Object coding of music using expressive MIDI

Welburn, Stephen J.

Object Coding of Music Using Expressive MIDI

Stephen J. Welburn

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of the
University of London.

Centre for Digital Music
School of Electronic Engineering and Computer Science
Queen Mary University of London

January 2011

Stephen J. Welburn

12th January 2011

Abstract

Structured audio uses a high level representation of a signal to produce audio output. When it was first introduced in 1998, creating a structured audio representation from an audio signal was beyond the state-of-the-art. Inspired by object coding and structured audio, we present a system to reproduce audio using *Expressive MIDI*, high-level parameters being used to represent pitch expression from an audio signal. This allows a low bit-rate *MIDI sketch* of the original audio to be produced.

We examine optimisation techniques which may be suitable for inferring Expressive MIDI parameters from estimated pitch trajectories, considering the effect of data codings on the difficulty of optimisation. We look at some less common Gray codes and examine their effect on algorithm performance on standard test problems.

We build an expressive MIDI system, estimating parameters from audio and synthesising output from those parameters. When the parameter estimation succeeds, we find that the system produces note pitch trajectories which match source audio to within 10 pitch cents. We consider the quality of the system in terms of both parameter estimation and the final output, finding that improvements to core components – audio segmentation and pitch estimation, both active research fields – would produce a better system.

We examine the current state-of-the-art in pitch estimation, and find that some estimators produce high precision estimates but are prone to harmonic errors, whilst other estimators produce fewer harmonic errors but are less precise. Inspired by this, we produce a novel pitch estimator combining the output of existing estimators.

Acknowledgements

This work would not have been possible without the support of my supervisor, Mark Plumbley. We got there. The inhabitants of the Centre for Digital Music at Queen Mary also deserve mention, having provided knowledge, companionship and entertainment. Particular thanks go to those involved in "C4DM Presents" for the opportunity to take part in projects for public consumption. One more QM thankyou must go to Melissa, for making sure that deadlines were met, courses were attended, and that the necessary forms were filed.

Of course, thanks go to my parents and my sister. Knowing they were rooting for me helped in the lows, and their interest in what I've been up to kept me on my toes. Hopefully I'll be less distracted during phone calls now.

Without Abigail, this could not have happened. Having taken all opportunities to vent, gripe and grump to her over the past few years her meagre reward is her name in print. For proof-reading beyond the call of duty, and for just being there at home I'll double the reward. Abigail.

And special thanks to two who are no longer with us: Martin Gardner for his Mathematical Puzzles and Diversions, without which I may never have enjoyed maths; and Alan Tayler, supervisor for my first degree, without whose inspiration this would not have happened.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past ten years, computer audio has come of age, the internet and MP3 files creating a new model for audio distribution. MP3 itself is almost 20 years old [ISO/IEC, 1993], raising the question of "what next for audio ?". MP3 is actually just one of a series of standards produced by the Motion Pictures Expert Group (MPEG) – officially, MP3 is "MPEG-1 Layer 3". Subsequent standards have been adopted for DVD and digital television but have yet to be adopted by consumers.

One of the more interesting MPEG audio standards is MPEG-4 Structured Audio (SA) [Scheirer, 1998], a format for creating synthesised sounds based on supplying both a synthesiser definition and a high-level "score" with which to control the synthesiser. Two approaches to structured audio were included in the standard: a script-based approach; and an approach combining sample-based synthesis and a control stream of MIDI data. Technologies related to the latter approach are commonly available, MIDI being ubiquitous in synthesised audio and sample-based synthesis existing "under the hood" on Windows PCs, Apple Macs and mobile devices. When MPEG-4 was introduced, however, it was noted that:

> "...it is very difficult to generate a MIDI representation automatically from a given waveform...It is unlikely that robust systems capable of translating expressive acoustic performances... into high-quality MIDI representations will be built within the next decade." [Vercoe et al., 1998]

It is now over a decade later and although some advances have been made, audio representations using MIDI are still simplistic, combining MIDI Note On and Note Off messages with low-level control of pitch and expression using the Pitch Bend and Expression controls. We believe that the time has come to re-examine MPEG-4

Structured Audio and that the technologies now exist to produce better quality MIDI representations from audio.

Extracting MIDI data from audio can be seen as an example of the general problem of analysis/synthesis audio coding, in which an audio signal is analysed to estimate the necessary parameters to allow synthesis of a similar signal – MIDI simply providing a convenient set of protocols to transfer parameters to a synthesiser. As the estimated parameters are preserved at the expense of losing other signal content, this is a "lossy" encoding.

Inspired by object coding and structured audio, we sought to produce a system to reproduce audio using an *Expressive MIDI* format, in which high-level parameters would represent features of the audio signal. This would allow a low bit-rate MIDI "sketch" of the original audio to be produced. Noting that this is a challenging problem involving the integration of components from distinct research areas, we focussed on estimating nuances of pitch, and on creating an Expressive MIDI system to both extract parameters from audio and to synthesise audio from those parameters to reproduce pitch features from the source material.

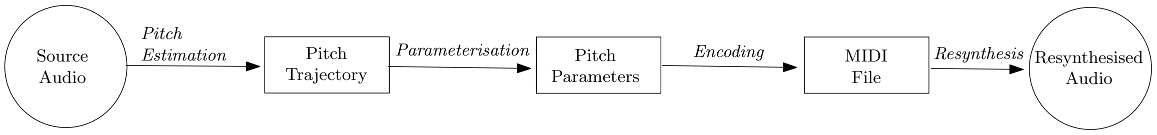


Figure 1.1: Model of an Analysis/Synthesis encoding system

In order to create such an Expressive MIDI system we can estimate pitch trajectories from audio, select synthesiser parameters to approximate the pitch trajectories, store these parameters in a suitable file and use a synthesiser to play back the file (Figure 1.1) – in Chapters 2 to 6, we build such a system. We then reconsider the state-of-the-art in pitch estimation looking for ways to improve Expressive MIDI (Chapters 7 and 8).

1.1 Thesis Structure

In Chapter 2, we introduce audio coding techniques including *object coding* (an analysis-synthesis approach to audio coding) and *structured audio* (in which high-level audio parameters are used). We identify MPEG-4 Structured Audio (SA) Wavetable Synthesis as a possible format for object coding audio and examine the relationship between this standard and standards from the MIDI Manufacturers Association.

In Chapter 3 we define the problem upon which we focus – an *Expressive MIDI* approach to object coding. Based on MPEG-4 SA Wavetable Synthesis, this encodes individual note pitch trajectories using *Envelope Generators* (EGs) and *Low Frequency Oscillators* (LFOs). We look to build an Expressive MIDI system for recordings of monophonic instruments by combining existing technologies and begin to create such a system, describing an approach to extracting note pitch trajectories from audio based on the YIN pitch estimator [de Cheveigné and Kawahara, 2002].

In Chapter 4 we continue to develop the Expressive MIDI system, examining algorithms which may be able to infer Expressive MIDI parameters from note pitch trajectories. We introduce several *bit-wise* optimisation algorithms and encodings. We consider properties of the encodings and introduce a new visualisation, giving insights into the effects of encodings on optimisation. We extend the work of Mathias and Whitley [1994] on evaluating combinations of algorithms and codings, finding that the choice of a suitable encoding can allow simple techniques to solve otherwise difficult problems. On the basis of this evaluation, we select the *CHC*[1] algorithm [Eshelman, 1991] (a non-traditional Genetic Algorithm) with a *Binary Reflected Gray Code* (BRGC) [Gray, 1953] encoding to infer Expressive MIDI parameters from the note pitch trajectories.

In Chapter 5 we infer Expressive MIDI parameters for audio files from the *RWC Musical Instrument Sounds* database [Goto et al., 2003], applying the CHC/BRGC optimisation to note pitch trajectories. We present a summary evaluation of the results of this parametrisation, and examine extracts from several RWC files in detail. We find that the Expressive MIDI parametrisation closely matches the pitch trajectories for many notes.

In Chapter 6 we complete our Expressive MIDI system, selecting an existing synthesiser with which to recreate audio from the Expressive MIDI parameters (to *resynthesise* the audio) and a MIDI encoding to pass those parameters to the synthesiser. We create MIDI encodings of Expressive MIDI parameters and resynthesise audio from these files. We examine examples of resynthesised pitch trajectories, identifying where differences between the resynthesis model and the original Expressive MIDI model produce differences in the output. Even so, we find that resynthesised trajectories are close to the Expressive MIDI model (within 10 pitch cents) and closely resemble the original pitch trajectories. We hypothesise that either creating a synthe-

[1]Eshelman simply presents CHC as the name of the algorithm, However, according to Whitley [1994], CHC stands for "*C*ross generational elitist selection, *H*eterogeneous recombination and *C*ataclysmic Mutation"

siser matching the Expressive MIDI model or modifying the Expressive MIDI model to match the available synthesiser would allow an even closer match.

In Chapter 7 we take a step back from the Expressive MIDI system. The completed system showed small pitch differences between the resynthesised pitch trajectories and the Expressive MIDI parametrisation. In order to identify the source of these differences, we examine the quality of YIN pitch estimates, introducing several pitch datasets and pitch metrics. We present a novel analysis of the performance of the YIN pitch estimator. We identify that YIN produces *harmonic errors* and present a theoretical analysis of the effect of removing such errors from the YIN output.

In Chapter 8 we present a comparison between the performance of YIN and two alternative pitch estimators: the PRAAT autocorrelation based pitch estimator [Boersma, 1993]; and the recent Sawtooth Wave Inspired Pitch Estimator (SWIPE) [Camacho, 2007]. Finding that SWIPE and YIN both produce good results, but that SWIPE achieves fewer harmonic errors, we look to produce a "best of both worlds" pitch estimator, automatically selecting either the YIN or SWIPE pitch estimate using a Bayesian classifier. We show that on datasets where YIN was particularly prone to harmonic errors this can, indeed, produce a better pitch estimator.

In Chapter 9 we represent our main conclusions from this work and consider future research directions.

1.2 Previously Published Work

This thesis is partly derived from the following previously published work by the author:

- Parts of Chapter 4 were originally published as a Technical Report by the Centre for Digital Music, Queen Mary University of London [Welburn and Plumbley, 2009b].
- Parts of Chapter 5 were originally presented at the 2009 conference on Digital Audio Effects (DAFx '09) [Welburn and Plumbley, 2009a].
- Parts of Chapter 6 were originally presented at the 127th Convention of the Audio Engineering Society [Welburn and Plumbley, 2009c].
- Parts of Chapters 7 and 8 were originally presented at the 128th Convention of the Audio Engineering Society [Welburn and Plumbley, 2010].

Chapter 2

Background

We firstly examine the purposes of audio coding and introduce approaches which have been used to date. We consider various audio related standards produced by the Motion Pictures Experts Group (MPEG) and the MIDI Manufacturers Association (MMA). In this manner, we lead up to the MPEG-4 Structured Audio (SA) standards [ISO/IEC, 2005] for audio synthesis and examine the lack of solutions for encoding audio as MPEG-4 SA. In subsequent chapters, we look to use features of MPEG-4 SA as a starting point for an audio coding system.

2.1 Audio Coding

Coding is the "accurate representation of one thing by another" [Pierce, 1980]. However, accuracy is not simply a matter of identically reproducing source material – in digital audio and images, judgements of accuracy are mediated by human perception and the use of the data.

All digital audio is encoded: recorded audio is a digital approximation of an analog signal; and digital audio files only become actual audio when played back – and the audio produced from a file will depend upon the chain of decoding equipment (e.g. the audio heard when a CD is played depends upon the CD player, amplifier and loudspeakers used) and the listening environment. Pulse Code Modulation (PCM) [Kientzle, 1998] is the most well known form of audio coding. In PCM, an analog audio signal is *sampled*, at some sample rate s_{R} Hz, and *quantised* to be stored using a number of binary digits (bits) i.e. PCM audio approximates the continuous signal at discrete times and discrete levels. In order to store high quality audio, the sample rate and the number of bits can be increased – increasing the overall *bit-rate* of the data. According to the Nyquist sampling theorem, the highest frequency which can be accurately represented in audio sampled at s_{R} Hz is $\frac{s_{\mathrm{R}}}{2}$ Hz [Loy, 2007].

The aim in audio coding is to take audio *data* and present it in a format which is appropriate for a given purpose. Considerations when selecting an appropriate encoding include:

- **Portability** - the need to represent the audio in a format which allows other platforms to reproduce the original audio;
- **Compression** - the ability to store or transmit the audio data using fewer bytes than the raw digital audio data;
- **Speed** - for an audio recording format, real-time encoding must be possible, for playback real-time decoding may be required;
- **Manipulation** - if the audio is to be manipulated during playback then it must be stored in a suitable format.

Portability can be achieved by including *metadata* that describes the audio data within the file. Electronic Arts Interchange Format Files (IFF) [Morrison, 1985] introduced a standard of using four byte ASCII codes to identify types of files and "chunks" of data within a file (so called "FourCC" formats). Microsoft WAVE files [IBM and Microsoft, 1991] and Apple's AIFF [Apple, 1989] are FourCC schemes containing chunks for metadata and sound data. The metadata for PCM data includes the number of audio channels in the file, the sample rate, and the number of bits per sample – without this data is not possible to play the audio back correctly. The MMA Standard MIDI File (SMF) [MMA, 1996] is also an example of a FourCC format.

Compression schemes can be classified as either *lossless*, where the original data can be exactly recovered (considered further in Section 2.2.2), or *lossy*, if the recovered data is not identical to the source material but provides a "good enough" match for the intended use of the data [MacKay, 2003, p. 74] (considered further in Section 2.2.1) e.g. for audio *playback* a 16-bit 44.1kHz signal ("CD quality") is usually regarded as being "good enough", components above 22.05kHz being outside the range of human hearing [Loy, 2006]. *Lossy* schemes discard data irrelevant to the intended use; whilst *lossless* compression schemes reduce *redundancy* in the data (e.g. using a Huffman coding) (see Section 2.2).

Encoding and decoding speed are affected both by the choice of *algorithms* and by the *target platforms* - a coding that works in real-time on a desktop computer may need too many resources to operate in real-time on a mobile device. In order to maximise playback performance (e.g. the number of audio files that can be played back concurrently in a sequencer), it is necessary to consider both the processing overhead of any decoding algorithms used, and the system overheads involved – e.g. audio formats with native hardware support will require least processing, but may require larger amounts of data than formats using more advanced non-native compression schemes. At the maximum compression level, the reference model MPEG-4 Audio Lossless Coder [Liebchen, 2004] was capable of encoding 16-bit 48kHz stereo audio signals at more than 5 times real-time rates on a 1.2GHz Pentium III-M which may be "good enough" if less than 5 stereo channels of audio are being recorded, but would be unsuitable for more channels. However, it could decode the compressed audio at more than 23 times real time which would be suitable for playing back a larger number of audio channels.

The ability to manipulate an audio signal during playback is a function of the encoding. Lempel-Ziv LZ78 encoding uses a dictionary of strings previously seen in a document to encode subsequent sequences [Nelson and Gailly, 1997, Ch. 9]. However, it is not possible to decode a symbol in the middle of the file without decoding all preceding data to rebuild the dictionary used to encode that symbol. With audio, it is a common requirement to be able to skip to a specific position in the file. Adaptive Differential PCM (ADPCM) audio coding [Kientzle, 1998, Ch. 13] encodes audio as a series of *packets*. For each packet, a starting value and quantisation level are specified and subsequent samples are stored as the quantised difference from the previous value. Hence, the playback position in ADPCM audio can be set to the start of any packet.

The Motion Pictures Expert Group (MPEG) has defined several standard formats for audio data, and *decoding* standards for these formats (e.g. MP3 and AAC). However, *encoding* of audio into MPEG standards is left as an implementation issue [Brandenburg, 1999] i.e. given a digital audio file, it is possible for multiple valid MP3 files to be produced based on the implementation of the standard. We next consider various approaches to audio coding, and the relevant MPEG standards.

2.2 Lossy and Lossless Codings

2.2.1 Lossy Coding

Noisy data includes information other than the required *signal.* This additional information does not need to be encoded, as it is irrelevant to the quality of the *signal.* Extending this principle beyond noise, it is not necessary to encode parts of the signal that are judged irrelevant to an application.

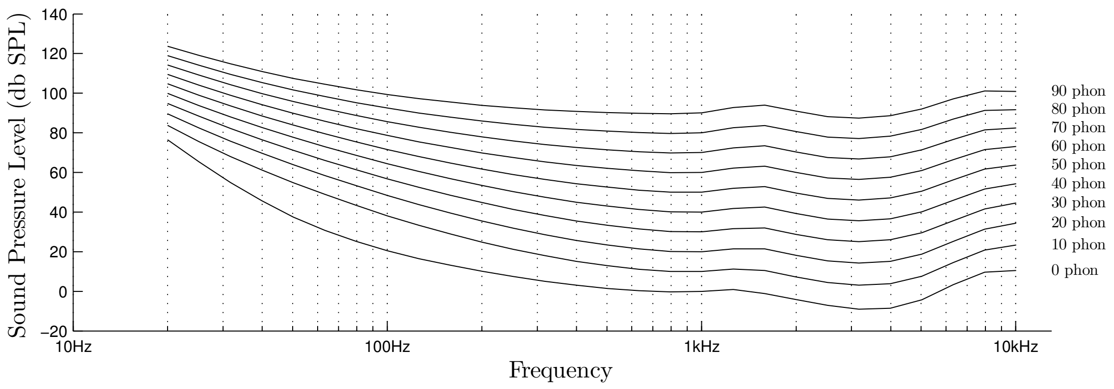


Figure 2.1: ISO 226:2003 Equal Loudness Contours showing the Sound Pressure Level required at each frequency to produce a loudness sensation of 0 to 90 phons. Loudness at the threshold of hearing is defined as 0 phon. The loudness of a 1kHz signal is then defined as matching the SPL. The loudness for other frequencies is then defined with reference to a 1kHz signal.

Human hearing does not respond equally to all parts of an audio signal:

- there is a frequency-dependent threshold of hearing beneath which a signal is imperceptible [Robinson and Dadson, 1957, van de Par et al., 2002] (the "0 Phon" level in Figure 2.1);
- frequency response varies, being most responsive to frequencies in the 1–5kHz range [Fletcher and Munson, 1933, ISO/IEC, 2003] (Figure 2.1);
- frequencies below 16Hz are not perceived as tones, but as pulses [Beament, 2001, p. 94];
- the upper limit of hearing can be above 16kHz, but lowers with age and can be under 9kHz [Beament, 2001, p. 94];
- noise in the signal can mask speech or tones [Hawkins, Jr. and Stevens, 1950].

Hence, for audio playback, it is not necessary to represent all components of the audio at the same level of detail. A scheme which allows a detailed representation of the more perceptible parts of the signal, whilst reducing the detail in less perceptible parts can be indistinguishable from the source material for most listeners. This *irrelevancy reduction* allows *lossy* compression of the audio data – the data may sound the same to a human listener (either exactly the same with some compression or "good enough" at higher compression levels e.g. for listening on mobile devices), but the audio produced will not match the source material. As these schemes compress less of the original *information*, it is possible to achieve high levels of compression.

MPEG-1 layer 3 audio ("MP3"), was originally defined in 1991, subsequently being ratified as an ISO/IEC standard [ISO/IEC, 1993]. When compressing the signal, the number of bits allocated to psychoacoustically uninteresting parts of the signal is reduced, more bits being allocated to the most perceptually important parts. The resulting data is then compressed using a lossless Huffman coding. In early MP3 encoders, various qualities of coding schemes were observed based on implementation choices - all of which produced "valid" MP3s, but meant that there were subjectively "good" and "bad" encoders [Brandenburg, 1999].

The subsequent MPEG-2 standard [ISO/IEC, 1998] was developed for audiovisual applications, and is used for DVD and digital television. The audio component was intended simply to extend the MPEG-1 standard to include support for multichannel audio, and to allow additional lower sampling frequencies (16kHz. 22.05 kHz and 24kHz). During testing of the standard, it became apparent that significant improvements in coding efficiency could be achieved by introducing new coding algorithms, and a new standard for audio coding was introduced, MPEG-2 "Advanced Audio Coding" (AAC). AAC can reach similar quality to MP3 at about 70% of the bit-rate [Brandenburg, 1999].

MPEG-4 standardisation was originally targeted at videophone applications. By 1994, however, predicted improvements in video compression had not arisen, and new application areas were emerging. Based on trends towards wireless communications, interactive computer applications and the integration of audio-visual data in applications, MPEG-4 was realigned to the intersection of *telecoms*, *computers* and *TV and film*. Two separate MPEG audio groups were created, one to evolve existing standards for natural audio [Brandenburg et al., 2000], and one to develop new standards for Synthetic-Natural Hybrid Coding (SNHC) [Puri and Eleftheriadis, 1998, Scheirer et al., 2000] allowing natural audio to be combined with synthesised sounds.

In contrast to the "one size fits all" audio codings of MPEG-1 and MPEG-2 audio standards, MPEG-4 audio [ISO/IEC, 2005] incorporated several lossy coding schemes each for a specific type of natural audio:

- MPEG-4 AAC (Advanced Audio Coding) – a follow up to MPEG-2 AAC for general audio coding (adopted as the standard audio format on Apple iTunes);
- MPEG-4 CELP (Code excited linear prediction) and HVXC (Harmonic Vector eXcitation Coding) – linear-prediction based methods for natural speech coding;
- MPEG-4 HILN (Harmonic and Individual Lines, plus Noise) – a parametric audio format in which harmonics components, individual sinusoidal "lines" and a noise component are modelled for use at very low bit-rates [Purnhagen and Meine, 2000].

In order to make best use of the MPEG-4 natural audio standards, the most appropriate codec should be used for the signal content (e.g. CELP, HXVC, AAC or HILN for natural audio). Beritelli et al. [1998] used fuzzy logic, genetic algorithms and neural nets to classify audio under four classes: talk, music, noise and tone, with various subclasses beneath. Eighteen audio features were used in the classification, and successfully identified the classes under a limited set of tests (e.g. the "music" tested used trumpet, harpsichord and castanet recording to represent the wind, string and percussion subclasses). Theoretically, audio could be analysed, classified and then encoded using the most appropriate codec.

2.2.2 Lossless Codings

A lossless encoding scheme uses complementary encoding and decoding algorithms to allow the original "raw" data to be recovered perfectly from the encoded version i.e. a lossless encoding is an invertible transformation from the (digital) audio space to a subspace of the space of possible files. Lossless compression is possible as the structure within audio creates *redundancy* in the data. Without structure to the data, samples would be uniformly distributed random numbers, i.e. noise. *Redundancy reduction* takes advantage of this structure within the signal to represent it in a more compact manner. Information-theoretic compression algorithms allow data to be compressed efficiently, using small symbols (e.g. bit strings) to represent common values and longer symbols for less frequently seen values. To reduce the file size further, *predictive modelling* can be used – using a model of the signal to predict samples based on preceding data – only encoding the *residual* difference between the predicted value and the actual sample. Reducing redundancy in data relies upon a suitable model of the data - a model that works for one type of data (e.g. audio) is unlikely to be appropriate for another type of data (e.g. images).

