceps_rows+66,:) = cepstra;
% fv(80:92,1:total_cols-1) = cepstra_deriv;

icol = ste_rows;
fv(icol,:) = ste;
icol = icol + ste_deriv_rows;
fv(icol,1:total_cols-1) = ste_deriv;
icol = icol + 1;
fv(icol:fft_rows+(icol-1),:) = fftbins;
icol = icol + fft_rows;



disp(sprintf('### Feature Vector Initialized! ###'));
output = fv;
```

A.1.2 Alignment Model (alignment_model.m)

```
function [ output ] = alignment_model( fv, fv2, marker )
%ALIGNMENT_MODEL(fv,fv2,marker)
%   This routine estimates the optimal alignment point between to feature
%   vectors fv and fv2 representing a sequence of audio over time.
%   Additionally a marker (int) needs to be parsed as input to refine the
%   region of interest.
%
%   Jason FILOS, Imperial College London.

%% Compute Similarity Matrix (SM)
SM = dist(fv,fv2);      % euclidean distance
% figure; imagesc(SM);    % plot 2-D SM

%% Compute Shortest Path through Similarity Matrix
t=cputime;[p,q,D] = dp2(SM); cputime-t  % compute shortest path through dynamic
path = zeros(1, size(fv,2));
for i = 1:length(path); path(i)=p(min(find(q >= i))); end
% figure; plot(path);

%% Estimate Regions of Interest (Gradient detection)
for i = 1:length(path)-100; dx_dy(1,i)=std(path(1,i:i+100)); end
% figure; plot(dx_dy);

%% Compute Possible Alignment Points
[xmax, imax]=extreme(dx_dy);
regions = zeros(3,length(xmax));
for i=1:length(imax); pre_align(:,i)=(imax(i)-path(imax(i))); end
regions(1,:) = imax;        % location of maxima (sorted)
regions(2,:) = xmax;        % magnitude of maxima
regions(3,:) = pre_align;   % preliminary alignment points
```

```
%% Refine Region of Interest
temp = regions;
temp(3,:)=0.02*temp(3,:)/2;

j=1;
match=(marker*2)/0.02;
while j<length(temp)
    res = abs(temp(3,j)-marker);
    if (res<=2)                 % refine region if within +- 2 seconds
        match = regions(3,j)   % matching frame
        break;
    else
        j = j+1;
    end
end

%% Perform Exhaustive Search bounded by new Region
alignment = zeros(1,length(fv));

len1 = match-length(fv2);
len2 = match+length(fv2);

% Avoid getting out of range values
if (len1<=1);len1=1;end
if (len2>=length(fv));len2=length(fv);end

temp_alignment = align_exhaust(SM,len1,len2);   % exhaustive search
alignment(len1:len2)= temp_alignment(len1:len2);
% figure; plot(alignment);
```

```
%% Return Optimal Alignment Point
output = simple_compare2(alignment,length(fv),length(fv2));
```

A.1.3 Pitch Detection (spectral.m)

```
function [ output, xmax_array, imax_array,deriv_array] = spectral5( frames, Fs )
%SPECTRAL5 computes the fundamental pitch(es) for each block
%   frames: Input Array ;
%   Fs: Samplerate (Hz) ;
%
%   Jason FILOS, Imperial College London.

%% Data setup and initialization
[framesize, blocks] = size(frames)
deriv_array = zeros(framesize,blocks);

%% Compute FFT and first derivative
u = fft(frames);
deriv_array = diff(frames,1,2);

%% Find peaks in spectrum and output in descending order
for l = 1 : blocks

[xmax imax] = extreme(abs(audib_clean(u(:,l))));

xmax(end:-1:1);
fliplr(xmax);

imax(end:-1:1);
fliplr(imax);

imax = imax * (Fs/framesize)-(Fs/framesize); % Offset correction and translation
```

```
xmax_array(1:length(xmax),l) = xmax;
imax_array(1:length(imax),l) = imax;

end

%% Temporary graphical output (Debugging)
% col = 20;
% temp = abs(u(:,col));
%
% ax=0:(44100/framesize):44100-(44100/framesize);
% subplot(2,1,1), plot (ax,20*log10(abs(u(:,col))),'g')
% xlabel('frequency');
% ylabel('magnitude (dB)');
% AXIS ([0 3000 0 50]);

%% Reduce FFT output to look for dominant peaks only
for m = 1 : blocks
    [reduced] = reducer(imax_array(:,m),xmax_array(:,m),5);
    reduced_array(1:length(reduced),m) = reduced;
end

%% Estimate Pitch
for n = 1 : blocks
    [pitch] = pitch_estimator2(reduced_array(:,n),10);
    pitch_array(1:length(pitch),n) = pitch;
end;

%% Print output
output = pitch_array;
subplot(2,1,2), bar(output(1,:),'stacked')
```

```
end
```

A.1.4 Reducer (reducer.m)

```
function [ output_positions, output_mags ] = reducer( input_pos, input_mag, thre
%Reducer: Reduces the results of the peak-picking algorithm to values higher
%than the specified threshold.
%INPUT: input_pos = vector containing location of peaks (Hz); input_mag =
%vector of associated magnitudes for each peak (dB); threshold = threshold valu
%   Jason FILOS, Imperial College London.

len_mag = length(input_mag);
end_point = 0;



for i = 1 : len_mag
    if (input_mag(i) > threshold)
        end_point = end_point + 1;
    end
end

output_positions = input_pos(1:end_point);
output_mags = input_mag(1:end_point);

end
```

A.1.5 Pitch estimator (pitch_estinmator.m)

```
function [ output ] = pitch_estimator2( reduced_positions_array,threshold )
%UNTITLED1 Summary of this function goes here
%   Detailed explanation goes here
```

```
%
%   Jason FILOS, Imperial College London.

output = [];
pitch_array = [];

harmonics = 5; % Set number of harmonics to check, i.e. fc*1, fc*2 .. fc*harmoni

len_data = length(reduced_positions_array);

for l = 1:len_data

candidate_freq = reduced_positions_array(l);

for j = 2:harmonics
    candidate = j*candidate_freq;

    for i = 1:len_data
        if (abs((candidate)-reduced_positions_array(i))<=threshold) % Set pitch
            reduced_positions_array(i) = candidate_freq;
        end
    end
end

end

pitch_array = reduced_positions_array;

len_data = length(pitch_array);

for m = 1:len_data
```

```
candidate_freq = pitch_array(m);
count = 0;

for i = 1:len_data
    if (pitch_array(i)==candidate_freq)
        count = count + 1;
    end
end

if (count >= 2 && candidate_freq ~= 0)
    output = [output, candidate_freq];
end

output = unique(output);

end
```