General purpose "universal" coders such as Lempel-Ziv, as used in `gzip` can be applied to a wide range of sources [MacKay, 2003, pp. 119–122]. However, they are less effective on data of a known type (e.g. speech) than schemes targeted at that specific type of data – appropriate *modelling* of the data allows compression to work more effectively [Nelson and Gailly, 1997]. Any lossless coding can increase the size of the data if inappropriate data is used - e.g. Lempel-Ziv is effective in the limit as the quantity of data increases, but may increase file sizes for small amounts of data; Rice coding (a specific Huffman code [MacKay, 2003, Ch.5 pp. 98–101] for Laplacian data) will be inappropriate if the data is not distributed in an approximately Laplacian manner.

An example lossless audio coding from the literature is that used by the SHORTEN audio encoder [Robinson, 1994]. SHORTEN models each frame in the signal using either: one of four standard polynomial models; or linear predictive coding (LPC). The residual differences are then encoded using a Rice code (Robinson observing residuals to be "approximately" Laplacian). The non-proprietary, fully open Free Lossless Audio Codec (FLAC) [Coalson, 2000] is based on SHORTEN, adding an additional polynomial model and allowing model parameters to be adapted across subframes in the audio. Noting that there was little variation in the performance of predictive lossless encoders, Hans and Schafer [2001] proposed AudioPaK, in which only the polynomial predictors were used – compression ratios were achieved which were comparable to other codecs at the time.

A lossy coding can be made lossless by additionally including the *residual* difference between the behaviour expected according to the model and that observed in the original system [Hans and Schafer, 2001]. If the lossy encoded signal, $\hat{\mathbf{x}}$ is

$$\hat{\mathbf{x}} = \Theta(\mathbf{x}) \ , \tag{2.1}$$

where Θ is the lossy encoding function and $\mathbf{x}$ is the original signal, then the *residual error*, $\mathbf{e}$, between the lossy decoding and the source data is given by

$$\mathbf{e} = \mathbf{x} - \Phi(\hat{\mathbf{x}}); \tag{2.2}$$

where Φ the lossy decoding function. If we store both the lossy coded version of the signal, $\hat{\mathbf{x}}$, and the residual error, $\mathbf{e}$, then we can recover the original signal

$$\mathbf{x} = \Phi(\hat{\mathbf{x}}) + \mathbf{e} \tag{2.3}$$

allowing files to be recoded as improved encoders and new voice models become available. If a low bit-rate version of the data is required, the residual can be discarded or transcoded to a lossy format with a subsequent reduction in file size and audio quality. MPEG-4 Scalable Lossless Coding (SLS) adds an enhancement layer over lossy MPEG-4 AAC (pp.26) encoded audio to include the residual difference between the lossy coding and the original signal. This allows a lossless coding with a 50% reduction in file size over raw PCM audio data [Geiger et al., 2007].

2.3 Parametric Coding

Expressing audio as parameters for a model of the audio signal can achieve high compression ratios for audio matching that model. Unless the audio exactly matches the model and the precise parameters are inferred, this is a lossy encoding of the audio. It is an "analysis-synthesis" coding technique – the source audio is *analysed* to estimate parameters and audio can then be *synthesised* using those parameters.

Parametric coding based on additive synthesis has a long history. Additive synthesis representations appeared in the late 1980s for both speech [McAulay and Quartieri, 1986] and musical sound [Smith and Serra, 1987]. Subsequent work extended this – representing a signal as an additive model of sinusoidal components, and a subtractive model for noise components [Serra, 1989, 1997]. The low bit-rate *Analysis/Synthesis Audio Codec* (ASAC) encoded audio as a set of single spectral lines, successively reducing the residual signal as components were added to the model [Edler et al., 1996]. This parametric model was expanded, as an "object-based" analysis/synthesis audio coder, to include harmonic tones and noise content [Purnhagen et al., 1998]. Bit-rates up to 16 kbit/s were targeted, and the codec was standardised in MPEG-4 version 2 as Harmonic and Individual Lines plus Noise (HILN) [Purnhagen and Meine, 2000].

Parameter estimation for other synthesis techniques has also been examined:

- Horner et al. [1993] and Wun and Horner [2005] considered wavetable synthesis – playing back an oscillator table, or combining the output of multiple oscillator tables, whilst varying the frequency (sample rate) and amplitude;
- Beauchamp [1981] and Mitchell and Creasey [2007] looked at FM synthesis [Chowning, 1973].

However, these have not been adopted in standard audio codecs.

2.3.1 Object Coding

Object coding of audio analyses a piece of audio to estimate parameters for synthesis *objects* – each object encapsulating a basic behaviour pattern (although this behaviour may be parameter dependent). Driving the objects with the parameters allows an approximation of the original audio to be created. It is a form of parametric encoding. Traditionally, it has been regarded as a technique for low bit-rate encoding and has been examined in relation to sum-of-sinusoids models [Vincent and Plumbley, 2005] and instrument models [Tolonen, 2000]. We seek to produce a high quality representation of audio using similar techniques.

Given a set of predictable objects, object coding finds suitable parameters to drive those models, and to combine their outputs to produce a more complex behaviour. For an audio signal, we wish to represent the evolution of the signal over time as a combination of some underlying time-evolving objects and a series of time-dependent parameters. The modelled objects (alone or in combination) may or may not represent physical objects present in the system which produced the source audio.

Using machine-learning techniques such as evolutionary algorithms, neural networks or Bayesian models, parameters can be estimated for the synthesis objects. For each object, these parameters can then be used to resynthesise part of the original scene.

An “ultimate” form of object coding would involve capturing all audio related details of a performance and the ability to synthesise audio from those parameters – the synthesised instruments being driven by the performance details to recreate the audio. However, for coding, it is not necessary to represent the original source material, only the content of the file e.g. if events always occur simultaneously we have no need to regard these as individual events but can use a composite event to produce the combined output.

Various individual voice models have been modelled as “object based”: Glass and Fukudome [2004] estimated parameters for a physically modelled Karplus-Strong plucked string model; Peltola et al. [2007] use linear prediction to find parameters for physics-based models of hand-clapping; Chang and Su [2002] used neural networks to extract waveguide synthesis parameters from struck-string sounds e.g. piano; and Melih and Gonzalez [2002] used sinusoidal coding to find speech and piano model parameters. However, these are very specific in their application, and not suited to general music coding.

More general object coding schemes have been developed combining sinusoidal tracks, harmonics and noise models for MPEG-4 HILN ("Harmonic and Individual Lines plus Noise") [Purnhagen and Meine, 2000] and using Bayesian object parameter extraction [Vincent and Plumbley, 2007]. However, these schemes are (i) intended for low bit-rate applications of low-to-medium quality; and (ii) provide parameters closely related to the synthesis model rather than higher level features which may be musically meaningful. We now consider existing audio synthesis standards that allow description of both synthesisers and their parameters – making them possible candidates for use with object coding.

2.4 MPEG-4 Synthetic Audio Coding

As indicated above (Section 2.2.1) the MPEG-4 standard considered both Natural audio and "Synthetic Natural Hybrid Coding" (SNHC). Part of the SNHC standard considers production of synthetic audio. The MPEG-4 audio standard [ISO/IEC, 2005] offered two synthetic audio standards:

- MPEG-4 Text To Speech (TTS) provides a format allowing speech to be synthesised according to provided text and prosody;
- MPEG-4 Structured Audio (SA) to allow "musicians and sound designers" to produce synthetic audio.

Of these two, MPEG-4 Structured Audio is aimed at music and may be a candidate solution for use in object coding of musical audio.

Conceptually, structured audio represents sound as a series of parameters for models of audio, emphasising the possibility of multiple models being appropriate for a single piece of audio [Vercoe et al., 1998]. Unlike general object codings, structured audio parameters have a *semantic meaning*, representing high-level features of the sound and allowing sound designers to modify the output audio by varying these parameters (e.g. allowing direct control of vibrato depth and frequency). Synthesising audio from high-level parameters allows pitch and amplitude modulators to be applied directly by the synthesis engine – thus allowing structured audio to modulate the signal at the full audio sample rate. Encoding sound using structured audio is expected to produce a smaller representation than the raw audio as a result of the high-level "structural redundancy" in the data [Scheirer, 2001].

MPEG-4 Structured Audio (MPEG-4 SA) was developed from the work of Vercoe and Scheirer [Vercoe et al., 1998, Scheirer, 1999] as a synthetic audio component for "musicians and sound designers" to complement the MPEG-4 natural audio standards. Target applications would then use "Synthetic-Natural Hybrid Coding", combining synthesised sounds (using MPEG-4 SA) with natural audio (e.g. recorded speech, using MPEG-4 CELP) [Scheirer et al., 2000]. The MPEG-4 standard includes a set of profiles, specifying subsets of the full standard which can still achieve MPEG certification. MPEG-4 has four high-level profiles – the "Main" and "Synthetic" profiles include MPEG-4 Structured Audio, "Speech" and "Scalable" do not [Koenen, 2000].

	Supported Components			
Object Type Profile	MIDI	SASBF	SAOL	SASL
General MIDI	✓	✗	✗	✗
Wavetable Synthesis	✓	✓	✗	✗
Algorithmic Synthesis	✓	✗	✓	✓
Full	✓	✓	✓	✓

Table 2.1: MPEG-4 Structured Audio object type profiles

MPEG-4 SA encapsulates: MIDI data (see Section 2.5.1, below); the Structured Audio Sample Bank Format (SASBF); and the Structured Audio Orchestra and Score Languages (SAOL and SASL). [1] Within MPEG-4 SA there are four sub-profiles, referred to as "object types" by Koenen [2000], which support subsets of these MPEG-4 SA functions (Table 2.1) [Scheirer, 1998].

Although it has been predicted that structured audio will only be widely used when sources can be separated [Viste and Evangelista, 2001], there has been little evidence of work on structured audio even in the monophonic case. Examining the state of MPEG-4 Structured Audio reveals several reasons for this gap between the vision and the reality.

As with the other MPEG standards, MPEG-4 Structured Audio defines how sound should be decoded from the given data – it does not consider how to encode the data –

[1] An orchestra of instruments can be defined using SAOL scripts, operating at the audio sample rate (`a`-rate). These instruments are then driven via SASL score commands, at a separate control rate (`k`-rate), – the `k`-rate usually being lower than the `a`-rate.

and, when MPEG-4 SA was agreed, decoding it was "slightly beyond state-of-the-art" [Scheirer and Ray, 1998].

Real-time use of SAOL has proven to be elusive. Lazzaro and Wawrzynek [2001] produced a standard implementation of the MPEG-4 SA decoder, cross-coding SASL and SAOL into a C program. Profiling the performance of the SAOL language, Zoia and Alberti [2003] noted that combining an interpreted language with sample-by-sample a-rate processing was a poor combination for real-time use. Using a stackless architecture and parallel processing they developed the Structured Audio Interpreter (SAINT) as a higher performance SAOL solution. Siao et al. [2005] developed JavaOL (Java Orchestra Language) as an efficient model for real-time streaming of MPEG-4 Structured Audio over the internet (or other TCP-based networks). Java based, its performance was increased by relying on a C++ engine for wavetable synthesis. Created as a SAOL interpreter, it was found that interpreting SAOL produced a performance bottleneck [Su et al., 2004]. JavaOL subsequently evolved into a Java library implementing core SAOL opcodes within Java classes producing a faster system than the SAOL-based version [Wang et al., 2006]. In other words, although inspired by SAOL, JavaOL deprecated SAOL itself from the implementation.

SAOL and SASL were based on CSound, initially developed by Vercoe at MIT in 1985 [Boulanger, 2000]. According to Vercoe, a proponent of the structured audio concept, "the SAOL structure is much less efficient than CSound" [Lyon, 2002] – raising the question of why CSound was not used for MPEG-4. At the time MPEG-4 SA was being standardised, CSound, although freely available for non-profit and research use, was published under MIT copyright with "All rights reserved" [Vercoe et al., 2010, pp. xxix]. SAOL and SASL were therefore designed to allow CSound-like functionality whilst avoiding issues related to the MIT licensing agreement. Additionally, CSound lacked support for real-time processing [2].

Scheirer [2001] showed that SAOL is a Turing-complete computation system. SAOL can therefore can be used to produce any synthesis algorithm, and offers a generalised model for audio coding – it should be possible to implement any other coder within MPEG-4 SA. Indeed, Vasiloglou et al. [2002] implemented the AudioPak

[2]The development of CSound 5, a thorough overhaul of the legacy CSound code-base [ffitch, 2006] and the 2003 licensing of CSound under the Lesser Gnu Public License (LGPL version 2.1 and later), overcame most of the issues which prevented CSound being a possible format for a structured audio standard at the time MPEG-4 SA was being agreed. For "Orchestra + Score" computer music production, CSound therefore currently appears to be a better option than the MPEG-4 "SAOL + SASL" standard.

lossless codec [Hans and Schafer, 2001](see Section 2.2.2) in MPEG-4 SA as a demonstration of the capabilities of the SAOL language. However, it was noted that the MPEG-4 SA model was inconvenient for implementing the codec. Gaps were also found in the MPEG-4 SA specification (e.g. SA defines computation as 32-bit float, but fails to specify the mantissa / exponent split; the range of output from SA is given as [-1, 1], but the precision is not given). Additionally, a SAOL+SASL program can only terminate at a `k`-rate cycle, making it difficult to implement precise lossless coding - one work-around being increasing the `k`-rate to the `a`-rate with attendant increase in processing and possible issues for performance.

Although the MPEG-4 SAOL+SASL model for structured audio looked to be a suitable standard for use in object coding of audio, the above implementation issues reveal it to be inappropriate. However, the remaining Wavetable Synthesis elements of MPEG-4 Structured Audio may also be a suitable for use with object coding. Having identified MPEG-4 Wavetable Synthesis as a possible format for object coding of audio, we take a step back to consider the various MIDI Manufacturers Association (MMA) standards, and their relationships to MPEG-4 SA Wavetable Synthesis.

2.5 MIDI Manufacturers Association Standards

2.5.1 Musical Instrument Digital Interface (MIDI)

MIDI is a hardware and software specification, introduced by the MIDI Manufacturers Association (MMA), which allows musical instruments and related devices (e.g. mixers, sequencers and computers) to exchange information over 16 "channels"[MMA, 2000b]. The standard defines both a hardware interface and a data format for messages and was originally intended for live performance.

The MIDI Hardware Specification

MIDI was designed to operate over a unidirectional serial interface at 31.25 kb/s, requiring separate MIDI "in" and "out" connections, often combined with a MIDI "thru" connector to pass the unmodified input to another device. Subsequent concerns over the available bandwidth for MIDI data have been overcome in recent years by encapsulating MIDI in higher speed transports: a high speed serial (IEEE-1394 "Firewire" [IEEE, 1996]) adaptation layer has been produced, allowing 8 MIDI data

Name	**Status Byte**	**Data Byte(s)**	**Data (7-bit values, 00 to 7F)**
Note Off	8X	NN VV	NN : Note Number, VV : Velocity
Note On	9X	NN VV	NN : Note Number, VV : Velocity
Key Pressure	AX	NN PP	NN : Note Number, PP : Pressure
Control Change	BX	NN VV	NN : Controller Number, VV : Controller Value
Program Change	CX	NN	NN : Program Number
Channel Pressure	DX	PP	PP : Pressure
Pitch Bend	EX	HH LL	HH : 7 most significant bits, LL : 7 least significant bits

Table 2.2: MIDI Channel Messages ("X" indicates the MIDI channel number)

streams (i.e. 128 channels of MIDI data) to be multiplexed in a single "MIDI conformant data channel" [MMA, 2000a]; and a standard for embedding MIDI in the internet real-time protocol (RTP) [Schulzrinne et al., 1996] has been set [Lazzaro and Wawrzynek, 2004].

The MIDI Data Format

The MIDI data format consists of *status bytes* which set the state of the MIDI system and indicate the type of *data bytes* to expect. Status bytes can be identified as the most significant bit (msb) is set to 1, the msb for data bytes being 0. Status bytes can indicate either system messages or channel messages.

System messages consist of System Exclusive (sysex), System Common and System RealTime messages. A system exclusive message starts with a status byte of "F0", identifies a target system based on a Device ID, includes an arbitrary number of data bytes and terminates with a "EOX" status byte ("F7"). System Common messages (status byte F1–F7) and System RealTime messages (status byte F8–FF) are used to synchronise all MIDI devices within a system.

A single MIDI connection supports 16 channels of MIDI data – the lower four bits of *channel message* status bytes identifying the channel the status applies to. If multiple sets of data use the same status byte then the status byte only needs to be sent once, followed by each set of data bytes (this is termed *running status*).

The supported channel messages are:

Voice ID	Voice Name	Pitch Bend Range
1–1	Piano L	1
2–3	Strings Heavy 1 FC	3
2–5	Strings Mellow 1 FC	5
2–8	Solo Violin MW	8
3–1	Closed Pipe MW	7
4–5	Female Vocal 1 MW	4
4–7	Male Bass BC	2
6–1	Elec. Piano Tremolo L MW	2
6–5	Elec. Piano 1	4
11-1	FM Piano 1	7

Table 2.3: Example Pitch Bend Ranges for Standard Yamaha TX816 Voices

- "Note On" and "Note Off" messages including the "note number" (e.g. based on the key pressed) and "velocity" data';
- "Program Change" messages to select a program (a "preset", "patch" or "voice");
- "Channel Pressure", "Key Pressure" and "Pitch Bend" control messages;
- "Control Change" messages (CCs) for *modifying* tones using controllers other than keys. CCs are *not* used for parameters of tones [MMA, 2000b, pp. 9,11] except via RPNs and NRPNS:
  - Registered Parameter Numbers (RPNs) for standard MMA defined synthesiser settings;
  - Non-Registered Parameter Numbers (NRPNs) for manufacturer provided settings.

Each channel message uses either 1 or 2 data bytes (the status and data bytes for these messages are shown in Table 2.2).

The MIDI standard supports 128 notes for each voice. For pitched voices, the usual convention for MIDI notes is that they occur at semitone intervals with note number 69 defined as 440Hz based on the standard Western well-tempered diatonic scale. The fundamental frequency, f, for a given MIDI note number, k, is then:

$$f = 440 \times 2^{\frac{k-69}{12}} \text{ Hz} . \tag{2.4}$$

When discussing small variations in pitch, we will consider *pitch cents*, 1 pitch cent being $\frac{1}{100}$th of a semitone.

Subtler adjustments of pitch can be achieved by using the Pitch Bend controller. However, as the MIDI standard was designed for live performance, it does not define the relationship between Pitch Bend controller values and pitch adjustment (e.g. whether is it a linear relationship and what range Pitch Bend should cover). The actual effect of Pitch Bend is regarded as a feature of the synthesiser being used, or even of the specific voice currently in use e.g. the Yamaha TX816 includes the Pitch Bend range as a voice parameter, and this parameter varies across the standard voices (Table 2.3) [Yamaha, 1985]. Independent Pitch Bend control can be applied to each of the 16 MIDI channels.

Standard MIDI Files (SMF)

In order to allow MIDI song data to be transferred between systems, the Standard MIDI File format (SMF, ".mid") was published by the MMA [MMA, 1996]. An SMF contains a block of header data describing the file format and one or more "tracks". Each track is a series of MIDI event messages (e.g. CC, NRPN, sysex, Note On, Note Off) each with a "delta-time" indicating the timing relative to the previous event message. Delta times are stored in "ticks", the resolution of which can be specified based on the number of ticks per quarter note (TPQN) and the track tempo, both specified in the file header.

SMF supports three main formats: "0" with a single multi-channel track; "1" with one or more simultaneous tracks; and "2" for storing triggerable MIDI "patterns" in tracks.

In addition to standard MIDI data, *meta events* may be included in the track to provide additional information (e.g. copyright details, track name, lyrics, tempo).

2.5.2 MMA Synthesiser Specifications

2.5.3 General MIDI (GM)

The MIDI 1.0 standard and SMF define ways to pass messages between controlling devices and synthesisers. However, the actual sounds produced by the synthesiser in response to those messages are undefined – e.g. there is no concept of requesting the synthesiser to play a "trumpet" sound. In order to allow song data written using the MIDI protocol to be played back on different systems using appropriate sounds, the MMA created General MIDI (GM) [MMA, 1991a]. GM associates 16

Prog.	**Family**	**Prog.**	**Family**
1-8	Piano	9-16	Chromatic Percussion
17-24	Organ	25-32	Guitar
33-40	Bass	41-48	Strings
49-56	Ensemble	57-64	Brass
65-72	Reed	73-80	Pipe
81-88	Synth Lead	89-96	Synth Pad
97-104	Synth Effects	105-112	Ethnic
113-120	Percussive	121-128	Sound Effects

Table 2.4: General MIDI Instrument Families

instrument families (Table 2.4), each containing 8 specific instruments, with MIDI program numbers 1 to 128 – the appropriate *voice* (instrument sound) is selected by specifying a program number in the MIDI data. Similarly, specific percussion sounds are also available, mapped to individual notes on MIDI channel 10 – each note playing a specific type of percussive "hit" e.g. a bass drum or snare drum.

GM aims to provide a level of consistency in playback by indicating that devices support: MIDI note 60 as "Middle C" (the conventional MIDI tuning given in Section 2.5.1); a minimum required level of polyphony (24 voices); the standard voices on all channels; different voices being active on each of the 16 MIDI channels; velocity control of all voices; and a default pitch bend range of ± 2 semitones.

As a result, MIDI files designed for GM can be played back on compliant hardware with some consistency. However, implementation details vary, and the precise sound of a synthesiser produced by one manufacturer is unlikely to match that of another – no specific synthesis technique is given, and an FM synthesis "Trumpet" voice will not sound the same as a "Trumpet" sound synthesised from audio sampled from a real trumpet.

For further control of the synthesiser, the GM specification also includes support for specific Control Change messages (CCs) (Table 2.5a) and Registered Parameters (RPNs) (Table 2.5b). General MIDI 2 (GM2) [MMA, 1991b] supports additional Control Change messages. However, implementation details are not specified – the GM2 standard simply stating "Exact behaviour is left to the manufacturer's discretion". Implementations of the specification may therefore vary, and only those controllers for which the precise behaviour is specified can be relied upon to give consistent results across GM2 synthesisers. For audio coding purposes, the output audio needs to be

Controller #	Summary	Description
1	Modulation	Changes “the nature of the sound in the most natural (expected) way” [MMA, 1991a, p. 7]
7	Volume	Main channel volume to set overall dynamics and balance of channels, 0 = minimal (silence), 127 = maximum volume, default 100 [MMA, 1991a, p. 3]
10	Pan	Determines where a single source is in the stereo field, 0 = “hard left”, 64 = “centre”, 127 = “hard right” [MMA, 2000b, pp.13]
11	Expression	Volume accent above the programmed or main volume [MMA, 2000b, p. 13]
64	Sustain	Ignores subsequent Note Off events, held notes ending when the sustain controller is released
121	Reset All Controllers	Resets all controller values to their initial state [MMA, 2000b, p. 25]
123	All notes off	Provides a Note Off for all notes that are currently “on” [MMA, 2000b, p. 24]

(a) Control Change messages (CCs) supported in General MIDI

Registered Parameter #	Description	
0	Pitch Bend Sensitivity	The maximum absolute change in pitch using the pitch bend controller (MSB is in semitones, LSB in cents – MSB `06` and LSB `00` therefore defines a pitch bend range of ± 6 semitones i.e. 1 octave total range)
1	Fine Tuning	An offset from A440 tuning: $V = \text{MSB} \times 128 + \text{LSB}$ offset $= \frac{100V}{8192}$ cents
2	Coarse Tuning	An offset from A440 tuning: offset = MSB $-$ 64 semitones

(b) Registered Parameter Numbers (RPNs) supported in General MIDI

Table 2.5: CCs and RPNs supported in General MIDI

predictable – there is a specific sound which we wish to create – and the inability of GM to support this consistency renders it inappropriate for audio coding.

Roland and Yamaha both extended the General MIDI standard, creating their own proprietary "standards", supported by their subsequent GM compatible synthesisers - Roland GS (1991) and Yamaha "Extended General MIDI" (XG) [Yamaha, 1994, pp. 112–115] e.g. providing additional sets of sounds and audio effects such as chorus.

2.5.4 Downloadable Sounds (DLS)

The introduction of the MMA Downloadable Sounds format (DLS1) moved MIDI further towards replicable audio [MMA, 1997]. DLS1 defines a standard for wavetable synthesis (in this case defined as sample playback plus modulators), encapsulating both: basic waveform content; and how the synthesiser engine should respond to controller messages. The file format is FourCC based, using four character codes to identify "chunks" of data within the file (pp.22). This was further standardised in the DLS2 format [MMA, 2006]. DLS is one of the most prevalent synthesis standards, as it is integrated in Microsoft Windows, Apple Mac OS X and many mobile devices. MPEG-4 Structured Audio Sample-Bank Format (SASBF) is an implementation of DLS2[3].

DLS2 is a sample-based synthesis engine in which base samples are manipulated by modifying pitch, amplitude and timbre using envelope generators (EGs), low frequency oscillators (LFOs) and filters. The basic unit of synthesis within a DLS2 file is the *instrument*, corresponding to a specific MIDI Program Number. In order to allow the timbre of an instrument to vary with pitch and loudness, multiple *regions* can be defined for the instrument, each responding to a range of MIDI Note On messages – a specific region being selected based upon the MIDI Note Number and Velocity received in the Note On message. In turn, each region is associated with a sample of audio data. In order to play back a sample for various note lengths, it can be labelled with a loop region, consisting of one or more cycles of the waveform. During synthesis, the loop region will then be repeated to produce the required length (Figure 2.2). The combination of a set of samples for an instrument and the associated metadata

[3]The Creative Labs SoundFont$^{\text{TM}}$format is closely related to DLS [Rossum, 1997]. In 1993 Creative Labs created the SoundFont as a standard format for wavetable synthesis data for use with their Sound Blaster AWE32 PC sound card. In 1996, SoundFont 2.0 [Labs, 1998] was introduced and the details of the standard were publicly disclosed. MPEG, the MMA and Creative Labs jointly created DLS2 to merge features of their independent standards [MMA, 1998].

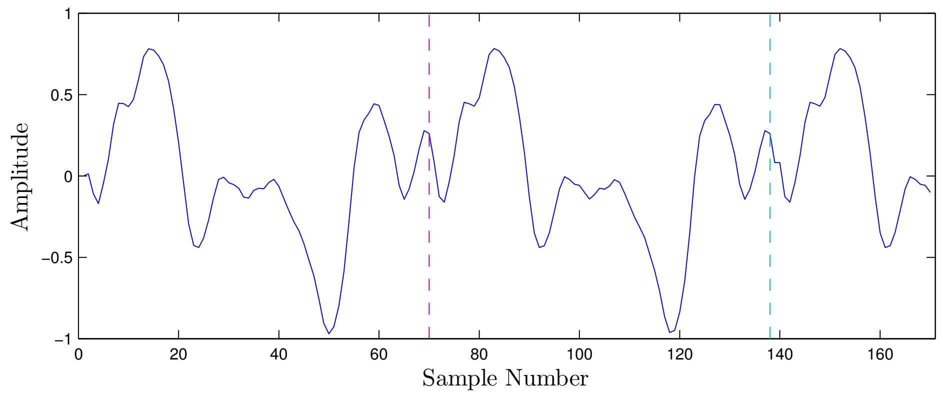


Figure 2.2: A short clarinet sample with the loop region marked. When a note is played using this sample, the first 69 samples are played, the section from samples 70–137 is then repeated to extend the length of the note as required and the final samples (from 138–170) are played at the end of the note.

regarding when each sample should be used is termed a *multisample*.

During playback, "Articulators" can be used to modify the pitch, amplitude and timbre of the sample. The pitch is typically adjusted by varying the size of the steps taken through the sample data – e.g. in order to lower a sample one octave, a "half sample" step is taken through the sample data for each output sample generated, interpolating the sample data appropriately. The amplitude is then adjusted by applying gain related articulators, and the timbre may be modified by specifying resonance and cutoff parameters for a filter.

2.5.5 Combining MIDI and Synthesis: XMF

Combining the event triggering protocol of the SMF files, and the sound definitions from General MIDI and DLS1, the eXtensible Music Format (XMF) [MMA, 2001] was created to allow consistent audio playback of MIDI pieces across audio players and platforms. As the MPEG-4 SA Wavetable Synthesis object type (Section 2.4) supports a MIDI score and an implementation of the MMA DLS standard (SASBF) for synthesis, both XMF and MPEG-4 offer suitable container formats for MIDI + DLS audio coding. Recent mobile phones have support for XMF for ringtones and multimedia audio clips [Copp, 2003].

2.6 Extracting MIDI from Audio

Both the MMA XMF and MPEG-4 SA Wavetable Synthesis standards promote the encapsulation of a MIDI "command" stream with a DLS wavetable synthesiser definition. These are possible formats for "object coded" audio if we can infer MIDI parameters from an audio stream and use this data to drive a DLS synthesiser.

Several groups have looked at estimating MIDI parameters from audio signals. This work has concentrated on estimation of the core Note On and Note Off messages, based on estimations of *onset* and *offset* times, pitch and amplitude.

Sieger and Tewfik [1997] analysed a database of instrument sounds, building a dictionary of audio components from which to resynthesise the signal. Sounds using piano and drums were considered, and were separated into tonal and residual tracks based on the harmonic content. Overlapping tracks were combined into segments which were then modelled using the dictionary and basis pursuit [Chen, 1995]. Audio segments (i.e. onsets and durations) and pitches were extracted showing that simple MIDI extraction from audio was possible.

Modegi and Iisaku [1998] used MIDI to encode physiological sounds (e.g. heart beats and lung sounds) captured by attaching a pair of microphones to stethoscopes. Real-time algorithms were used for peak detection, audio segmentation and note expression, and were coded as a sequence of MIDI Note On and Note Off messages, containing the detected pitch, and encoding the peak level in each sound as the Note On velocity. The Yamaha XG and GS Roland "heartbeat" sounds were used to resynthesise the signal, and judged to be "similar" to the original sound. Experiments on encoding singing data were performed offline, and resulted in features resembling those in the original audio in a low bit-rate format (ca. 6 kb/s, a compression ratio of $\frac{1}{27}$). Although not rigorously evaluated, audio quality was deemed "a little bit poor".

Dixon [2000] synthesised audio files from MIDI data using various General MIDI voices and then produced MIDI files from that audio. The timing and pitch of Note On events was compared, and a score was calculated based on the number of false positives and negatives observed. A wide variation in this score was noted, most sounds having a success rate of 60%-80%, down to ca. 30% when a "Violin" sound was used.

More recently, Bertin et al. [2007] applied non-negative matrix factorisation algorithms and K-SVD techniques to audio to extract onsets, duration and pitches. Similar results were observed for recordings of acoustic instruments and synthesised audio suggesting that, currently, estimating MIDI parameters from synthesised audio is a suitable focus for research - the details that may make the acoustic problem more difficult (e.g. room effects) did not appear to be a significant factor in their results. However, it was also noted that the use of algebraic techniques for feature extraction meant that performance was adversely affected by the length of the piece under consideration - whereas for frame-by-frame algorithms, performance should be, at worst, linear in the length of the piece.

2.6.1 Commercial Applications

Outside the research community, several commercial applications are available which offer audio to MIDI capabilities. These include:

- AKoff Music Composer 2.0 which estimates Note On, Note Off, dynamics and pitch bend [Akoff Sound Labs];
- Intelliscore 8.0 which produces polyphonic note events for multiple instruments using the expression and pitch bend controllers to adjust amplitude and pitch of the output signal [Innovative Music Systems, Inc.];
- TS-AudioToMIDI 2.01 which produces polyphonic Note On and Note Off events [Tallstick Sound Project];
- Melodyne 3.2 which extracts pitch, timing, dynamics, pitch bend and vibrato from monophonic audio producing Note On and Note Off events and using the expression and pitch bend controllers to adjust amplitude and pitch [Celemony].

It is clearly possible to produce a MIDI representation from audio. However, as noted above, MIDI alone done not provide a suitable framework for audio coding – although it can control a synthesiser, the variety of possible synthesisers mean that there is no guarantee of consistency in the output audio. However, we believe that by adopting a specific synthesis model, it should be possible to generate MIDI from source audio material and provide consistent audio output from that MIDI data.

2.6.2 Conclusions

Systems currently exist which produce a MIDI representation from audio using expression and pitch bend to modulate output from a synthesiser. However this representation does not provide a meaningful representation of the modulations (e.g. there is just a series of pitch bend values, rather than any concept of vibrato on a signal). Additionally, the quality of the modulation may be limited by the available bandwidth – to modulate a signal at the audio sample-rate, pitch bend values would need to be provided at that rate.

We considered a selection of audio coding techniques and identified that structured audio would allow the representation of a signal incorporating high-level parameters which (a) are meaningful to a sound designer (e.g. "vibrato") and (b) can be applied by the synthesis engine at the audio sample-rate. Looking to work from existing standards, we found that MPEG-4 Structured Audio Wavetable Synthesis is a possible format with which to build an object coding system for musical audio. Examining related standards from the MIDI Manufacturers Association, we saw that the MPEG-4 SA Structured Audio Sample Bank Format (SASBF) is an implementation of MMA Downloadable Sounds (DLS) and that the MMA have a complementary standard, XMF, which also encapsulates a MIDI control stream with a DLS synthesiser specification. However, no systems currently exist for the production of structured audio parameters from audio.

We believe that it should be possible to (i) infer suitable MIDI parameters from source audio material; and (ii) control a DLS synthesiser using those MIDI parameters to create output audio with features of the source audio material. We therefore looked to find existing technologies which could be combined to create such a system. In the coming chapters, we: consider this problem in more detail (Chapter 3); examine optimisation schemes which may allow us to estimate the parameters (Chapter 4); estimate DLS parameters from audio (Chapter 5); and resynthesise audio based on those parameters (Chapter 6).

Chapter 3

An "Expressive MIDI" model of pitch

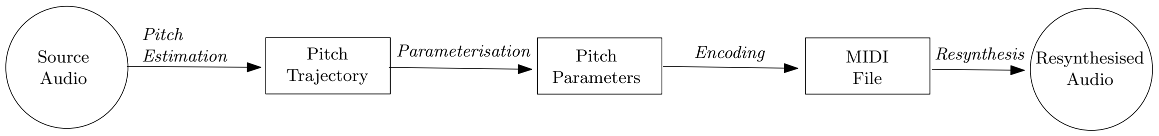


Figure 3.1: System Model: We are looking to find suitable components to build this system

The existing MMA "eXtensible Music Format" (XMF, Section 2.5.5) and MPEG-4 Structured Audio Wavetable Synthesis (Section 2.4) standards allow control of a Downloadable Sounds (DLS) synthesiser (Section 2.5.4) using standard MIDI messages (Section 2.5.1). We wish to create a system which allows a "MIDI + DLS" representation to be created directly from source audio material. As a first stage in this process, we produced an *Expressive MIDI* representation of audio – a MIDI "sketch" of the audio, capturing the pitch expression and representing it using MIDI parameters. In order to do so we needed to be able to: extract pitch information from the source audio material; represent this pitch information as DLS parameters; encode these parameter values in a MIDI stream; and resynthesise audio from that MIDI stream (Figure 3.1).

The core of this system is the representation of the pitch trajectories from the source audio as a set of pitch parameters. We next consider the facilities provided by DLS for representing pitch.

3.1 Representation of Pitch Trajectories in DLS

Within DLS the base pitch for a note is controlled by:

- the MIDI note number;
- MIDI Pitch Bend (MIDI RPN 0 being used to adjust the pitch bend range);
- MIDI RPN 1 to fine tune pitch output (in pitch cents, i.e. $\frac{1}{100}$ths of a semitone);
- MIDI RPN 2 to transpose output by semitones.

This pitch is then modulated using:

- an *Envelope Generator* (EG);
- a *Low Frequency Oscillator* (LFO).

The rate at which the EG and LFO modulators operate is defined by the synthesiser used, allowing full sample-rate signal modulation based on the EG and LFO settings.

3.1.1 dAhDSR Envelope Generator

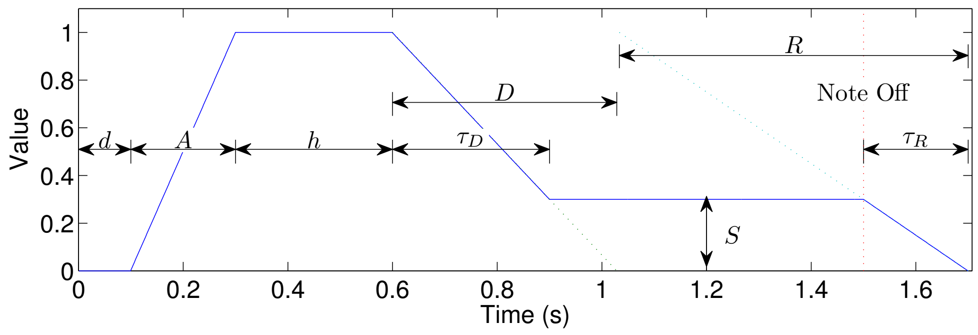


Figure 3.2: Envelope Generator. The MIDI "Note On" event occurs at $t = 0$, the MIDI "Note off' event at the start of the release phase ($t = 1.5$ in this example).

The DLS pitch EG is triggered by the MIDI Note On event and consists of six stages (Figure 3.2). The stages are parameterised by:

- Delay time, d seconds, during which time the EG output is zero;
- Attack time, A seconds, the time taken to reach an output level of 1;

- Hold time, h seconds, the time the EG stays at the peak level;
- Decay rate, D seconds, the time it would take to decay from the peak level to a level of zero;
- Sustain level, S (unitless), indicates the proportion of the maximum EG level (0 to 1) at which the sustain phase is held;
- Release rate, R seconds, the time it would take to decay from 1 to a level of zero (the actual time spent in the release phase τ_{R} depends upon the sustain level).

Hence, such envelope generators are referred to as *dAhDSR* EGs. An additional parameter, the envelope depth, d_{EG} in pitch cents, indicates the peak value during the hold phase. The EG level during the sustain phase is then $S \times d_{\mathrm{EG}}$ pitch cents. The attack phase begins after the MIDI "Note On" message is received and the MIDI "Note Off" event signifies the start of the release phase.

The actual time spent in the decay phase, τ_{D}, is the time required to decay from the "full" EG depth to the sustain level. With a linear EG, this is:

$$\tau_{\mathrm{D}} = (1 - S) \times D \ \text{ seconds.} \tag{3.1}$$

Similarly, the time spent in the release phase, τ_{R}, is the time to decay from the sustain level to zero. Again assuming a linear EG, this is:

$$\tau_{\mathrm{R}} = S \times R \ \text{ seconds.} \tag{3.2}$$

For a linear dAhSDR EG the output for a note of length l seconds is given by

$$EG(t) = \begin{cases} 0 & \text{if } t \leq d \ , \\ \frac{t-d}{A} & \text{if } d < t \leq d + A \ , \\ 1 & \text{if } d + A < t \leq d + A + h \ , \\ 1 - \frac{t-d+A+h}{D} & \text{if } d + A + h < t \leq d + A + h + \tau_{\mathrm{D}} \ , \\ S & \text{if } d + A + h + \tau_{\mathrm{D}} < t \leq l - \tau_{\mathrm{R}} \ , \\ \frac{l-t}{R} & \text{if } l - \tau_{\mathrm{R}} < t \leq l \ , \\ 0 & \text{if } l < t \ . \end{cases} \tag{3.3}$$

where $t, d, A, h, \tau_{\mathrm{D}}, \tau_{\mathrm{R}}$ are in seconds.

3.1.2 DLS Low Frequency Oscillator (LFO)

The DLS pitch LFO is also triggered by the MIDI Note On' event, and has a delay time (δ seconds) and frequency (f Hz). When the MIDI "Note On" event is received, the output of the LFO is zero until the specified delay time has passed, after which a waveform with the specified frequency is generated (Figure 3.3). DLS supports LFOs based on sine and triangle waveforms – we consider a sine wave based LFO in this work.

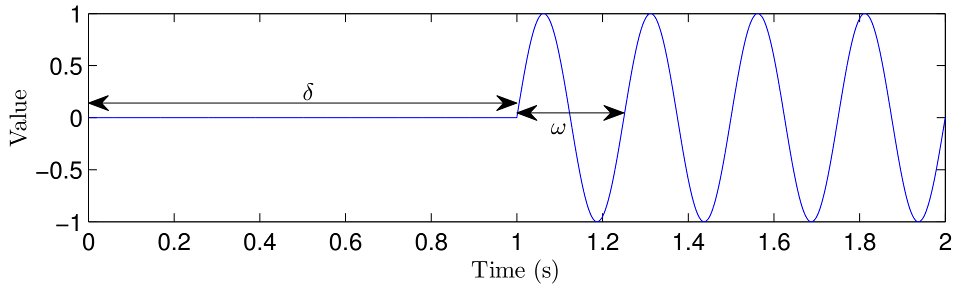


Figure 3.3: Low Frequency Oscillator showing the initial delay, δ seconds, and the period of the LFO, $\omega = \frac{1}{f}$ seconds.

A depth parameter d_{LFO} indicates the range of the LFO in pitch cents. Once triggered, the LFO continues to the end of the note. DLS supports the use of MIDI Controller 1 (Modulation) and MIDI Channel Pressure for adjusting the LFO depth.

The output of the LFO is given by:

$$LFO(t) = \begin{cases} 0 & t \leq \delta \\ \sin(2\pi(t-\delta)f) & \delta < t \leq l \end{cases} \tag{3.4}$$

where t is the time and l the length of the note (both in seconds).

3.1.3 The DLS Pitch Trajectory Model

Combining the envelope generator, the LFO and a base value, b pitch cents, with the relevant depths, an overall DLS pitch trajectory is produced (Figure 3.4) based on the eleven parameter values $(dAhDSR, d_{\mathrm{EG}}, \delta, f, d_{\mathrm{LFO}}, b)$. We refer to this as the "EG+LFO" pitch model.

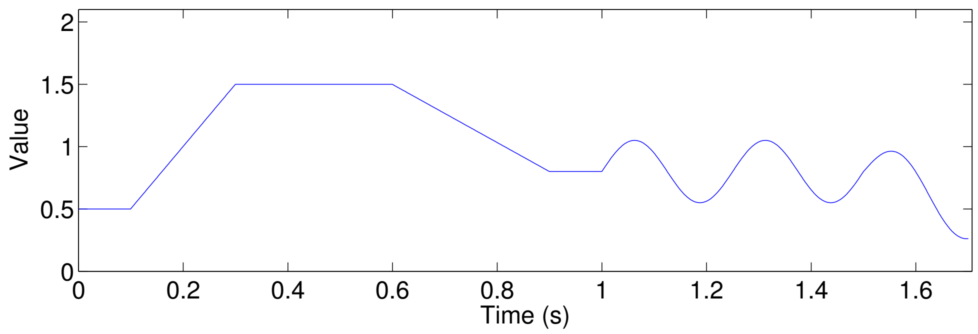


Figure 3.4: Trajectory generated by combining a base value, EG and LFO

The overall pitch trajectory, in pitch cents, is then:

$$e(t) = b + (d_{\mathrm{EG}} \times EG(t)) + (d_{\mathrm{LFO}} \times LFO(t)) \tag{3.5}$$

where b is the base pitch value, d_{EG} the EG depth and d_{LFO} the LFO depth, all in pitch cents (see figure 3.4).

3.1.4 DLS Parameter Formats

The DLS specification includes datatypes for each of the EG and LFO parameters (Table 3.1). For each datatype, a 32-bit integer is used to store the value – a mapping between the native units (time in seconds, pitch in semitones) and the 32-bit integers being defined for each datatype (Table 3.2).

3.1.5 The Expressive MIDI Pitch Model

We have shown how the DLS pitch model supports modulation of the pitch of a sample by specifying a base pitch, an Envelope Generator (EG) and a Low Frequency Oscillator (LFO). The MIDI note number for a note is specified in the Note On and Note Off messages, and the EG and the LFO are triggered by the Note On and Note Off events. We therefore applied the DLS pitch model to individual notes from source audio material – such "per-note" parameters being suitable when the overall pitch trajectory is known at the start of the note (e.g. for audio coding), but inappropriate for performance when the modulators need to be varied during the note. This is the basis of our *Expressive MIDI* pitch model.

In order to use this model, we can: extract the overall pitch trajectory from the source audio; segment this trajectory into individual notes; and estimate the appropriate pitch parameters for each segment. We believe suitable technologies should

Parameter	DLS Connection Block	Units	Min	Max
LFO Frequency	Vibrato LFO Frequency	Absolute Pitch	0.1 Hz	20 Hz
LFO Delay	Vibrato LFO Start Delay	Absolute Time	10 ms	10 s
LFO Depth	Vib LFO to Pitch	Relative Pitch	-1200 cents	1200 cents
EG Delay	Mod EG Delay Time	Absolute Time	0 s	40 s
EG Attack	Mod EG Attack Time	Absolute Time	0 s	40 s
EG Hold	Mod EG Hold Time	Absolute Time	0 s	40 s
EG Decay	Mod EG Decay Time	Absolute Time	0 s	40 s
EG Sustain	Mod EG Sustain Level	Percent	0%	100%
EG Release	Mod EG Release Time	Absolute Time	0 s	40 s
EG Depth	Mod EG to Pitch	Relative Pitch	-1200 cents	1200 cents

Table 3.1: Default DLS Connection Blocks for Pitch Parameters, their datatypes and the range of values allowed [MMA, 2006, pp.30–31].

Units	Data Format	Notes
Absolute Pitch	$6553600 \times \left(12 \log_2 \left(\frac{f}{440}\right) + 69\right)$	f frequency in Hz
Absolute Time	$1200 \times \log_2 (t) \times 65536$	t time in seconds, the value 80000000h being reserved to represent a time of exactly 0 seconds
Relative Pitch	$65536 \times \Delta p$	Δp is the change in pitch in cents
Percent	*Not specified by MMA*	Δp is the change in pitch in cents

Table 3.2: DLS 32-bit integer representations of datatypes [MMA, 2006, p.32]

exist for this – pitch estimation being a well-established research area – and look to build a system incorporating existing technologies.

3.2 Extracting a Pitch Trajectory from Audio

Pitch is a perceptual quality of a sound. It is defined with reference to the pitch sensation created by sine waves – the pitch of a sound having a logarithmic relationship to the frequency of the sine wave which creates the same "pitch sensation" [Sethares, 1997]. We note that pitch is only defined for certain discrete values, whereas frequency is continuous. Most sounds are not simple e.g. for harmonic signals, the fundamental frequency would give the pitch, but this frequency may not be present in the actual signal and in polyphonic music, several pitched notes will be present in the signal at the same time.

We are interested in pitch estimation for analysis/resynthesis audio coding. For accurate and high resolution resynthesis of audio, we need an accurate and high-resolution pitch estimator. The minimum difference between two pitches that can be perceived by a listener is termed the just noticeable difference (JND). The JND for two separate pure tones is ca. 10 cents (i.e. $\frac{1}{10}$ semitone), the JND for complex tones being smaller [Loy, 2006]. In order to synthesise audio with the same perceived pitch, a pitch error of less than 10 cents is therefore sought. For pitch variation within a note, the JND for pitch modulation (i.e. the smallest noticeable change in pitch) was measured as 2-3 cents for trained musicians by Sethares [1997].

Block-based pitch estimators operate on windows of samples taken at certain times in the signal (often equally spaced time intervals). A pitch detection function is then applied to the block of data, indicating how well the pitch is represented by a set of candidate solutions. The best candidate solution is then selected using a peak picking algorithm. The most fundamental pitch detection functions are autocorrelation and the Fourier transform. We now introduce the use of these for pitch detection.

3.2.1 Pitch Estimation Using Autocorrelation

The simplest approach to fundamental frequency estimation relies on the correlation of a signal with itself at various offsets over a window of W samples (the autocorrelation function, ACF), and the selection of a (non-zero) offset which gives the greatest

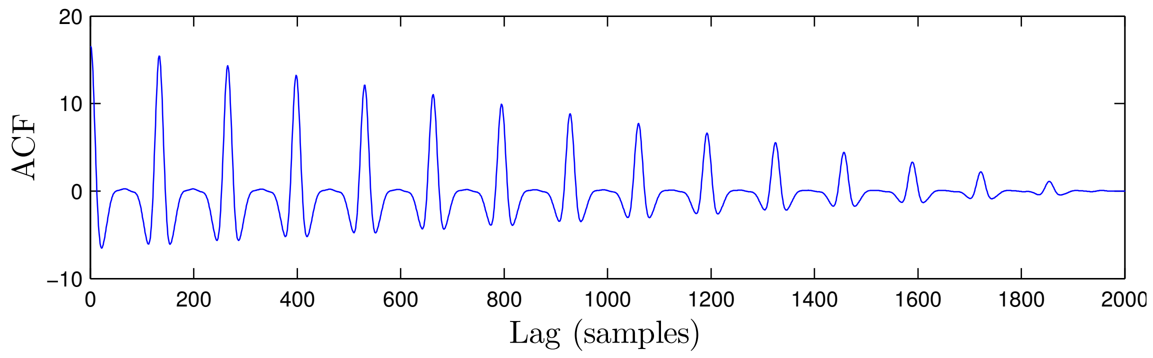


Figure 3.5: Example plot of the autocorrelation function (ACF) for a segment from a vibrato trumpet note.

correlation.

$$\omega(t) = \operatorname*{argmax}_{\tau} \left(\sum_{i=0}^{W-1} x_{t+i} x_{t+\tau+i} \right) \tag{3.6}$$

This gives an estimate of the period of the signal, ω, in samples, and hence, using the sample rate s_{R}, an estimate of the fundamental frequency, f:

$$f = \frac{1}{\omega} \text{ Hz.} \tag{3.7}$$

The pitch estimate (in terms of MIDI pitch) is then:

$$\begin{aligned} p_{\mathrm{AC}} &= 12 \log_2 \left(\frac{f}{440} \right) + 69 \\ &= \log_2 \left(\frac{s_{\mathrm{R}}}{440} \right) + 69 - \log_2 \omega \ . \end{aligned} \tag{3.8}$$

Hence, we see that the *pitch resolution* of an autocorrelation based pitch estimator is greatest for *low* pitches (i.e. large periods).

Picking the offset with the greatest correlation is not perfect. Often, the signal is highly correlated for offsets of a few samples (as in Figure 3.5), and harmonics may be more prominent than the fundamental frequency itself. It is therefore common to look for the first peak in the correlation after the first zero crossing rather than simply the greatest correlation. In particular, "octave errors" occur when the 2nd harmonic is more prominent than the fundamental frequency.

3.2.2 Pitch Estimation Using the Fourier Transform

Pitch can also be estimated from the power spectrogram of a signal (i.e. the square amplitude of the discrete-time short-term Fourier transform), by selecting the frequency bin with the highest power. This gives a direct estimate of the frequency – the resolution being limited by the window size used.

The Fourier transform is based around the assumption of a signal repeating with period N. This assumption generally does not hold, introducing aliasing errors. It is therefore usual to use a windowing function, $w(n)$ to reduce these errors.

$$f = \frac{s_{\mathrm{R}}}{N} \operatorname*{argmax}_k \left| \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} w(n) x_n e^{-2\pi i \frac{kn}{N}} \right|^2 \quad Hz \tag{3.9}$$

where $k > 0$

The Fourier transform pitch estimate is based on a direct estimate of the *frequency* bin, k, and the associated MIDI pitch is:

$$\begin{aligned} p_{\mathrm{FT}} &= 12 \log_2 \left(\frac{f}{440} \right) + 69 \\ &= \log_2 \left(\frac{s_{\mathrm{R}}}{N} \right) + 69 + \log_2 k \ . \end{aligned} \tag{3.10}$$

The pitch resolution of a Fourier transform based pitch estimator is therefore greatest for *high* pitches (i.e. large frequencies).

Various techniques have been proposed for improving the estimation of the position of the peak within the DFT, using interbin estimates rather than the discrete bin numbers [Jacobsen and Kootsookos, 2007].

3.2.3 YIN

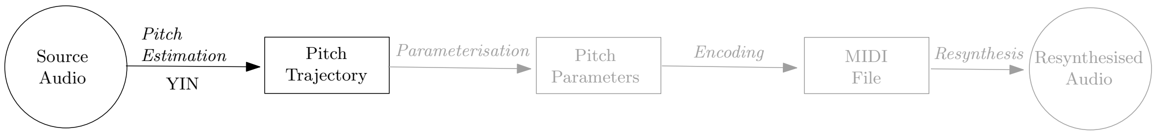


Figure 3.6: System Model: YIN, the first component of the system

At the start of this work, the YIN algorithm [de Cheveigné and Kawahara, 2002] was the state-of-the-art in pitch estimation. We therefore elected to use it to estimate pitch trajectories (Figure 3.6). YIN is inspired by the autocorrelation approaches to pitch estimation, but applies several adjustments to give a more accurate estimate. Given a signal and a hop size, h (default 32), YIN gives instantaneous estimates of pitch, power and aperiodicity each h samples.

Rather than using the autocorrelation directly, YIN calculates the square error difference function at time t

$$E(\mathbf{x}, t, \tau) = \sum_{i=1}^{W} (x_{t+i} - x_{t+i+\tau})^2 \tag{3.11}$$

where $\mathbf{x}$ is the audio signal, W the window size, and τ an offset. This is closely related to the autocorrelation as $\sum_i (x_i - x_{i+\tau})^2 = \sum_i x_i^2 - 2x_i x_{i+\tau} + x_{i+\tau}^2$ and $\sum_i x_i x_{i+\tau}$ is the autocorrelation function.

YIN has three options for calculating the difference function: holding the earlier window fixed and shifting the later window; shifting the earlier window with the later window fixed; and shifting both windows away from a centre point. We used the third of these options – the default mode for YIN. Using this scheme, the offset of the earlier window from the sample under consideration is:

$$\delta_e(\tau) = \left\lfloor \frac{\tau_{\max}}{2} \right\rfloor - \left\lfloor \frac{\tau}{2} \right\rfloor \tag{3.12}$$

samples where $\tau_{\max}$ is the maximum lag under consideration

$$\tau_{\max} = \frac{s_{\mathrm{R}}}{f_{min}} \tag{3.13}$$

where s_{R} is the sample rate and f_{min} the minimum frequency to be considered (default 30Hz). Similarly, the later window is offset by

$$\delta_l(\tau) = \left\lfloor \frac{\tau_{\max}}{2} \right\rfloor + \left\lceil \frac{\tau}{2} \right\rceil \ . \tag{3.14}$$

We note that $\delta_l(\tau) - \delta_e(\tau) = \left\lceil \frac{\tau}{2} \right\rceil + \left\lfloor \frac{\tau}{2} \right\rfloor = \tau$ as $\left\lceil \frac{\tau}{2} \right\rceil = \left\lfloor \frac{\tau}{2} \right\rfloor = \frac{\tau}{2}$ for τ divisible by 2 with $\left\lceil \frac{\tau}{2} \right\rceil = \frac{\tau}{2} + 1$ and $\left\lfloor \frac{\tau}{2} \right\rfloor = \frac{\tau}{2} - 1$ otherwise.

Using these timings, YIN finds the ith period estimate by minimising the difference function

$$\tau^*(i) = \operatorname*{argmin}_{\tau} \sum_{j=0}^{w-1} \left(x_{(i-1)h+\delta_e(\tau)+j} - x_{(i-1)h+\delta_l(\tau)+j} \right)^2 \ . \tag{3.15}$$

The ith pitch output from YIN, $i = 1 \ldots n$, is then

$$p_{\mathrm{YIN}} = \log_2 \left(\frac{s_{\mathrm{R}}}{440\tau^*((i-1)h)} \right) \tag{3.16}$$

octaves from 440 Hz and the MIDI pitch is given by $p_{\mathrm{MIDI}} = 12p_{\mathrm{YIN}} + 69$.

The precise timing at which the pitch estimate applies within the analysis window is not known. However, with no further information, the least biased estimate is the centre of the analysis window – as reversing the order of the samples considered will produce the same values for the difference function. The size of the YIN analysis window is the specified window size, w, plus the maximum lag considered in the YIN algorithm, $\tau_{\max}$. The estimated timing, t_i, in samples for the ith pitch estimate is:

$$t_i = (i-1)h + \left(\frac{w + \tau_{\max}}{2}\right) \ . \tag{3.17}$$

Noting that the default window size is $w = \left\lceil \frac{s_{\mathrm{R}}}{\mathrm{minf0}} \right\rceil = \tau_{\max}$ samples,

$$t_i = (i-1)h + \left\lceil \frac{s_{\mathrm{R}}}{\mathrm{minf0}} \right\rceil \tag{3.18}$$

samples if the default window size is used.

Given a period estimate of $\tau^*(i)$ samples, YIN calculates estimates of the power, P, and aperiodicity, a, of the signal (Figure 3.7):

$$\begin{aligned} \mathrm{P}(\tau^*(i), i) &= \sum_{j=0}^{\tau-1} x^2_{(i-1)h+j} \\ \mathrm{a}(\tau^*(i), i) &= \frac{\sum_{j=0}^{\tau-1} \left(x_{(i-1)h+j} - x_{(i-1)h+j+\tau}\right)^2}{2\sum_{j=0}^{\tau-1} x^2_{(i-1)h+j} + x^2_{(i-1)h+j+\tau}} \end{aligned} \tag{3.19}$$

We see that YIN calculates the power estimate across the first period in the analysis window. The timing for the power estimate, in samples, is therefore

$$t_{\mathrm{pwr}} = (i-1)h + \frac{\tau^*(i)}{2} \tag{3.20}$$

where $(i-1)h$ is the time at the start of the analysis window and $\tau^*(i)$ the period estimate. Similarly, the YIN aperiodicity estimate is calculated across the first two periods in the window. The timing for the aperiodicity estimate, in samples, is therefore

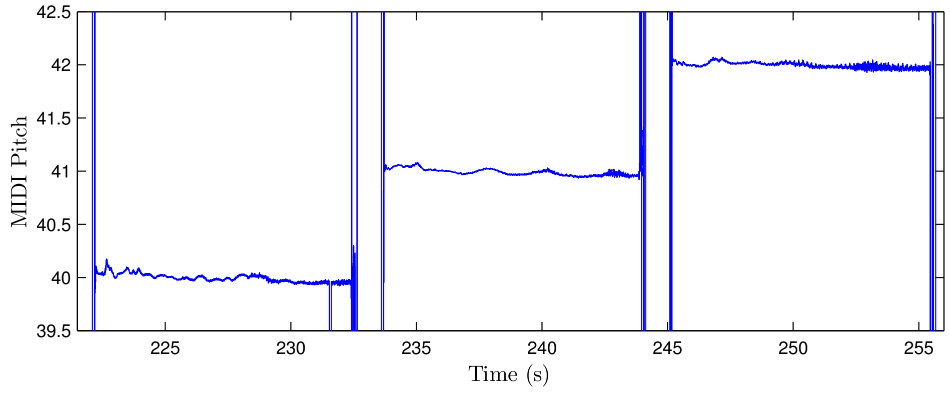
$$t_{\mathrm{ap}} = (i-1)h + \tau^*(i) \ . \tag{3.21}$$

Hence, the pitch, power and aperiodicity estimates are *not* time-aligned. However, the power and aperiodicity estimates depend upon an accurate pitch estimate and should therefore be calculated at the same timing as the pitch.

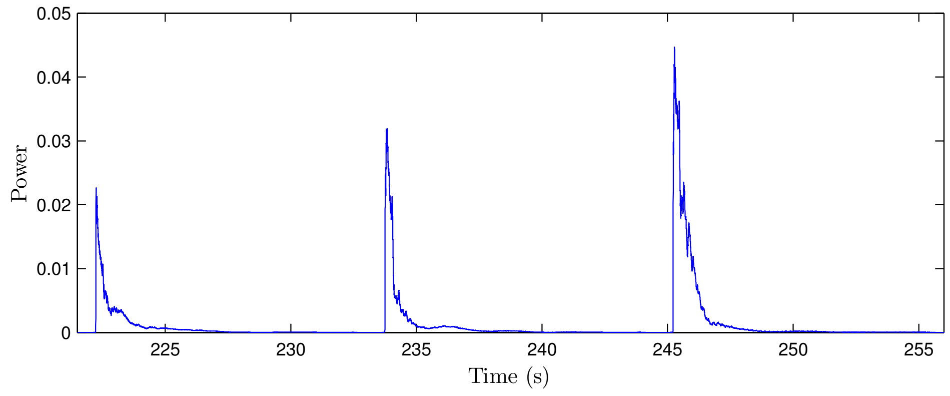
Although we now know that the YIN power and aperiodicity estimates are unreliable, our focus is on building the Expressive MIDI system and, as YIN was the state-of-the-art, we adopted it for our system.

3.3 Segmenting the Pitch Trajectory

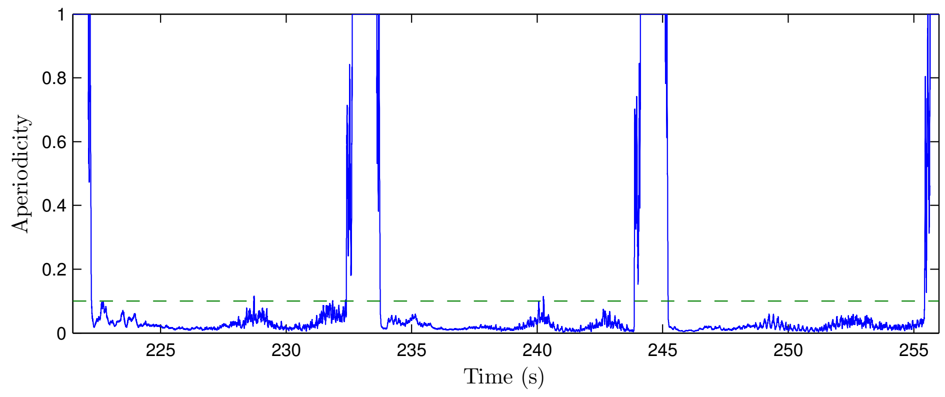
The Expressive MIDI pitch model (Section 3.1.5) applies to individual notes within a musical signal – the EG and LFO being triggered by the MIDI Note On and Note Off



(a) YIN pitch estimate



(b) YIN power estimate



(c) YIN aperiodicity. Aperiodicity threshold of 0.1 shown as dashed line.

Figure 3.7: Example output from YIN – Pitch, power and aperiodicity trajectories for three piano notes.

events. We want to estimate parameters to use with this model, and therefore need to examine individual notes from the musical signal. In order to do this, we need to segment the audio into individual notes.

The YIN aperiodicity indicates the proportion of the signal power which is from non-pitched sources – i.e. the output that is not explained by the estimated periodicity. Low aperiodicity values indicate the sections of the YIN output at which the signal pitched – given a perfectly periodic signal and an accurate estimate of the period the aperiodicity will be 0.

In order to test our Expressive MIDI system, we will examine samples of audio from the RWC Musical Instrument Sounds database [Goto et al., 2003]. The database contains audio for 123 instruments and each instrument has files covering various articulations (styles of playing) and dynamics (how loudly the instrument is played). Each file consists of a chromatic sequence of notes covering the complete range of an instrument (i.e. notes from the lowest to highest pitch produced by the instrument at 1 semitone intervals) separated by silences. As the individual notes in the files are separated by silences, we assumed that any contiguous pitched segments within the files would be individual notes. Our segmentation procedure therefore consisted of finding contiguous segments of YIN pitch estimates longer than 0.1 seconds with an aperiodicity level less than 0.1.

Given an aperiodicity threshold, α, the set of indexes, $\mathbf{S}$, where the aperiodicity fell below this threshold was found:

$$\mathbf{S} = \{i : (\mathrm{a}(\tau^*(i), i) <= \alpha) \wedge (\mathrm{a}(\tau^*(i-1), i-1) > \alpha)\} \tag{3.22}$$

giving the start points for regions of low aperiodicity. Similarly, the set of indexes, $\mathbf{E}$, where the aperiodicity rose above this threshold was found:

$$\mathbf{E} = \{i : (\mathrm{a}(\tau^*(i), i) <= \alpha) \wedge (\mathrm{a}(\tau^*(i+1), i+1) > \alpha)\} \tag{3.23}$$

giving the end points for regions of low aperiodicity. Contiguous pitched segments were then found from start points $s \in \mathbf{S}$ to end points $e \in \mathbf{E}$. The lengths of the segments (in seconds) were then calculated $l = (e - s) * \frac{h}{s_{\mathrm{R}}}$, where h is the YIN hop size in samples and s_{R} the sample rate of the original audio in samples per second. Segments longer than 0.1 seconds were kept and used as the source pitch trajectories which we aimed to match (Matlab 7.2 source code for the pitch trajectory segmentation is shown in Listing 3.1).

For the rest of this thesis, we will use the term *pitch trajectory* to refer to an individual note pitch trajectory unless otherwise stated.

Listing 3.1: Matlab Script for segmenting YIN output r

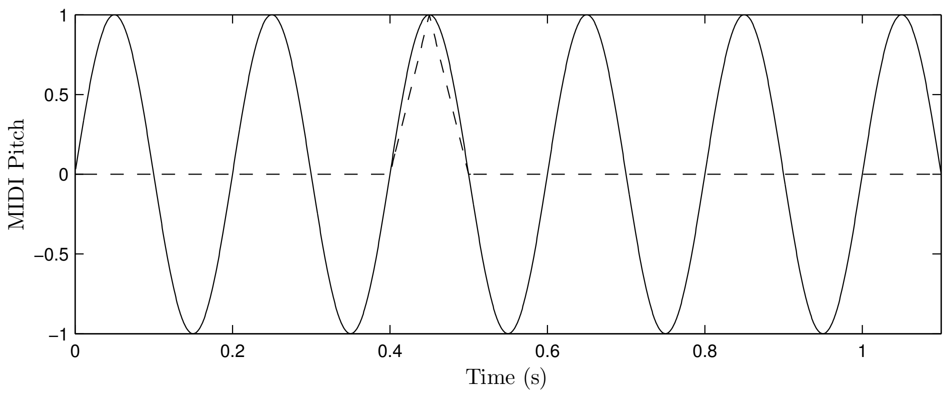
```
% Estimate "best" estimates of pitch
nonperiodic = (r.ap0' > r.plotthreshold);
changeval = diff(nonperiodic);
changeval = [2 * nonperiodic(1) - 1; changeval; 1 - 2 * nonperiodic(end)];

toperiodic = find(changeval == 1);
fromperiodic = find(changeval == -1);

if toperiodic(1) < fromperiodic(1)
    fromperiodic = fromperiodic(2:end);
end
if toperiodic(end) > numel(nonperiodic)
    if fromperiodic(end) == numel(nonperiodic)
        toperiodic = toperiodic(1:(end - 1));
        fromperiodic = fromperiodic(1:(end - 1));
    else
        toperiodic(end) = numel(nonperiodic);
    end
end

toperiodic = toperiodic(1:numel(fromperiodic));

b = [fromperiodic toperiodic];

bs = b * r.hop / r.sr; % times in seconds
bl = bs(:, 2) - bs(:, 1);

% Only keep longer segments
bthreshold = 0.1; % minimum length in seconds
bc = bl > bthreshold;
bs = bs(bc, :);
```

3.4 Estimating Pitch Trajectory Parameters

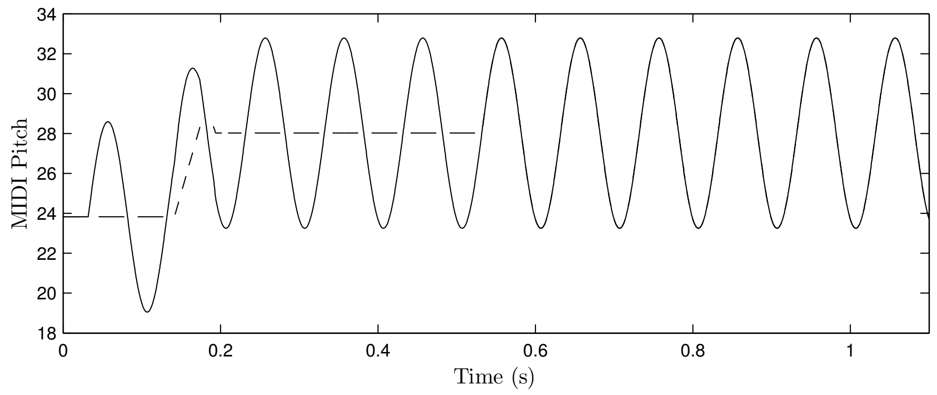
As part of our Expressive MIDI system, we have selected the YIN pitch algorithm to extract pitch trajectories from audio and have used the YIN aperiodicity to produce segments from the overall pitch trajectories. To complete the system, we need to estimate parameters for these pitch trajectories and resynthesise audio from the parametric representation[1].

The DLS standard includes data format definitions specifying 32-bit representations for units of pitch, time, gain and frequency (Section 3.1). These bit-wise representations form the basis of our encoding of EG and LFO parameters and bit-wise optimisation algorithms are therefore considered. Estimating the trajectory parameters is a difficult problem, the cost function having local minima e.g. when the LFO (or EG) approximates the function of the EG (or LFO) (Figure 3.8a) and each time the LFO delay is changed by the period of the LFO (Figure 3.8b). We next examine several bit-wise optimisation algorithms, and consider the effect of the bit-wise coding used on algorithm performance.

[1]We note that although the following work specifically considers the DLS representations of EG and LFO parameters, pitch trajectory resynthesis could occur using any synthesiser for which the EG+LFO pitch model was appropriate.



(a) EG Matching LFO, the LFO output being matched for half a cycle by the EG output



(b) LFO delayed a number of periods, the trajectories matching from $t = 0.55$ seconds

Figure 3.8: Example local minima in matching trajectories

Chapter 4

Optimisation

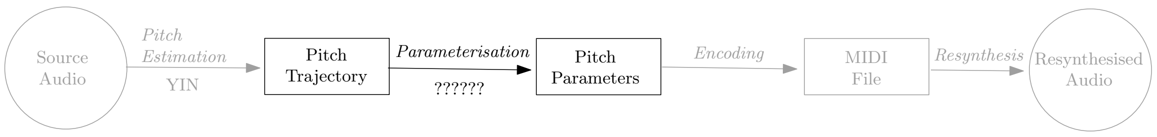


Figure 4.1: System Model: Having selected YIN to estimate pitch parameters, we need to select an optimisation technique to estimate the pitch parameters

In order to estimate pitch parameter values from the YIN pitch trajectories, we need to choose a suitable optimisation technique (Figure 4.1). The pitch trajectory estimation problem is presented in terms of the DLS 32-bit representation (Section 3.1.4), which specifies a mapping from integers (encoded using 32-bit Standard Binary) to the actual parameter values. Hence, we considered optimisation techniques which operate directly on a bit-wise representation (bit-wise optimisation algorithms).

Alternative 32-bit encodings may be more effective during the optimisation process – changing the encoding can change the number of local optima. We therefore examined both bit-wise optimisation techniques and encodings. Mathias and Whitley [1994] examined the performance of Steepest Descent, a Stochastic Hill Climber and Eshelman's CHC [Eshelman, 1991] using a standard Binary code and the Binary Reflected Gray Code (BRGC) [Gray, 1953]. We extended this analysis to include: the simple Genetic Algorithm; and the less known Monotone and Balanced Gray codes. After introducing the algorithms (Sections 4.1 and 4.2) and the binary encodings (Section 4.4), we present the results of our analysis.[1]

[1]Parts of this work were originally published as a Technical Report by the Centre for Digital Music, Queen Mary University of London [Welburn and Plumbley, 2009b].

Optimisation techniques aim to find the location of the optimum value of an *objective function* over a *search space* of possible locations. The objective function may be termed a *fitness function* for maximisation problems (the aim being to find the "fittest" solutions) or a *cost function* for minimisation problems (the aim being to minimise the "cost"). In order to optimise an objective function, search algorithms start with one or more candidate solutions and update them, apply some pressure towards producing "better" solutions e.g. by directly moving to better solutions (see Steepest Descent and SHC, Section 4.1) or by giving preference to better solutions as the basis for new candidate solutions (see the Simple GA and CHC, Section 4.2). Individual solutions are evaluated using the objective function and search continues until the value of the best solution meets some pre-determined criteria (e.g. a small difference to a target value or no improvement for a number of iterations) or until a fixed number of iterations or objective function evaluations have occurred.

In order to find the global optimum value, and to know that it actually is the global optimum, either an exhaustive search of the entire space of possible solutions must be carried out, or we must use information from the problem domain to solve the problem. If neither of these are appropriate (e.g. an exhaustive search would take too long) then we must trust that the technique being used will provide a good enough solution for our needs. For the pitch parameter estimation problem we are looking to find "good enough" pitch parameters rather than the theoretical optimum. As the space may contain multiple local optima, a search purely using local information (e.g. gradients) may not find the global optimum – both local and global searches must be used to find the region in which the best solution occurs, and to search that region for the best solution. These are termed *exploitation* of local features and *exploration* of the solution space [Holland, 1992].

We considered two hillclimbing/descent methods (Steepest Descent and a Stochastic Hillclimber) and two Evolutionary Algorithms (a Simple Genetic Algorithm and CHC). We next describe these algorithms.

4.1 Hillclimbing/Descent Methods

Hillclimbing and Descent methods take a single candidate solution and then repeatedly move to solutions for which the objective function gives a higher (Hillclimbing) or lower (Descent) value until no new position can be found. As a maximisation problem can be converted to a minimisation problem by considering the alternative objective function $\tilde{f}(\mathbf{x}) = -f(\mathbf{x})$, they are fundamentally the same types of techniques but with different problem domains.

4.1.1 Steepest Descent

Steepest Descent [Burden and Faires, 1997] is based on the principle that a local minimum of a function can be found by selecting a start point and successively taking small steps "down" in the direction of the gradient of the function.

$$\mathbf{x}^{i+1} = \mathbf{x}^i - \epsilon \nabla f(\mathbf{x}^i) \tag{4.1}$$

For non-differentiable $f(\cdot)$, Steepest Descent is performed by examining the function value for small changes to each individual dimension of $\mathbf{x}$, the dimension that produces the largest decrease to $f(\cdot)$ then being updated. For bitwise Steepest Descent the only possible change to each dimension is to invert the relevant bit ($y_i \leftarrow \bar{y}_i$).

For bit-wise data, Steepest Descent starts by selecting a, typically random, start point. All neighbours of the current point are considered, the neighbours being those candidate solutions that differ by a single bit (so called "Hamming" neighbours). If any better values are found, then the current point is moved to the neighbour with the best value. This process is repeated until no better value can be found, at which point the algorithm terminates and returns the current point.

Algorithm 1 *Bit-wise Steepest Descent*

```
set $\mathbf{x}^*$ to a random start position
repeat
    $\mathbf{x} = \mathbf{x}^*$;
    for $i = 1$ to $n$
        $\mathbf{y} = \mathbf{x}$;
        $y_i = 1 - x_i$;
        if $f(\mathbf{y}) < f(\mathbf{x}^*)$ then
            $\mathbf{x}^* = \mathbf{y}$;
        end if
    next $i$
until $(\mathbf{x} = \mathbf{x}^*)$
```

Where: $\mathbf{x} = (x_1, \ldots, x_n)$ *and* $\mathbf{y} = (y_1, \ldots, y_n)$ *are n-bit vectors; and* $f(\cdot)$ *is the function to be minimised.*

The equivalent ascent technique for maximisation would use $\mathrm{f}(\mathbf{y}) > \mathrm{f}(\mathbf{x}^*)$ as the test condition. This is equivalent to minimising $\tilde{\mathrm{f}}(\mathbf{x}) = -\mathrm{f}(\mathbf{x})$.

After selecting the start point, the algorithm is deterministic and will give the same results each time – the search landscape (i.e. the neighbours for a given point) being defined by the encoding. Steepest Descent exploits local features and moves to a local optimum – no wider exploration is made which could increase the probability of ending at better optima. Finding the global optimum requires that the start point is in the same "valley" as the solution – its "basin of attraction".

There are problems where it is not possible to find a local optimum solution using Steepest Descent. For *ridge problems*, updating individual dimensions leads the candidate solution onto a "ridge" of solutions which are better than the solutions off the ridge. Consider the function $f(x) = (x - y)^2 + \frac{1}{x+y}$, $x, y \in \mathbb{Z}$. Steepest Descent will move the candidate solution onto the ridge at $x = y$ and then changing either x or y will move the candidate off the ridge to a worse solution, every point on the ridge appearing to be a local optimum. It is necessary for the candidate solution to change both x and y to find better solutions and therefore Steepest Descent will fail [Rowe et al., 2004].

The only parameter required for bit-wise Steepest Descent is the function, $f(\cdot)$, to be optimised.

4.1.2 Stochastic Hill-Climber (SHC)

Rather than examining all neighbours to find the "best" move, a Stochastic Hill Climber (SHC) (also known as a Random Mutation Hill Climber (RMHC)) randomly selects a new point based on the current point and moves if the new point is better than the current candidate solution. Although this (i) may result in moves to less optimal neighbours than Steepest Descent and (ii) sometimes remains at the same point, the concomitant reduction in the number of function evaluations required (only a single function evaluation being required with each generation) can produce performance improvements over Steepest Descent.

Two versions of the SHC/RMHC exist in the literature: Forrest and Mitchell [1993] stating that the new point should be a *neighbouring* value, selected with a uniform probability; and Whitley et al. [1996] stating that the new point should be selected by mutating *every* bit with a low uniform probability. As for bit-wise Steepest Descent, only mutating neighbouring values will find a local optimum based on the start position – this local optimum may be a different local optimum to that found by Steepest Descent as the steepest gradient is not followed. Mutating all bits extends the search to consider both neighbours and other points in the search space and will no longer be restricted to finding solutions in the basis of attraction of the start point. In this work, we use Whitley et al.'s definition of SHC and mutate all bits.

The bit-wise Stochastic Hill-Climber again starts with a single, typically random, point, but then examines all bits, mutating each with a fixed probability, p. If the fitness at the new point is better, then we move to that point. SHC continues until a termination condition is reached (e.g. not being able to move to a better position after some number of attempts, or a "good enough" fitness being reached).

Algorithm 2 *Bit-wise Stochastic Hill Climber*

```
set $\mathbf{x}$ to a random start position
repeat
    $\mathbf{y} = \mathbf{x}$;
    for $i = 1$ to $n$
        choose $q$ at random from uniform over $[0, 1]$
        if $q < p$ then
            $y_i = \bar{y_i}$;
        end if
    next $i$
    if $\mathrm{f}(\mathbf{y}) > \mathrm{f}(\mathbf{x})$ then $\mathbf{x} = \mathbf{y}$;
until finished
```

Where: $\mathbf{x} = (x_1, \ldots, x_n)$ *and* $\mathbf{y} = (y_1, \ldots, y_n)$ *are n-bit vectors;* $0 \leq p \leq 1$ *is the probability of a bit flipping; and* $\mathrm{f}(\cdot)$ *is the function to be minimised.*

Using a mutation probability of $\frac{1}{n}$, where n is the length of the bit-string, on average a single bit will be expected to be mutated each generation. However, unlike Steepest Descent, which considers all neighbours and finds the "best" neighbour, SHC will move to the first better solution found. Additionally, there is the possibility of more than one bit being mutated in each generation, allowing the algorithm to explore the search space rather than simply searching locally for a solution.

In order to apply SHC to a minimisation problem, the test condition in Algorithm 2 can be changed to $\mathrm{f}(\mathbf{y}) < \mathrm{f}(\mathbf{x})$ – this is equivalent to maximising the complementary problem $\tilde{\mathrm{f}}(\mathbf{x}) = -\mathrm{f}(\mathbf{x})$.

Since random mutations are tested, the algorithm can give different results each time it is run. Additionally, as it is possible to reach any point in the search space from the current point (albeit with decreasing probabilities depending on the number of differing bits), all points in the space may be considered i.e. it is possible eventually to find the solution for any optimisation problem. However, if the set of points that lead to the global solution – its "basin of attraction" – is small relative to the search space, it may take many generations to enter the basin and find that optimum.

The parameters required for SHC are: the function to be optimised, $f(\cdot)$; the mutation probability p; and the termination parameters – e.g. the maximum total number of iterations or the maximum number of iterations without finding a better solution.

4.2 Evolutionary Algorithms (EAs)

Inspired by biological evolution, Evolutionary Algorithms (EAs) apply the operators recombination, mutation and selection to "evolve" a population of candidate solutions to maximise some *fitness function*.

Recombination and mutation are based on internally representing *phenotypes* (the candidate solutions) as *genotypes* (or *chromosomes*) composed of component *genes*. Mapping candidate solutions from the problem space (of phenotypes), $\mathcal{P}$, to the search space (of genotypes), $\mathcal{S}$, is accomplished using an encoding $\Phi : \mathcal{P} \to \mathcal{S}$. The encoding Φ is not necessarily a function, i.e. a one-to-one mapping between representations and candidate solutions – a candidate solution could have multiple possible representations e.g. in the Unary Code, values $0 \ldots n$ are represented by a series of n zeroes and ones, the represented value being determined by the number of 1s in the representation. In order to apply the power of recombination and mutation to a problem, a suitable representation is required.

Candidate solutions are tested using the fitness function, defined on the phenotypes. A decoding function Θ, that covers the space $\mathcal{P}$, is therefore required with which each phenotype is mapped deterministically to the original genotype i.e. $\Theta : \mathcal{S} \to \mathcal{P}$ where $x = \Theta(\Phi(x))$.

EAs are regarded as obtaining their power from a combination of *exploration* (searching over a large space) and *exploitation* (finding the best performance locally). Eiben and Schippers [1998] reviewed the existing literature and concluded that there was no common understanding of how exploration and exploitation contribute to that power. However, in general terms, exploration increases population diversity to search new areas whilst exploitation decreases population diversity and examines local structure.

The only domain knowledge EAs require is the ability to calculate a fitness function based on the candidate solutions, allowing them to be applied to arbitrary problems. Domain knowledge may be incorporated into EAs by applying them to part of the problem space (e.g. a subset of the parameters) and using additional optimisation techniques within the calculation of the fitness function (e.g. finding the additional parameters using local searches or algebraic solutions). This is termed "hybridisation". EAs which include local search techniques to select the local optimum solution after recombination and mutation are termed *memetic* algorithms [Moscato, 1989].

We considered two Evolutionary Algorithms, a Simple Genetic Algorithm and Eshelman's CHC, described below.

4.2.1 Simple Genetic Algorithm

The Simple Genetic Algorithm (GA) operates on a population of m bit-wise representations of candidate solutions i.e. the genotype is a string of 1s and 0s. This population is evolved using *recombination* and *mutation*. Recombination selects "parent" solutions based on fitness values – candidates with higher fitness values ("better" candidate solutions) having a higher probability of *selection* – and then combines these solutions using *crossover* (see below) to produce a child solution. The fitness function, $f(\cdot)$ is used to select the "parent" solutions – only the "fittest" solutions getting to breed. Mutation randomly changes bits in the updated population, all bits having the same fixed probability, p, of mutating.

In order to apply a Simple GA to a minimisation problem, the complementary maximisation problem $\tilde{\mathrm{f}}(\mathbf{x}) = -\mathrm{f}(\mathbf{x})$ can be considered – however a suitable selection scheme then needs to be used, as the standard "roulette wheel" selection scheme for Simple GAs [Mitchell, 1998, pp. 166–167] is designed to work with positive functions. A rank-based selection scheme [Mitchell, 1998, p. 169] allows a GA to be directly applied to the minimisation problem by ranking solutions based on how low their fitness is rather than how high.

For the Simple GA, the parameters required are: the function to be optimised, $f(\cdot)$; the population size, m; the mutation probability, p; and the termination parameters – e.g. the maximum total number of iterations or the maximum number of iterations without finding a better solution.

Crossover Schemes for Genetic Algorithms

The Simple Genetic Algorithm (Simple GA) uses single-point crossover. Single-point (1-point) crossover selects a position within the bit-string and exchanges all bits beyond that point between the two parents. Table 4.1 shows the offspring A^* and B^* produced from 8-bit values A and B using single-point crossover at bit 6. For high-dimensional data, single-point crossover explores by exchanging entire sets of dimensions from the start and end of the parents, and uses parts of the dimension at the crossover position to generate a new value. Values from individual dimensions will be present in greater numbers for fitter members of the population. Those values will propagate in turn.

Parents	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
	B	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
Offspring	A^*	a_1	a_2	a_3	a_4	a_5	b_6	b_7	b_8
	B^*	b_1	b_2	b_3	b_4	b_5	a_6	a_7	a_8

Table 4.1: Single Point Crossover

Alternative crossovers include:

- Two-point (2-point) crossover, in which bits between two selected positions are exchanged between the parents. Table 4.2 shows the offspring A^* and B^* produced from 8-bit values A and B using 2-point crossover at bits 3 and 6.

Parents	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
	B	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
Offspring	A^*	a_1	a_2	b_3	b_4	b_5	a_6	a_7	a_8
	B^*	b_1	b_2	a_3	a_4	a_5	b_6	b_7	b_8

Table 4.2: Two Point Crossover

- Half Uniform Crossover (HUX), in which the bits that differ between the parents are found, and half these bits are uniformly randomly selected and flipped. Table

4.3 shows the offspring A^* and B^* produced from 8-bit values A and B, where the bold bits indicate bits that have the same value in A and B – of the differing bits (bits 2, 3, 6 and 8) bits 3 and 6 were randomly selected and changed.

Parents	A	$\mathbf{x_1}$	a_2	a_3	$\mathbf{x_4}$	$\mathbf{x_5}$	a_6	$\mathbf{x_7}$	a_8
	B	$\mathbf{x_1}$	b_2	b_3	$\mathbf{x_4}$	$\mathbf{x_5}$	b_6	$\mathbf{x_7}$	b_8
Offspring	A^*	$\mathbf{x_1}$	a_2	b_3	$\mathbf{x_4}$	$\mathbf{x_5}$	b_6	$\mathbf{x_7}$	a_8
	B^*	$\mathbf{x_1}$	b_2	a_3	$\mathbf{x_4}$	$\mathbf{x_5}$	a_6	$\mathbf{x_7}$	b_8

Table 4.3: Half Uniform Crossover

A 1-point crossover exhibits "positional bias" – interacting bits relatively far apart are more likely to be disrupted than bits that are close together as an end of the parents is always exchanged [Caruana and Schaffer, 1988]. HUX and 2-point crossover do not suffer from this bias – HUX randomly selecting the bits to exchange and 2-point crossover is able to preserve both ends of each parent. With 2-point crossover, segments of the candidate solutions are preserved. HUX is the most disruptive of these crossovers, changing bits throughout the gene. If individual bits in the encoding provide information regarding the value encoded (e.g. in a standard binary encoding each bit contributes a specific amount to the value) then preserving segments of the values may help to find a solution by allowing exploration of a neighbourhood of similar values.

Under 1- and 2-point crossovers, if a given bit y_i is being exchanged then it is probable that neighbouring bits (y_{i-1}, y_{i+1}) will also be exchanged – as the crossovers exchange segments of consecutive bits. Changes in bit order will therefore affect 1- and 2-point crossover (given an 8-bit value, if bits 2 and 7 are exchanged 1/2/3 and 6/7/8 are no longer consecutive but 1/7/3 and 6/2/8 are). It has been proposed that the combination of partial solutions contributes to the power of EAs. This is termed the "Building block hypothesis" [Goldberg, 1989, pp. 41–45]. Using HUX, the only building blocks are bits which are common across parents, other segments of the data being disrupted.

Additional genes can be inserted in the genotype, separating segments with related bits. These additional genes are not expressed (i.e. do not affect the phenotype) but make it more likely that crossover will exchange complete related segments. This is analogous to the presence of *introns* in DNA. If **x** and **y** are vectors of related bits, rather than using [**xy**] as the genotype we can extend it by inserting additional genes **a** and **b** (i.e. using [**xayb**] as the genotype). Then, if a 2-point crossover extends from any bit in **a** to any bit in **b**, the complete vectors related to **x** and **y** will be exchanged. This can allow an EA to perform well on separable problems i.e. problems $f(x, y)$ that can be solved by separately finding y^*, the best y for any x, and x^* the best x for any y, and then combining the solutions. As HUX exchanges randomly selected individual bits from the parent solutions, inserting additional genes will, on average, have no effect on performance.

The different crossovers exhibit different degrees of sensitivity to the ordering of the bits in the encoding:

- For 1-point crossover, reversing the column order will give the same performance (2 equivalent codes);
- For 2-point crossover, both reversal and a shift of the columns will give the same performance ($2n$ equivalent codes);
- HUX is insensitive to column-order ($n!$ equivalent codes).

i.e. selecting one of the best codes for 2-point crossover is $\frac{(n-1)!}{2}$ times more difficult than selecting a best code for HUX. There is therefore a balance between the additional features available using position sensitive crossovers and the additional difficulty in finding a suitable encoding.

4.2.2 CHC

The CHC algorithm [Eshelman, 1991] is related to Genetic Algorithms, but uses uniform selection, in which all candidate solutions are used as parents, and HUX (as above) to evolve a population of candidate solutions, excluding parents that are too similar (incest prevention). CHC is *elitist* [De Jong, 1975, Chapter 4], keeping the best members of the combined population of candidate solutions and their children, and each generation is therefore guaranteed to contain a solution at least as good as the best in the previous generation.

The CHC algorithm is based on a population of candidate solutions. The candidate solutions are randomly paired off (uniform selection) and each pair of possible solutions is then compared. If the number of differing bits in the parents is large enough, then half these bits are exchanged to produce a pair of children using *Half*

Uniform Crossover (HUX) otherwise they are judged to be too similar to effectively explore the search space and produce no children. From the combined population of parents and children, the solutions that give the best values for the fitness function are adopted as the new population of candidate solutions. If the entire population is too similar to produce any children for several generations, then a new population is generated by mutating the current best candidate solution (referred to by Eshelman as *Cataclysmic Mutation*). We note that the number of children produced each generation is not fixed.

The CHC algorithm preserves population diversity by only combining suitably different parents, using the disruptive HUX crossover and applying Cataclysmic Mutation when population diversity is low.

4.3 A Comparison of Steepest Descent, SHC, the Simple GA and CHC

Reversing or complementing the bits has no effect on any of the algorithms. The performance of Steepest Descent, SHC, CHC and GAs using HUX are independent of the order of the bits representing a candidate solution – Steepest Descent selecting the "best" bit to mutate independent of its position, SHC, CHC and a GA using HUX randomly selecting the bits to mutate. However, the performance of a GA using 1- or 2-point crossover (e.g. the Simple GA) is affected by the bit order (as these crossovers preserve sequences of bits, the overall sequence is important). Shifting/rotating bits (e.g. changing the sequence $y_1, y_2, y_3, \ldots, y_n$ to $y_3, \ldots, y_n, y_1, y_2$) will affect 1-point crossover, and hence the Simple GA, as the ends of the sequence determine the elements that are exchanged.

Whereas Steepest Descent changes a single bit each generation, tracing a path through neighbouring values, SHC, CHC and the simple GA allow *multiple* bits to be changed in a single generation. These algorithms can therefore *explore* the parameter space rather than simply *exploiting* local features.

All four algorithms are computationally simple, the most complex functions involved being random number generation and sorting (for rank based selection). The CPU required to calculate the objective function is therefore typically much greater than that required for processing the algorithm itself. The time required to find a solution is then dominated by the number of times the fitness function needs to be evaluated.

- For Steepest Descent, all neighbours of the current value are tested each generation. For an n-bit representation, the expected number of function evaluations, E_{SD} is

$$E_{\text{SD}} = \text{E}(g_{\text{SD}}) \times n \tag{4.2}$$

  where $E(g_{\text{SD}})$ is the expected number of generations required to find a solution using Steepest Descent.

- For SHC, one value is tested each generation. Hence, the expected number of function evaluations, E_{SHC} is

$$E_{\text{SHC}} = \text{E}(g_{\text{SHC}}) \tag{4.3}$$

  where $E(g_{\text{SHC}})$ is the expected number of generations required to find a solution using SHC.

- For the Simple GA the complete population is replaced each generation and each member of the new population must be evaluated. Hence, with a population of m_{GA} solutions, the expected number of function evaluations, E_{GA} is

$$E_{\text{GA}} = \text{E}(g_{\text{GA}}) \times m_{\text{GA}} \tag{4.4}$$

  where $E(g_{\text{GA}})$ is the expected number of generations required to find a solution using the GA.

- For CHC with a population of m_{CHC} candidate solutions, up to m_{CHC} values can be evaluated each generation, the precise number of evaluations depending upon the number of children produced. CHC also works on smaller populations than the Simple GA. With a total population of m_{CHC} solutions, the expected number of function evaluations, E_{CHC} is bounded such that

$$E_{\text{CHC}} \leq \text{E}(g_{\text{CHC}}) \times m_{\text{CHC}} \tag{4.5}$$

  where $E(g_{\text{CHC}})$ is the expected number of generations required to find a solution using CHC.

The actual performance of each algorithm on a problem therefore depends upon both the algorithm parameters (e.g. population size) and the number of generations that the algorithm will require to find a solution.

4.4 Bit-wise Codings

An n-bit bit-wise encoding is a one-to-one mapping from a subrange of the integer values to vectors of n ones and zeroes (bits) – and can encode 2^n values. The problem space is then $\mathcal{P} = \{0, \ldots, 2^n - 1\}$, with these candidate solutions represented in the search space $\mathcal{S} = \{0, 1\}^n$. Given an integer candidate solution $x \in \mathcal{P}$ we will denote its representation in the search space as $\mathbf{x} = [x_{n-1}, \ldots, x_0] = \Phi(x)$, $\mathbf{x} \in \mathcal{S}$.

No Free Lunch theorems [Wolpert and Macready, 1997] establish that improvements in average search performance observed over one set of problems *must* be offset by reduced average search performance over some other problems i.e. that across all functions all algorithms will perform equivalently. The corollary of this being that there does not exist a single "black box" optimisation *algorithm* which will be effective in solving *all* optimisation problems. We are, however, looking at a specific piece-wise continuous problem – assuming continuous pitch trajectories within notes, the piece-wise continuous estimate of this trajectory implies a piece-wise continuous fitness function across the parameter space. We are therefore interested in algorithms which may perform well over the piece-wise continuous subclass of problems – at the expense of poor performance for some problems outside of this class. Considering bit-wise optimisation techniques, we can modify the overall function, $f\Theta(\cdot)$, evaluated by the algorithm by selecting an alternative coding i.e. choosing a different Θ and Φ.

We previously noted (Section 4.3) that the algorithms described above behave the same whether they flip bits $0 \to 1$ or $1 \to 0$ and that they are unaffected by certain changes in the bit-order – hence, for any code there is another code with identical performance (e.g. one in which all the bits are complemented) i.e. there is no single "best" *code* for a problem using any of the algorithms. Each code can, however, be seen as an example of a set of codes which will provide the same performance with an algorithm.

The *Standard Binary Code* $\mathbf{b} = [b_{n-1}, \ldots, b_0]$ allocates a power of 2 to each bit such that:

$$x = \sum_{i=0}^{n-1} b_i 2^i \quad . \tag{4.6}$$

Each bit, b_i, has an associated value with the left-most bit, b_{n-1} being largest (the most significant bit, msb) and contributing either 2^{n-1} or 0 to x and the right-most bit, b_0, smallest (least significant bit, lsb) contributing 1 or 0. As Standard Binary is the base-2 representation of the integers, it is a natural choice for arithmetic. The n bit Standard Binary code consists of the codewords for the $n-1$ bit code prepended by 0 followed by the same $n-1$ bit codewords prepended by 1.

For multidimensional data, individual values are encoded and combined into a single bit string. Two simple options are:

- concatenation – separately encoding each dimension and then concatenating the outputs;
- interleaving – separately encoding each dimension and then interleaving the bits (e.g. for Standard Binary high bits first, followed by successive bits in order).

Genes that are not expressed may be inserted between elements to increase the chance of building blocks being preserved (Section 4.2.1). For mutation based search, the choice between concatenation and interleaving (or a more complex scheme) has no effect on the output. EAs using crossovers that preserve sequences of bits (e.g. single / double point crossover) will be affected, whilst those that randomly select the updated bits (e.g. HUX) will not be affected. In the current work, we only consider concatenation of codewords to produce the representation – Steepest Descent, SHC and CHC being unaffected by either padding or interleaving.

Traditionally, the Standard Binary Code and the Binary Reflected Gray Code (BRGC) [Gray, 1953] have been used in optimisation problems. There have been many comparisons of the performance of the Standard Binary Code and the BRGC. For example, Mathias and Whitley [1994] compared the performance of the Standard Binary coding and BRGC over a set of standard test problems (Tables 4.8 and 4.9, below). The BRGC was found to perform better on this standard test set, but it was noted that there was no reason to assume better performance on an arbitrary function. Chakraborty and Janikow [2003] performed a theory-based numerical analysis of the performance of Standard Binary and BRGC using a Markov model to simulate Genetic Algorithms and a stochastic hillclimber, and found that, for small numbers of bits, there was only a small difference between the two representations over all possible functions. Rothlauf [2006, p117–140] compared the performance of Standard Binary and BRGC in selectorecombinative GAs and found that the overall performance of Standard Binary across all one-max problems (i.e. using the fitness function $f(x) = x$) was better. However, it was also noted that BRGC was the best choice for mutation based GAs.

Standard Binary and BRGC are, however, only two of the many possible binary encodings. We next consider some properties of binary encodings which may be desirable in optimisation problems, and introduce specific encodings related to those properties.

4.4.1 Hamming Distance and Locality

We define the *Hamming Distance* between two binary encoded vectors $\mathbf{x}$ and $\mathbf{y}$ to be the number of bits that differ:

$$D_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{n-1} |x_i - y_i| \ . \tag{4.7}$$

The values y such that $D_{\mathrm{H}}(\Phi(x), \Phi(y)) = 1$ are termed the *Hamming Neighbours* of x under the encoding Φ.

Binary encodings transform the search space, giving each value n Hamming neighbours as opposed to the 2 neighbours of the integer representation. In order to preserve local search properties from the integer problem it is necessary for small steps in the integer domain to correspond to small steps in the binary representation – this is termed *locality preservation* [Grajdeanu and De Jong, 2004]. Rothlauf [2006, p. 77] quantifies the locality preservation between two codings as the *Locality*, d_{m}. For binary encoding of integers, Locality, d_m, is defined in terms of the Hamming distance and a metric on the integers. A circular metric on the integers is used with the distance between two integers, x and y, being given by $|x - y| \mod 2^n$. For $x \neq y$ this distance has a minimum value of 1 when $y = (x \oplus 1)$, $\oplus$ indicating addition modulo 2^n. This metric differentiates between cyclic encodings (in which the values $2^n - 1$ and 0 are Hamming neighbours) and non-cyclic encodings. The locality, as defined by Rothlauf [2004] is given by:

$$d_{\mathrm{m}} = \sum_{x=0}^{2^n-1} |D_{\mathrm{H}}(x, (x \oplus 1)) - 1| \ . \tag{4.8}$$

A small d_{m} indicates *more* locality preservation. We note that $D_{\mathrm{H}}(x, (x \oplus 1)) \geq 1$ and the $|\cdot|$ is unnecessary in the case of Hamming distances. The locality indicates the number of bits, n_{b}, that need to change in cycling through all 2^n integer values, $n_{\mathrm{b}} = d_{\mathrm{m}} + 2^n$. For an encoding that preserves locality, all neighbouring integers should have a low distance between them. If each neighbouring pair of integers only differs in a single bit, the locality will be zero indicating perfect locality preservation – all neighbouring integers also being neighbours in the binary encoding.

In 1953, Frank Gray introduced the *Binary Reflected Gray Code* (BRGC) [Gray, 1953] – the term “Gray Code” subsequently being adopted for any code in which all neighbouring integer values are Hamming neighbours. The locality for the BRGC is therefore zero ($d_{\mathrm{m}} = 0$). As there is only one bit different between the n-bit representations for 0 and for $2^n - 1$, the BRGC is a *cyclic* Gray code. The n bit BRGC consists of the codewords for the $n - 1$ bit code prepended by 0 followed by the same $n - 1$ bit codewords but in reverse order (hence the "Reflected" part of Binary Reflected Gray Code) and prepended by 1. There are many other Gray codes [Savage, 1997], although the BRGC is commonly referred to as "the" Gray code. All cyclic Gray codes have locality zero ($d_{\mathrm{m}} = 0$) as all pairs of neighbouring integer values (x and $x \oplus 1$) have only one bit different.

In order to provide a more direct interpretation of the locality, we introduce the *relative locality* $\frac{d_{\mathrm{m}}}{2^n}$. Rather than showing the total number of bits changed across the set of integer values, this measures the average number of additional bits over 1 bit that need to be changed to reach a neighbouring integer.

It has been shown [Whitley, 1999] that reflected Gray codes induce more optima in "worst case" functions (where every other point is a minima) and hence, according to the No Free Lunch theorem (p.72), fewer optima in the remaining functions – and with fewer optima, reflected Gray codes are, on average, less likely to result in an algorithm becoming trapped at a local optimum.

In the Standard Binary code (p.72), neighbouring integer values can differ in more than one bit – e.g. between each odd value, $2i - 1$, and the next even value, $2i$, (e.g. three bits differ between the values 3 [0 1 1] and 4 [1 0 0]), but not between any even value, $2i$, and the next odd value, $2i + 1$ (the Hamming distance is 1 as only bit b_0 changes). These values in which a small move in the problem space requires a larger move in the search space are termed "Hamming cliffs". Although a 2 bit difference between neighbouring values seems small, it can make a problem much more difficult to solve – for an m bit Stochastic Hill Climber with a mutation probability of $\frac{1}{m}$ for each bit, the probability of testing a neighbouring value is $\frac{1}{m}$ for a 1-bit difference and $\frac{1}{m^2}$ for a 2-bit difference, and many points away from the neighbouring value are likely to be tested before the neighbouring value itself. The lack of Hamming cliffs has been regarded as enhancing the performance of Gray codes in optimisation [Srinivas and Patnaik, 1994] – this property is equivalent to a locality, d_{m}, of zero.

A *Maximum Distance Code* [Robinson and Cohn, 1981] has the largest possible Hamming Distance between neighbouring values – alternately a distance of n and $n - 1$ bits. It therefore has minimal locality preservation and maximises the locality, d_{m}. The locality of an n-bit Maximum Distance Code is $d_{\mathrm{m}} = 2^{n-1}(2n - 3)$.

The n-bit Maximum Distance Code alternates codewords from the $(n-1)$-bit BRGC, expanded to n bits by prepending a zero, with the same codewords but inverted. Denoting the Maximum Distance encoding function as Φ_{M} and the ith bit of the encoding as $\Phi_{\mathrm{M}}(\cdot)_i$:

$$\begin{aligned} \text{for even } x: \quad & \Phi_{\mathrm{M}}(x)_{n-1} = 0 \\ & \Phi_{\mathrm{M}}(x)_i = \Phi_{\mathrm{G}}\left(\frac{x}{2}\right)_i \quad i = 0, \ldots, n-2 \\ \text{for odd } x: \quad & \Phi_{\mathrm{M}}(x)_i = 1 - \Phi_{\mathrm{M}}(x-1)_i \quad i = 0, \ldots, n-1 \end{aligned} \tag{4.9}$$

where Φ_{G} is BRGC encoding function. If the small distance between codewords, and the lack of Hamming cliffs, allow Gray codes to perform *well*, then the Maximum Distance Code should perform *poorly*, offering an appropriate counter-example to use in comparisons.

4.4.2 Transition Counts and Balancedness

If the codewords for the values from 0 to $2^n - 1$ are considered in turn, individual bits sometimes change and sometimes remain the same. The *Transition Count* for a bit is the number of times that that particular bit changes value in traversing the values of x. The Transition Count for bit i is given by:

$$t_i = \sum_{x=0}^{2^n-1} |x_i - (x \oplus 1)_i| \tag{4.10}$$

including the transition from $2^n - 1$ to 0 if $(2^n - 1)_i \neq (0)_i$. The Transition Count indicates the number of distinct regions in the search space identified by the bit.

For any cyclic Gray code, a single bit changes between each pair of successive values, giving a *Total Transition Count*, $\sum_i t_i$, of 2^n – including a transition between $2^n - 1$ and zero. This is the minimum possible Total Transition Count, as at least one bit must vary between neighbouring values. If the Total Transition Count across all n bits is greater than 2^n then there must be values where integer neighbours differ in more than one bit i.e. Hamming cliffs.

A *balanced* binary code has the same transition count for each bit. Neither the Standard Binary Code nor the BRGC is balanced – in Standard Binary, bit i has 2^{n-i} transitions, for the BRGC bit $n-1$ has 2 transitions and subsequent bits have 2^{n-i-1} transitions as a result of the reflected nature of the code. We can introduce the (un-)*Balancedness* for a code:

$$b = \frac{1}{n}\left(\sum_{i=0}^{n-1} t_i^2\right) - \left(\frac{1}{n}\sum_{i=0}^{n-1} t_i\right)^2 \quad . \tag{4.11}$$

This indicates how widely the Transition Counts for the bits are spread, taking a value of 0 when all bits have the same Transition Count – i.e. when the encoding is perfectly balanced.

A *Balanced Gray Code* [Robinson and Cohn, 1981, Bhat and Savage, 1996] gives each bit as similar a Transition Count as possible whilst satisfying the condition for a Gray code – i.e. that neighbouring values differing in a single bit. The method given in Robinson and Cohn [1981] can produce various balanced codes, as there are choices in how subsequences used to build the code are produced.

4.4.3 Hamming Weight

Under a binary encoding, the *Hamming Weight*, $W_{\mathrm{H}}(x)$, is the number of bits set in the encoded value:

$$W_{\mathrm{H}}(x) = \sum_{i=0}^{n-1} x_i \ . \tag{4.12}$$

A *monotone* binary code orders the codewords according to their Hamming weights.

The *Monotone Gray Code* [Savage and Winkler, 1995] relaxes the constraints typically imposed on Gray codes by allowing $D_{\mathrm{H}}(2^n - 1, 0) \neq 1$ (i.e. it is non-cyclic) and is built in such a way as to be approximately monotone. It is *not* strictly monotone, as neighbouring Gray codes cannot have the same weight as they differ by a single bit, but is formed from two interleaved monotone subsequences of even and odd weights. Under a Monotone Gray Code, for even n, $W_{\mathrm{H}}(2^n - 1) = n - 1$ and, for odd n, $W_{\mathrm{H}}(2^n - 1) = n$.

We note that if the Hamming weight of a Monotone Gray codeword is known, then the associated value is within a specific range. The number of items with Hamming weight w is $\kappa_w = \begin{pmatrix} w \\ n \end{pmatrix}$ and the minimum value with that weight is $\psi(w)$,

$$\psi(w) = \begin{cases} 0 \text{ for } w = 0 \\ 1 \text{ for } w = 1 \\ \psi_{w-2} + 2\kappa_{w-2} \text{ otherwise} \ . \end{cases} \tag{4.13}$$

4.5 A New Analysis of Binary Codings

The Locality and Balancedness criteria, described above, defined the BRGC, Maximum Distance and Balanced Gray codes. Additionally, we introduced the Standard Binary code and Monotone Gray codes. We introduce a new analysis of these codes, examining their properties to gain insights into their differences and their similarities.

We examine the properties of example codings, showing: full 4-bit codes; a graphical representation of 5-bit codes; the distribution of the properties for the 6-bit codes; and the variation of the properties with the number of bits.

4.5.1 Sample 4-bit Encodings

Value	Standard Binary	BRGC	Maximum Distance	Balanced Gray	Monotone Gray
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	1 1 1 1	1 0 0 0	1 0 0 0
2	0 0 1 0	0 0 1 1	0 0 0 1	1 1 0 0	1 1 0 0
3	0 0 1 1	0 0 1 0	1 1 1 0	1 1 0 1	0 1 0 0
4	0 1 0 0	0 1 1 0	0 0 1 1	1 1 1 1	0 1 1 0
5	0 1 0 1	0 1 1 1	1 1 0 0	1 1 1 0	0 0 1 0
6	0 1 1 0	0 1 0 1	0 0 1 0	1 0 1 0	0 0 1 1
7	0 1 1 1	0 1 0 0	1 1 0 1	0 0 1 0	0 0 0 1
8	1 0 0 0	1 1 0 0	0 1 1 0	0 1 1 0	0 1 0 1
9	1 0 0 1	1 1 0 1	1 0 0 1	0 1 0 0	1 1 0 1
10	1 0 1 0	1 1 1 1	0 1 1 1	0 1 0 1	1 0 0 1
11	1 0 1 1	1 1 1 0	1 0 0 0	0 1 1 1	1 0 1 1
12	1 1 0 0	1 0 1 0	0 1 0 1	0 0 1 1	1 0 1 0
13	1 1 0 1	1 0 1 1	1 0 0 1	1 0 1 1	1 1 1 0
14	1 1 1 0	1 0 0 1	0 1 0 0	1 0 0 1	1 1 1 1
15	1 1 1 1	1 0 0 0	1 0 1 1	0 0 0 1	0 1 1 1

Table 4.4: Example 4-bit encodings

To visualise the codes we are discussing, Table 4.4 shows small (4-bit) examples.

The Standard Binary and BRGC both have a "tree like" block structure: for the Standard Binary code, the first bit has two transitions and each subsequent bit doubles the transition count (the bits have transition counts of 2, 4, 8 and 16 in the example); for the BRGC the first two bits have 2 transitions, a similar doubling process to Standard Binary then being observed (the bits having 2, 2, 4, 8 transitions). This is a product of the algorithms used to generate the codes (p.72 and p.75).

Examining the codes further, neighbouring Gray codewords vary in only one bit, whereas the other codes exhibit Hamming cliffs. Pairs of consecutive Maximum Distance codewords have a Hamming distance of 3 or 4. Each bit of the Balanced Gray Code has a Transition Count of 4 (2 from 0 to 1 and 2 from 1 to 0). The Hamming weights of the Monotone Gray codewords interleave increasing sequences of odd and

Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hamming Weight (even)	0	-	2	-	2	-	2	-	2	-	2	-	2	-	4	-
Hamming Weight (odd)	-	1	-	1	-	1	-	1	-	3	-	3	-	3	-	3

Table 4.5: Hamming weights for Monotone Gray Code

even weights (Table 4.5) in agreement with Equation 4.13, reaching a weight of 4 for value 14.

4.5.2 Plotting 5-bit Codings

We now introduce a new visualisation to help compare the characteristics of the various codings.

In figure 4.2, 5-bit values are shown on a circle and lines join them to their Hamming neighbours (respectively under the Standard Binary, BRGC, Balanced Gray and Monotone Gray codings). This provides a partial representation of the codes – the identification of *which* bit relates to each pair of neighbours and which end has the bit set is not indicated. As the connections between values are not labelled as to which bit they relate to, all permutations of the columns of a bit encoding will have the same circular representation. Similarly, as the graphs do not indicate which end of the connection has the bit set, encodings formed by negating any bits will also have the same representation. As noted in section 4.3 (p.70), negating and permuting the columns does not affect the performance of Steepest Descent, SHC or CHC – these plots therefore represent *all* the information relevant to these optimisation techniques.

If consecutive values are not Hamming neighbours (i.e. they differ in more than one bit) then they are not directly linked in this representation and the resulting Hamming cliffs are indicated by gaps between the values on the circle. The Gray codes (BRGC, Balanced and Monotone) have no Hamming cliffs, apart from between 0 and $2^n - 1$ for the non-cyclic Monotone Gray code (between 0 and 31 in Figure 4.2).

The simple algorithmic nature and the strong structure of the Standard Binary and BRGC codes can be seen to produce a regular pattern linking values in specific regions, the Balanced and Monotone codes are generated recursively, extending low-order codes to generate those of higher orders producing a more “chaotic” map with no distinct regions of linked values.

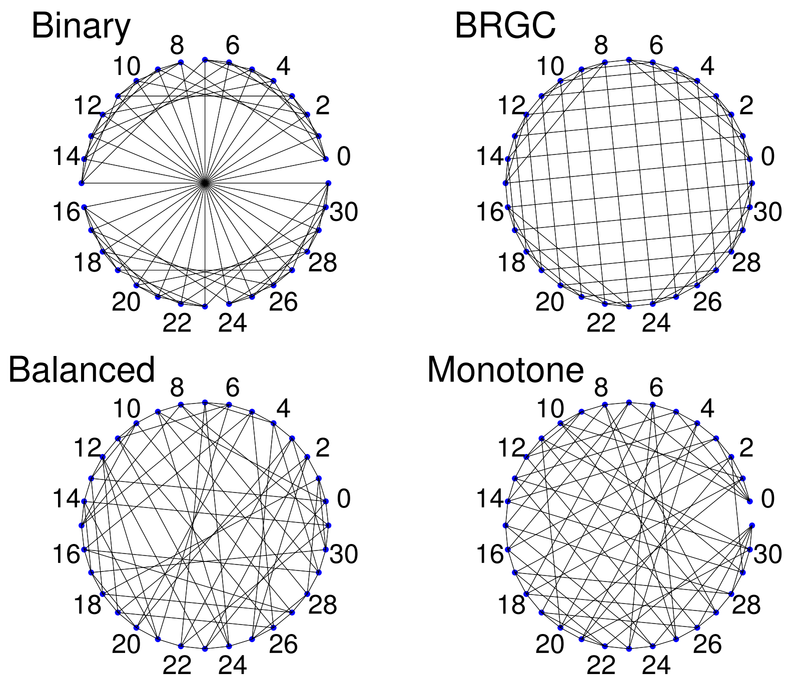


Figure 4.2: Hamming Neighbours for 5-bit Codes. Values are dots on the circle, even values being labelled.

4.5.3 Distribution of 6-bit Property Values

Random binary encodings can be generated by selecting a random permutation of the set of 2^n codewords. We examined the balancedness and locality for a sample of random binary encodings, in order to gain insights into their range and distribution – the specific properties of individual encodings can then be considered relative to this distribution.

Property	Min	Mean	Max
Locality, d_m	78.00	131.05	184.00
Balancedness, b	0.00	16.25	152.67

Table 4.6: Summary Balancedness and Locality statistics from the 100,000,000 example 6-bit Random Binary Codes

We generated 100,000,000 6-bit Random Binary Codes using permutations of the 2^6 codewords and examined the Balancedness and Locality of the codes. On average, we would expect 3 bits to differ in two randomly selected codewords giving an expected value for d_m of 128. The mean observed locality, d_m of 131.05 (Table 4.6) compares well

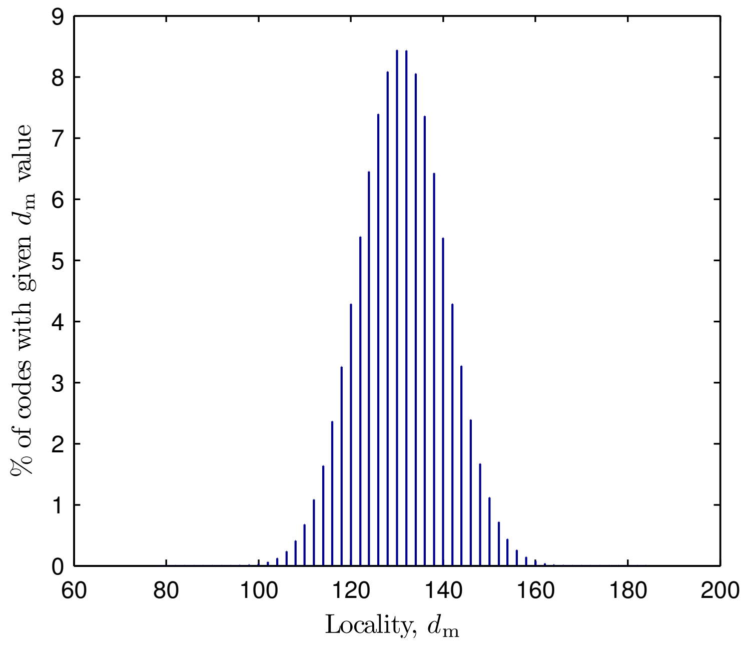


Figure 4.3: Histograms indicating the proportion of 100,000,000 example 6-bit Random Binary Codes with specified locality, d_m

with this – the relative locality (p.75) being 2.0477 indicating an average of 3.0477 bits differing between neighbouring integers. We note that the generated locality values included no Gray codes ($d_\mathrm{m} = 0$) and no Maximum Distance codes ($d_\mathrm{m} = 2^{n-1}(2n-3) = 288$) indicating that these codes are relatively rare – there are $2^n!$ possible n-bit codes and the Gray codes are a subset of the $2^n n$ sequences in which one bit changes between values (many of the $2^n n$ sequences being invalid as they repeat codewords). The locality distribution (Figure 4.3) is symmetrical and shows that locality takes discrete values as there are spaces in the distribution.

Of the random codes, $12,000$ were perfectly balanced, with the minimum observed "Balancedness", b, of 0 (Table 4.6) – although these codes were balanced they were not Gray codes. The skewed distribution (Figure 4.4) indicates that some encodings have b much larger than the mean (16.25) – as shown by the maximum observed b of 152.67 (Table 4.6).

4.5.4 Example Individual 6-bit Codings

Comparing these properties with those for the 6-bit versions of the codings under consideration (Table 4.7), the three Gray codes exhibit low locality, d_m, in accordance with neighbouring integer values being Hamming neighbours under the binary repre-

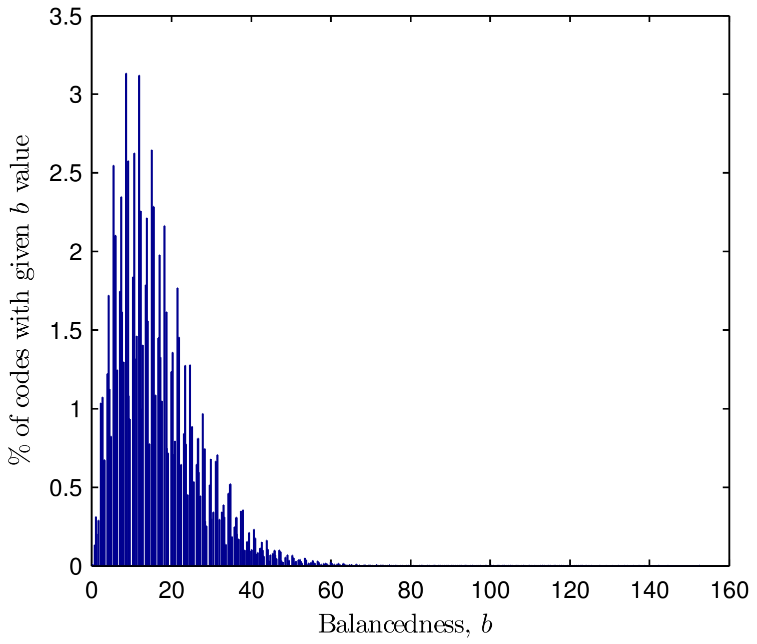


Figure 4.4: Histograms indicating the proportion of 100,000,000 example 6-bit Random Binary Codes with specified balancedness, b

Code	b	d_{m}
Standard Binary	469.00	62
BRGC	114.22	0
Maximum Distance	28.89	288
Balanced	0.89	0
Monotone	7.56	4

Table 4.7: (un)Balancedness, b, and Locality, d_{m}, for 6-bit encodings

sentations. As the BRGC and Balanced Gray code are cyclic they both have $d_{\mathrm{m}} = 0$. The Monotone Gray Code has $d_{\mathrm{m}} > 0$ as it is non-cyclic ($d_{\mathrm{m}} = n - 2$ or $d_{\mathrm{m}} = n - 1$ depending upon whether n is even or odd). The Standard Binary Code is particularly unbalanced as each successive bit has twice the transitions of the previous one – the balancedness, b, is 469.00 whereas the maximum balancedness in the 100 million sample codes was 152.67. The BRGC is more balanced as the first two bits have two transitions each, subsequent bits doubling the previous number of transitions.

4.5.5 Variation of Properties with Number of Bits

We examined the variation in the properties of the encodings, as the number of bits in the coding increases from 1 to 15. The values have been scaled for comparison: we examined the relative locality (the average number of bits more than one to move to a neighbouring integer); and the Balancedness is in units squared and is scaled by 2^{2n}.

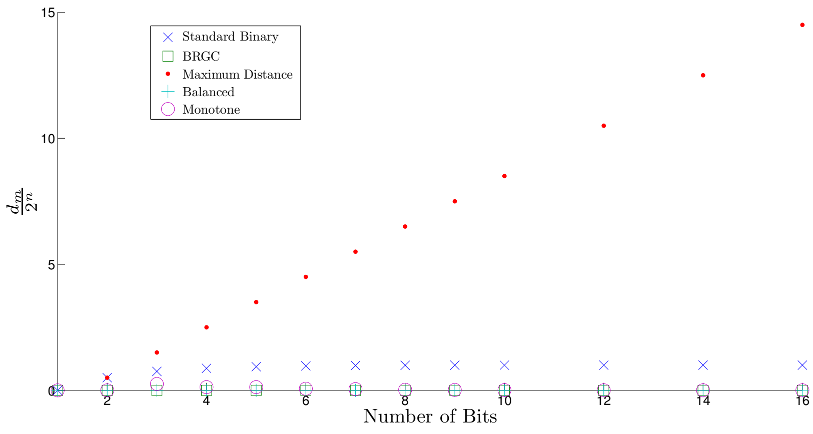


Figure 4.5: Variation of relative locality with number of bits and code (values scaled as shown)

We examined the relationship between the relative locality and the number of bits in the code (Figure 4.5). The relative locality measure, $\frac{d_{\rm m}}{2^n}$, of the Maximum Distance code increases with the number of bits, showing an increasing Hamming distance between neighbouring values. This is a feature of the design of the Maximum Distance Code (each integer is Hamming distances of n and $(n-1)$ from its neighbours). The Gray codes had a locality of 0, except for the non-cyclic Monotone Gray code – however, as only the change from $2^n - 1$ to 0 changes more than one bit, the relative locality for the Monotone Gray code is still very low (either $\frac{n-2}{2^n}$ or $\frac{n-1}{2^n}$ depending upon whether n is even or odd).

A low locality indicates that modifying a single bit in the search space can move to a neighbouring value in the problem space, and that exploitation of immediate integer neighbours is possible.

The relationship between the relative balancedness and the number of bits in the code (Figure 4.6) shows that for all codes, the relative balancedness, $\frac{b}{2^{2n}}$, decreases as the number of bits increases above 4 bits – i.e. codes become more balanced. The Standard Binary and BRGC are significantly less balanced than the other codes.

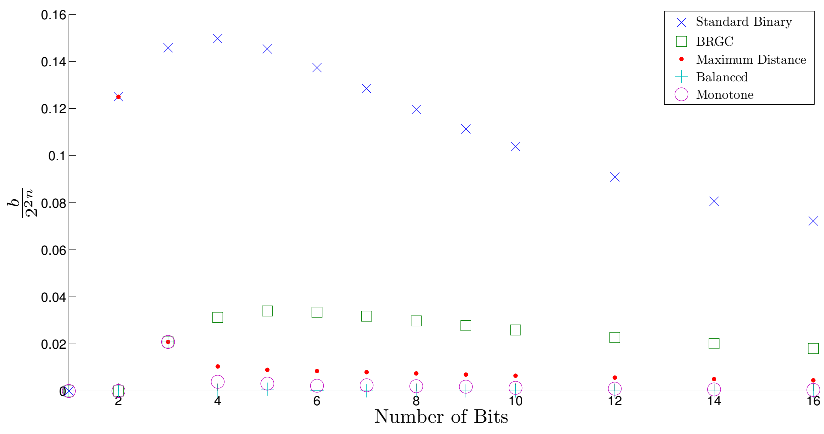


Figure 4.6: Variation of relative balancedness with number of bits and code (values scaled as shown)

Low values for balancedness imply that each bit changes a similar number of times – each bit subdivides the search space into a similar number of segments rather than some bits being "easy" to set (e.g. in the Standard Binary / BRGC the high-bit indicates which half of the search space a value is in) and some "difficult" (BRGC "low" bit is ± 1). Each bit contributes similarly to exploration of the problem space.

If the balancedness and locality were sufficient to quantify the ability of the code to explore/exploit the search space then the Monotone and Balanced Gray codes should exhibit good performance. If exploitation is important, and quantified by locality, then the Maximum Distance code will perform poorly and the Gray codes well. If uniform exploration across bits is important, and quantified by balancedness, then the Standard Binary code will perform poorly, the Balanced and Monotone Gray codes well.

4.6 Algorithm and Coding Performance Analysis

We examined the performance of the optimisation algorithms and binary encodings over a set of test problems, extending the work of Mathias and Whitley [1994]. Mathias and Whitley compared the performance of Steepest Descent, a Stochastic Hill-climber (SHC) and Eshelman's CHC algorithm [Eshelman, 1991] using the Standard Binary code and the Binary Reflected Gray code (BRGC) on a set of standard test functions. We repeated these experiments, extending them to include: a Simple GA; and Monotone and Balanced Gray codes.

The test functions used are the same as used by Mathias and Whitley (Tables 4.8 and 4.9). Test functions F1–F5 are those defined by De Jong [1975, Appendix A], the remaining as given by Whitley et al. [1995]. Many of these functions have been used throughout the literature on Genetic Algorithms (e.g. [Hinterding et al., 1995, Karaboğa and Őkdem, 2004]) and are considered in some detail by Whitley et al. [1996])

4.6.1 Method

In line with the method used by Mathias and Whitley [1994], for each combination of coding and optimisation algorithm, 100 tests were run. Each started at a random position / population and continued until either the optimum solution was found or until a number of generations had passed without finding the solution. If the optimisation algorithm stayed at a local optimum for a number of generations, then the algorithm was restarted at a new random position.

A test harness was created in Matlab 7.2 to run the tests using the Matlab Distributed Computing Engine on an Apple XServe cluster with 7 worker nodes. The test harness was set up to save the state after each test to allow resumption if the batch was interrupted. The codes and algorithms to be used, the test functions to run and the number of tests to use were passed as parameters, and the parameters for each individual test function were set at values given in Mathias and Whitley. Procedures were created for each optimisation algorithm. These called individual procedures for the test functions passing population values decoded from the bit-wise representations.

Algorithm/Coding combinations were assessed based on: the number of tests that were solved for each problem; the number of times the algorithm had to be started to solve a test; and the average number of generations to find the test solution for successful starts. The best possible algorithm for this test would solve all the problems for all 100 tests with a single start – and in a low number of generations.

4.6.2 Results

The specific behaviour of the DeJong F4 noisy function (Table 4.8, p.86) is not defined – "noise" could either be applied once to the data (cached noisy data) or applied each time the function is evaluated ("online" noisy data, e.g. trying to match live sensor readings). In our version of the Dejong F4 function, each time the function was tested,

Function	**Notes**
De Jong F1 – "Sphere" $$\mathrm{F1}(\mathbf{x}) = \sum_{i=1}^{3} x_i^2 \quad (4.14)$$ $x_i = \frac{y_i - 512}{100}$, $0 \leq y_i < 2^{10} - 1$, $y \in \mathbb{I}$ Minimum zero at $(0, 0, 0)$	3-dimensional 30 bits required to represent a candidate solution. A parabola and an easy problem. It is a continuous, convex, unimodal, separable function.
De Jong F2 – Rösenbrock function $$\mathrm{F2}(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \quad (4.15)$$ $x_i = \frac{y_i - 2048}{1000}$, $0 \leq y_i < 2^{12}$, $y_i \in \mathbb{I}$. Minimum zero at $(1, 1)$	2-dimensional 24 bits required to represent a candidate solution. It is a continuous, non-convex, unimodal, non-linear function, sometimes referred to as the "Banana" function as the global optimum lies inside a long, parabolic valley. Finding the valley is usually straightforward, finding the optimum less so as the valley is narrow and flat.
De Jong F3 – Step Function $$\mathrm{F3}(\mathbf{x}) = \sum_{i=1}^{5} \lfloor x_i \rfloor \quad (4.16)$$ $x_i = \frac{y_i - 512}{100}$, $0 \leq y_i < 2^{10} - 1$, $y \in \mathbb{I}$ Minimum -30 at any point s.t $\forall i, x_i < -5$	5-dimensional 50 bits required to represent a candidate solution A step-function, it a series of flat plateaus from which no information regarding the direction to the optimum can be derived. All points on the plateaus are local optima causing local-search methods to fail. It is discontinuous, non-convex, unimodal, 5-dimensional, step function, piece-wise constant.
De Jong F4 – Noisy Data $$\mathrm{F4}(\mathbf{x}) = \sum_{i=1}^{30} i x_i^4 + g \quad (4.17)$$ $x_i = \frac{y_i - 128}{100}$, $0 \leq y_i < 2^8$, $y_i \in \mathbb{I}$. g is a zero-mean unit variance Gaussian noise generator. Ignoring the noise, the minimum is zero at $(0, 0, \ldots, 0)$. Including the noise, the location of the minimum depends upon the particular noise values generated. De Jong defines the minimum to be zero at $(0, 0, \ldots, 0)$.	30-dimensional 240 bits required to represent a candidate solution This would be an easy function, but features a Gaussian noise generator meaning that the function value returned for a given point varies each time the function is called.

Table 4.8: De Jong test functions F1 to F4 [De Jong, 1975, Appendix A]

Function	Notes
De Jong F5 – Shekel "Fox Holes" $$\frac{1}{\mathrm{F5}(\mathbf{x})} = \frac{1}{500} + \sum_{j=1}^{25} \frac{1}{f_j(\mathbf{x})} \quad (4.18)$$ where $$f_j(\mathbf{x}) = j + \sum_{i=1}^{2} (x_i - a_{ij})^6 \quad (4.19)$$ $x_i = \frac{y_i - 65536}{1000}$, $0 \leq y_i < 2^{17}$, $y_i \in \mathbb{I}$. The overall minimum occurs at $(a_{11}, a_{21}) \equiv (-32, -32)$ with value $\approxeq 1$.	2-dimensional 34 bits required to represent a candidate solution The function is almost flat but with 25 holes of varying depth, it is a difficult function as search techniques can get stuck in one of the holes. It is continuous, non-convex, non-quadratic and multimodal, with 25 local minima. Minima occur at (a_{1j}, a_{2j}) with value $\approxeq j$. $$A = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \dots & 32 & 32 & 32 \end{pmatrix}$$ i.e. the sequence of values -32 -16 0 16 32 is repeated five times for the first row of A and each value in the sequence is repeated five times for the second row of A.
Rastrigin function $$\mathrm{F6}(\mathbf{x}) = 10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i) \quad (4.20)$$ $x_i = \frac{y_i - 512}{100}$, $0 \leq y_i < 2^{10} - 1$, $y \in \mathbb{I}$ The minimum is zero at $(0, 0, \dots, 0)$.	Used in 20 dimensional form ($n = 20$) 200 bits required to represent a candidate solution Continuous, non-convex, non-quadratic, multi-dimensional, multi-modal
Schwefel function $$\mathrm{F7}(\mathbf{x}) = -\sum_{i=1}^{n} x_i \sin(\sqrt{\lvert x_i \rvert}) \quad (4.21)$$ $x_i = \frac{y_i - 512}{100}$, $0 \leq y_i < 2^{10} - 1$, $y \in \mathbb{I}$	Used in 10 and 20 dimensional forms ($n = 10$ and $n = 20$). 100 / 200 bits required to represent a candidate solution. It is a multi-modal function with a second optimum far away from the global minimum.
Griewank function $$\mathrm{F8}(\mathbf{x}) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \quad (4.22)$$ $x_i = \frac{y_i - 512}{100}$, $0 \leq y_i < 2^{10} - 1$, $y \in \mathbb{I}$ The minimum is zero at $(0, 0, \dots, 0)$.	Again, used in both 10 and 20 dimensional forms ($n = 10$ and $n = 20$). 100 / 200 bits required to represent a candidate solution. The product term makes the function non-separable so it is not possible for a technique to find the best individual components and then combine them for the best overall answer.

Table 4.9: De Jong test function F5 [De Jong, 1975, Appendix A] and the Rastrigin, Schwefel and Griewank functions [Whitley et al., 1995]

	Solved	Mean Starts per Solution	Mean Generations per per Solution	Mean Evaluations per Solution
Binary	33.7%	162.49	13	504.02
BRGC	55.5%	106.92	37	8159.13
Balanced Gray	71.8%	78.84	73	23017.41
Monotone Gray	57.9%	113.06	61	12947.02
Max. Distance	33.1%	117.34	13	492.82

Table 4.10: Summary Statistics for Steepest Descent. Generations and Evaluations per Solution apply to start positions from which the solution was successfully found.

the noise was resampled – as for an online sampling rather than batch data. Hence, no "fixed" best point exists, and we want the algorithm to find the best point in the underlying noiseless function by sampling the noisy function. Under these conditions, we failed to find solutions with any of the algorithm/code combinations. For the other functions, results for SHC and CHC using Standard Binary and BRGC parallel those from the original paper. Detailed results are presented at the end of this chapter (pp.95–100), and a summary of the results is presented below.

Steepest Descent (Table 4.10) often found local minima and required restarting to solve problems. With Steepest Descent, the Balanced Gray code was most successful both in terms of number of problems solved (71.8%) and, on average, needing fewest restarts (78.84) to achieve those solutions. Although the mean number of generations per solution, and hence evaluations, are larger for the Balanced Gray code, these figures are affected by its success on the Schwefel problems. Excluding the Schwefel problems the Balanced Gray Code required an average of 56 generations to find the solution over the remaining problems.

Examining the detailed results (Table 4.16), the Balanced Gray code was the only code to solve all 200 Schwefel problems (Table 4.9, p.87) (100 for the 10 dimensional problem and 100 for 20 dimensions). None of the other codings solved any of the 20 dimensional Schwefel problems. The 20 dimensional problems required an average of 154 generations to find a solution, the 10 dimensional problems an average of 77 – the difficulty of the problem therefore appearing to scale linearly with the number of dimensions. For both the 10 and 20 dimensional problems, the Balanced Gray code took an average of approximately $\frac{3}{4}n$ generations to find the solution, where n is the number of bits in the problem (100 or 200). Bits in the start position were set to 0 or 1 according to a uniform random distribution. We would therefore expect half the bits to be set correctly at the start position. If the global optimum was reached

	Solved	Mean Starts per Solution	Mean Generations per per Solution	Mean Evaluations per Solution
Binary	17.0%	2.35	50,368	50,368
BRGC	61.5%	1.46	33,868	33,868
Balanced Gray	62.1%	1.45	32,668	32,668
Monotone Gray	46.3%	1.73	59,505	59,505

Table 4.11: Summary Statistics for SHC

by successively setting bits to the correct value, we would therefore expect the mean number of generations to be $\frac{1}{2}n$ – i.e it was actually necessary for the Balanced Gray to change an additional $\frac{1}{8}n$ bits, and then change them back, in order to find the solution. The Schwefel problem is a multi-modal function with a second optimum far away from the global minimum. We hypothesise that the balanced nature of the Balanced Gray code allowed the search to move away from the local optimum to find the global optimum, whilst the local optimum trapped the other codes preventing the algorithm from finding the global optimum.

However, the Balanced Gray code showed the worst performance on the De Jong F3 step function (Table 4.8, p.86), only solving 69% of the problems whilst the other encodings allowed almost all to be solved. Again, the nature of the codes can explain this performance difference – the Standard Binary, BRGC and Maximum Distance codes have bits ranging from most significant to least significant, subdividing the search space into smaller subsections and would be expected to set these bits sequentially to produce the Steepest Descent (the largest decrease occurring with the largest possible steps towards the minimum); and the Monotone Gray code can successively set bits to zero to move down to the minimum.

Although Steepest Descent required few generations to find the solution, a large number of fitness function evaluations occur, as all Hamming neighbours of the candidate solution are tested. In addition to the BRGC, Standard Binary code, and Balanced and Monotone Gray codes, we examined the performance of the Maximum Distance Code with Steepest Descent and found that its performance – with Hamming cliffs between neighbouring values – was very similar to that of the standard Binary code.

Performance of the Stochastic Hillclimber (SHC) (Table 4.11) was very similar using either the BRGC or the Balanced Gray code, both of which were better than the Monotone Gray code, whilst the Binary code gave poor performance. The Stochastic

Hillclimber solved six of the problems with a single start using BRGC, the Balanced Gray code solved all the tests for 5 problems, and 96% of tests on a sixth. Whereas the higher levels of exploration for the Balanced Gray code caused significant performance variations with Steepest Descent, SHC allowed both exploration and exploitation as the candidate solution could move anywhere in the whole problem space from the current candidate solution. A low number of starts were required whenever solutions were found – a single start can eventually find the solution but may take a large number of generations to reach the solution e.g. given the problem $y(x) = 0$ if $x = x_0$ and $y(x) = 1$ otherwise, we would expect an n-bit random start position to have $\frac{n}{2}$ bits correctly set and SHC with a mutation rate of $\frac{1}{n}$ would take an average of $n^{\frac{n}{2}}$ generations to find the solution. As SHC only evaluates a single point each generation, it runs quickly. However, all codes took more function evaluations than Steepest Descent, the Standard Binary and Balanced Gray codes solving fewer problems with SHC than Steepest Descent.

Examining the detailed results (Table 4.17), the BRGC performed well on the Rastrigin function (Table 4.9, p.87) whilst the Balanced Gray code performed poorly, and conversely the Balanced Gray performed well on the 20 dimensional Schwefel function whilst the BRGC performed poorly. Again, the explorative nature of the Balanced Gray code may be the cause – the Rastrigin function has a series of local minima as a result of the cosine function and whilst the Balanced Gray code is likely to switch between these local minima, the more exploitative nature of the BRGC (with its less significant bits) will allow it to follow valleys to the minimum. Interestingly, although Steepest Descent with Gray codes could find the solution for the Griewank problems, SHC failed.

The Standard Binary code performed poorly on the DeJong F1 “Sphere” problem (Table 4.8, p.86). This is a smooth surface, and the problem can be solved by moving to neighbouring integer values and the Hamming cliffs in the Standard Binary representation degrade performance. The Standard Binary code also performed poorly on the F5 “Shekel’s Foxholes” problem (Table 4.9, p.87). The solution to this is one of a set of 25 local minima centred in the search space and the geometry of the Standard Binary code makes it difficult to explore these local minima.

With a population of 100 candidate solutions, the performance of the Simple GA (Table 4.12) was similar to SHC for each code but generally required fewer restarts. Each generation of the Simple GA evaluates the entire new population, therefore requiring more fitness function evaluations than SHC (Table 4.12 showing approximately double the number of evaluations of Table 4.11), this increasing again with

	Solved	Mean Starts per Solution	Mean Generations per per Solution	Mean Evaluations per Solution
Binary	18.6%	2.15	1,270	127,000
BRGC	60.1%	1.33	1,139	113,900
Balanced	61.3%	1.30	1,247	124,700
Monotone Gray	48.8%	1.64	1,585	158,500

Table 4.12: Summary Statistics for Simple GA (population 100)

	Solved	Mean Starts per Solution	Mean Generations per per Solution	Mean Evaluations per Solution
Binary	33.2%	1.51	1,078	1,078,000
BRGC	60.3%	1.16	1,082	1,082,000
Balanced	61.4%	1.30	1,175	1,175,000
Monotone Gray	49.7%	1.61	1,419	1,419,000

Table 4.13: Summary Statistics for Simple GA (population 1000)

increasing population size. With the population increased to 1000 (Table 4.13), a further decrease in the number of restarts required was achieved, especially for the Binary code with an attendant increase in the number of evaluations from the larger population.

In general, the ability of CHC (Table 4.14) to solve problems is less affected by the specific code used – using the HUX crossover bits are randomly selected to change during recombination. However, the code used affects the number of generations until a solution is found. Excluding De Jong F4 (Table 4.8, p.86), CHC with both BRGC and the Balanced Gray code solved all problems in their first attempt giving confidence that, for suitable problems it can be simply left to find the solution without considering restarts. Finding the solution using the BRGC took substantially fewer generations on average than using the Balanced code. As CHC produces a variable

	Solved	Mean Starts per Solution	Mean Generations per per Solution	Mean Evaluations per Solution
Binary	84.2%	1.07	49,570	<2,478,500
BRGC	90.0%	1.00	1,842	<92,100
Balanced	90.0%	1.00	6,635	<331,750
Monotone Gray	79.9%	1.00	10,414	<520,700

Table 4.14: Summary Statistics for CHC (population 50)

	Solved	Mean Starts per Solution	Mean Generations per per Solution	Mean Evaluations per Solution
Single-point crossover	60.3%	1.16	1,082	1,082,000
HUX	60.5%	1.16	1,136	1,136,000

Table 4.15: Effect of Crossover on Simple GA (population 1000) using BRGC

number of children each generation, a precise number of fitness function evaluations is not given, however, the number of evaluations per generation must be at most the population size. With the BRGC, CHC took fewer evaluations than the GA with a population of 100 and solved more problems.

In the detailed results (Table 4.21), the Dejong F3 (Table 4.8, p.86) and Rastrigin functions (Table 4.9, p.87) were solved most easily (i.e. in the fewest generations) under the Monotone coding. However, using this code CHC was unable to solve F5.

The CHC algorithm and the Simple GA use different crossovers (CHC using HUX (p.67), the Simple GA using a 1-point crossover (p.67)). Running the experiments using a Simple GA (pop. 100) modified to use HUX crossover and BRGC encodings produced similar results to the standard Simple GA (Table 4.15). As 1-point crossover preserves subsequences within the candidate solutions and HUX randomly selects the bits to change, this consistency is surprising. Given the similar performance of the Simple GA using HUX and 1-point crossover, the ability of the CHC algorithm to solve most of the problems from a single start is not simply due to the crossover operator used.

In the detailed results (pp.95–100), the Monotone Gray code was most effective at solving DeJong F3 (Table 4.8, p.86) when using SHC, the Simple GA or CHC. F3 is a “step function” ascending from a minimum at (-5.12, -5.12, -5.12, -5.12, -5.12) and the number of bits in the Monotone Gray code increases with the value – therefore candidate solutions that reduce the number of bits set for a dimension will either remain close to the previous solution or will move closer to the origin.

For separable problems CHC and the GA would be expected to work well as partial solutions can be combined from different members of the population. Steepest Descent would need to solve each problem separately. SHC with a low probability of mutation (e.g. $p = \frac{1}{l}$) would be unable to solve the problems in parallel, and would be more serial in its approach.

At best, the Simple GA (with single-point crossover) gave similar results to SHC (Tables 4.12 and 4.11). However, SHC is significantly cheaper to run than the Simple

GA – unless the problem domain suggests otherwise, SHC with a suitable coding is a better choice than a Simple GA.

However, CHC (Table 4.14) consistently found a solution with a single start – which gives confidence that results found on other problems may be optimal.

4.7 Conclusions

We introduced four optimisation algorithms – two Descent/Hill-Climbing methods (Steepest Descent and a Stochastic Hill Climber, Section 4.1) which search based on a single candidate solution, and two Evolutionary Algorithms (a Simple GA and Eshelman's CHC, Section 4.2) which "evolve" a population of candidate solutions. However, the No Free Lunch theorem states that the performance of algorithms across all functions will be the same. We therefore need to consider the functions being optimised.

For optimisation based on a bit-wise representation of data, the function on which the algorithm operates is not simply the objective function $f(\cdot)$, but the combination of this and the decoding function, Φ, that converts the bit-wise representation into suitable function parameters i.e. the algorithm operates on $f\Phi(\cdot)$. We examined several properties of bit-wise codings that may be relevant to search – the Locality, Balancedness, Hamming Weight – and introduced four bit-wise encodings based on these properties: Standard Binary code, BRGC, and the Monotone and Balanced Gray codes (Section 4.4). We looked at small example codes and the properties for each of these codes (Section 4.5) – introducing a new visualisation showing the effect of the encodings on Steepest Descent, SHC and CHC (Section 4.5.2).

We examined the performance of each combination of algorithm and encoding on a standard set of piece-wise continuous test functions (Tables 4.8 and 4.9) extending the work of Mathias and Whitley [1994]. We found that a suitable encoding allows a simple technique such as Steepest Descent or SHC to solve problems that cannot be solved using the standard Binary encoding (e.g. for SHC only the Balanced Gray code allowed the Schwefel 20×10 problem to be solved, whereas the Rastrigin problem was only solved using the BRGC) – if an optimisation algorithm fails to solve a problem, then it is worth considering alternative codings as well as alternative algorithms. We hypothesise that as each bit in the Balanced Gray code has a similar number of transitions, it gives a more uniform distribution of neighbours across the search space and is therefore particularly effective at exploring the search space.

For the test problems, CHC consistently found the correct solution from a single start using either BRGC or the Balanced Gray Code. However, the BRGC took fewer generations to find the solutions. Assuming that the pitch trajectory estimation problem is more like the test problems than like some arbitrary fitness landscape, we adopted CHC algorithm and BRGC encoding for solving the pitch trajectory parameter optimisation problem.

Test Function	F1 3x10	F2 2x12	F3 5x10	F4 30x8	F5 2x17	Rastrigin 10x10	Schwefel 10x10	Schwefel 20x10	Griewank 10x10	Griewank 20x10
Binary										
Number Solved	100	33	100	0	99	0	0	0	5	0
Average Starts	8.14	506.70	97.81	-	256.65	-	-	-	406.80	-
Average Best	0	0.000010	-30	1.55	0.998004	4.00	-4014.45	-7675.40	0.068631	0.141303
Average Gens	13.85	5.97	12.21	-	12.07	-	-	-	43.80	-
BRGC										
Number Solved	100	48	98	0	100	0	9	0	100	100
Average Starts	1	473.08	286.94	-	2.36	-	525.56	-	29.95	4.49
Average Best	0	0.000001	-29.98	2.60	0.998004	4.86	-4031.41	-7617.48	0	0
Average Gens	16.10	6.83	11.13	-	17.80	-	39	-	51.93	101.20
Balanced Gray Code										
Number Solved	100	49	69	0	100	0	100	100	100	100
Average Starts	1	424.73	397.81	-	12.36	-	3.25	14.12	49.31	3.46
Average Best	0	0.000001	-29.69	2.38	0.998004	2.78	-4189.83	-8379.66	0	0
Average Gens	22.54	9.14	11.01	-	36.21	-	77.48	153.85	72.28	147.14
Monotone Gray Code										
Number Solved	100	36	100	0	100	0	43	0	100	100
Average Starts	1	479.69	182.38	-	2.28	-	474.74	-	82.79	9.33
Average Best	0	0.000002	-30	2.78	0.998004	2.65	-4123.60	-7982.20	0	0
Average Gens	24.52	12.53	12.82	-	60.34	-	53.35	-	72.33	152.84
Maximum Distance										
Number Solved	100	29	100	0	100	0	0	0	2	0
Average Starts	8.62	472.72	227.24	-	7.96	-	-	-	374.50	-
Average Best	0	0.000005	-30	0.374006	0.998004	3.56	-4122.54	-7907.50	0.105763	0.177379
Average Gens	13.69	5.76	10.72	-	17.53	-	-	-	39	-

Table 4.16: Gray Codes with Steepest Descent.

Test Function	F1 3x10	F2 2x12	F3 5x10	F4 30x8	F5 2x17	Rastrigin 10x10	Schwefel 10x10	Schwefel 20x10	Griewank 10x10	Griewank 20x10
Binary										
Number Solved	8	37	100	0	0	0	25	0	0	0
Average Starts	12.50	2.70	1	-	-	-	4	-	-	-
Average Best	0.000158	0.000256	-30	5.83	-	1.52	-4131.62	-7818.37	0.181277	0.398687
Average Gens	165.25	19605.41	413.56	-	-	-	311782.52	-	-	-
BRGC										
Number Solved	100	100	100	0	100	100	100	5	3	7
Average Starts	1	1	1	-	1	1	1	20	33.33	14.29
Average Best	0	0	-30	6.21	0.998004	0	-4189.83	-8099.76	0.096018	0.103639
Average Gens	310.79	24627.23	669.77	-	922.57	20093.82	116949.44	381833.20	424908.67	183943.71
Balanced Gray Code										
Number Solved	100	96	100	0	100	13	100	100	4	8
Average Starts	1	1.04	1	-	1	7.69	1	1	25	12.50
Average Best	0	0.000001	-30	5.80	0.998004	2.12	-4189.83	-8379.66	0.124762	0.092878
Average Gens	428.90	38504.71	743.67	-	4313.32	365545.92	17358.40	79166.72	242904.25	83199.38
Monotone Gray Code										
Number Solved	100	73	100	0	94	81	6	0	8	1
Average Starts	1	1.37	1	-	1.06	1.23	16.67	-	12.50	100
Average Best	0	0.000013	-30	6.16	1.29	0.210163	-3877.32	-7479.40	0.109486	0.144548
Average Gens	460.37	55119.75	337.50	-	4037.24	239178.01	226047	-	267479.12	198373

Table 4.17: Gray Codes with Stochastic Hill Climber

Test Function	F1 3x10	F2 2x12	F3 5x10	F4 30x8	F5 2x17	Rastrigin 10x10	Schwefel 10x10	Schwefel 20x10	Griewank 10x10	Griewank 20x10
Binary										
Number Solved	11	66	100	0	0	0	9	0	0	0
Average Starts	9.09	1.52	1	-	-	-	11.11	-	-	-
Average Best	0.000152	0.000119	-30	1.04	-	1.66	-4148.49	-7731.86	0.223456	0.396582
Average Gens	349	2046.50	249.49	-	-	-	8036.44	-	-	-
BRGC										
Number Solved	100	100	99	0	100	100	97	0	1	4
Average Starts	1	1	1.01	-	1	1	1.03	-	100	25
Average Best	0	0	-29.99	1.25	0.998004	0	-4186.27	-7977.45	0.158252	0.110326
Average Gens	114.87	912.98	642.68	-	158.10	1785.10	3210.31	-	1816	2722.75
Balanced Gray Code										
Number Solved	100	100	95	0	100	13	100	99	0	6
Average Starts	1	1	1.05	-	1	7.69	1	1.01	-	16.67
Average Best	0	0	-29.95	1.30	0.998004	2.19	-4189.83	-8379.65	0.192965	0.098689
Average Gens	141.16	1355.20	889.20	-	284.13	7706.54	1044.50	2839.41	-	2659.33
Monotone Gray Code										
Number Solved	100	100	100	0	100	75	11	0	1	1
Average Starts	1	1	1	-	1	1.33	9.09	-	100	100
Average Best	0	0	-30	2.27	0.998004	0.251706	-3952.79	-7526.41	0.167307	0.141587
Average Gens	144.16	2955.38	99.87	-	444.18	4713.56	3456.91	-	9536	8262

Table 4.18: Gray Codes and Simple GA – Pop 100

Test Function	F1 3x10	F2 2x12	F3 5x10	F4 30x8	F5 2x17	Rastrigin 10x10	Schwefel 10x10	Schwefel 20x10	Griewank 10x10	Griewank 20x10
Binary										
Number Solved	21	100	100	0	100	0	11	0	0	0
Average Starts	4.76	1	1	-	1	-	9.09	-	-	-
Average Best	0.000127	0	-30	1.12	0.998004	1.71	-4150.05	-7690.22	0.220368	0.407944
Average Gens	356.57	1116.30	217.85	-	1437.16	-	6664.73	-	-	-
BRGC										
Number Solved	100	100	100	0	100	100	99	0	0	4
Average Starts	1	1	1	-	1	1	1.01	-	-	25
Average Best	0	0	-30	1.19	0.998004	0	-4188.64	-7953.23	0.166644	0.091919
Average Gens	98	645.75	714.38	-	156.48	1577.79	3290.47	-	-	1825.25
Balanced Gray Code										
Number Solved	100	100	93	0	100	11	100	100	0	10
Average Starts	1	1	1.08	-	1	9.09	1	1	-	10
Average Best	0	0	-29.93	1.24	0.998004	2.02	-4189.83	-8379.66	0.188201	0.098794
Average Gens	129.90	942.79	945.56	-	212.20	7842.55	1089.15	2761.94	-	3351.20
Monotone Gray Code										
Number Solved	100	100	100	0	100	79	10	0	2	6
Average Starts	1	1	1	-	1	1.27	10	-	50	16.67
Average Best	0	0	-30	2.14	0.998004	0.222999	-3948.70	-7578.22	0.172085	0.149744
Average Gens	125.09	1303.55	114.38	-	330.83	5391.48	5861.60	-	4998	3890.50

Table 4.19: Gray Codes and Simple GA – Pop 1000

Test Function	F1 3x10	F2 2x12	F3 5x10	F4 30x8	F5 2x17	Rastrigin 10x10	Schwefel 10x10	Schwefel 20x10	Griewank 10x10	Griewank 20x10
BRGC										
Number Solved	100	100	100	0	100	100	99	0	0	6
Average Starts	1	1	1	-	1	1	1.01	-	-	16.67
Average Best	0	0	-30	1.17	0.998004	0	-4188.64	-7950.90	0.161465	0.101390
Average Gens	110.72	899.20	684.47	-	163.30	1719.82	3075.48	-	-	4184.67

Table 4.20: Gray Codes and Simple GA – HUX

Test Function	F1 3x10	F2 2x12	F3 5x10	F4 30x8	F5 2x17	Rastrigin 10x10	Schwefel 10x10	Schwefel 20x10	Griewank 10x10	Griewank 20x10
Binary										
Number Solved	100	100	100	0	100	100	100	100	86	56
Average Starts	1	1	1	-	1	1	1	1	1.16	1.79
Average Best	0	0	-30	0.476525	0.998004	0	-4189.83	-8379.66	0.003506	0.022915
Average Gens	24631	5264.60	78.55	-	547.93	132532.49	1228.86	4836.22	83090.14	315718.43
BRGC										
Number Solved	100	100	100	0	100	100	100	100	100	100
Average Starts	1	1	1	-	1	1	1	1	1	1
Average Best	0	0	-30	1.00	0.998004	0	-4189.83	-8379.66	0	0
Average Gens	119.90	335.24	100	-	144.42	2759.68	930.74	3230.35	3361	5595.77
Balanced Gray Code										
Number Solved	100	100	100	0	100	100	100	100	100	100
Average Starts	1	1	1	-	1	1	1	1	1	1
Average Best	0	0	-30	0.261254	0.998004	0	-4189.83	-8379.66	0	0
Average Gens	215.94	554.85	105.76	-	970.63	3772.60	2926.09	7007.39	9771.81	34385.76
Monotone Gray Code										
Number Solved	100	100	100	0	0	100	100	100	100	99
Average Starts	1	1	1	-	-	1	1	1	1	1.01
Average Best	0	0	-30	3.72	-	0	-4189.83	-8379.66	0	0.000246
Average Gens	161.25	1639.76	71.95	-	-	1956.39	1723.89	6589.72	8550.64	63141.93

Table 4.21: Gray Codes and CHC.

Chapter 5

Parameter Estimation

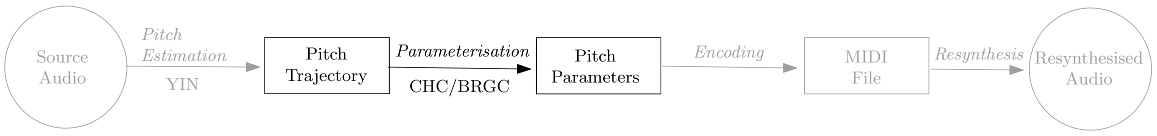


Figure 5.1: System Model: Having selected CHC+BRGC optimisation, we can now estimate pitch parameters

We have proposed an Expressive MIDI model for object coding of audio based on the MMA Downloadable Sounds specification (Chapter 3). In order to use this model to produce an Expressive MIDI encoding of a piece of audio, it is necessary to estimate the model parameters from the source audio material. We examined various optimisation techniques and bit-wise codings (Chapter 4) and found that using Eshelman's CHC with a Binary Reflected Gray Code (BRGC) gave good results on a range of piece-wise continuous problems. We now look to apply this technique to the problem of finding suitable parameter values to represent YIN pitch trajectories in Expressive MIDI (Figure 5.1). [1]

5.1 The DLS Pitch Trajectory Parameter Estimation Problem

For pitch modulation, the MMA Downloadable Sounds (DLS) standard [MMA, 2006] provides an Envelope Generator (EG) and a sine- or triangle-wave based Low Fre-

[1]Parts of this work were originally presented at the 2009 conference on Digital Audio Effects (DAFx 09) [Welburn and Plumbley, 2009a].

quency Oscillator (LFO) using the default modulation routings (as introduced in Chapter 3). We seek to minimise the difference between YIN note pitch trajectory estimates and the pitch trajectories produced using EG+LFO. To do so, we specify a cost function that quantifies this difference and select an appropriate optimisation algorithm to perform the minimisation.

5.1.1 Possible Cost Functions

Pitch is the perceptual analogue of frequency, the difference in pitch indicating how different the frequencies in two pieces of audio will sound to a listener (Section 3.2). The difference between the original pitch trajectory and a trajectory generated from a set of pitch parameters will therefore provide an indication of how similar the pitches of the two pieces of audio will sound.

Using the sum of the absolute pitch differences, $\sum |e_i - g_i|$, as a cost function would reduce pitch differences, however it applies pressure to produce the largest reduction in the pitch difference. Using a cost function based on the square error, $\sum |e_i - g_i|^2$, will apply pressure to reduce large pitch differences – as the effect of a large difference δ on the cost will be related to δ^2. As we are seeking "good enough" pitch estimates, and are more concerned with avoiding large pitch errors than minimising individual pitch errors, the cost function used is based on the root-sum-square-error between the pitch trajectory and the EG+LFO estimate. However, to allow comparisons between the errors for trajectories with different depths, we normalise this error, giving as the cost function:

$$\mathrm{f}_{est}(k) = \sqrt{\frac{\sum_i (e_i - g_i)^2}{\sum_i e_i^2}} = \frac{|\mathbf{e} - \mathbf{g}|}{|\mathbf{e}|} \tag{5.1}$$

where $\mathbf{e} = (e_1, \ldots, e_n)$ is the segment pitch trajectory extracted using YIN (Section 3.2.3) and $\mathbf{g} = (g_1, \ldots, g_n)$ is the EG+LFO estimate.

Minimising the cost function will select pitch parameters which avoid large pitch errors and, we believe, should allow an approximation of the pitch trajectories produced by YIN. In order to test the pitch parameter estimation, we used CHC with BRGC encodings (introduced in Chapter 4) to jointly optimise the pitch parameters. Based on the DLS pitch parameter definitions (Chapter 3) we next consider how to optimise these parameters.

5.2 Joint Optimisation of EG and LFO Pitch Parameters

The DLS pitch parameters include time domain and pitch domain parameters (i.e. the LFO delay, EG Delay, Attack, Hold and Release times; and the base pitch, the EG and LFO depths and the sustain level). Rather than attempting to estimate all parameters directly (e.g. using CHC), we examined the component parts of the parameter model, and found that best-fit pitch domain parameters could be calculated given the time domain parameters. This will reduce the size of the space over which we need to search for optimal parameters whilst guaranteeing optimal pitch parameters given the time parameters.

The EG output (Section 3.1.1) can be considered as the combination of two overlapping sub-envelopes, covering the Attack-hold-Decay and Decay-Sustain-Release phases (EG1 (dAhD) and EG2 ((d+A+h)DSR)):

$$EG1(t) = \begin{cases} 0 & t \le d \\ \frac{t-d}{A} & d < t \le d + A \\ 1 & d + A < t \le d + A + h \\ 1 - \frac{t-d+A+h}{\tau_{\mathrm{D}}} & d + A + h < t \le d + A + h + \tau_{\mathrm{D}} \\ 0 & d + A + h + \tau_{\mathrm{D}} < t \end{cases} \tag{5.2}$$

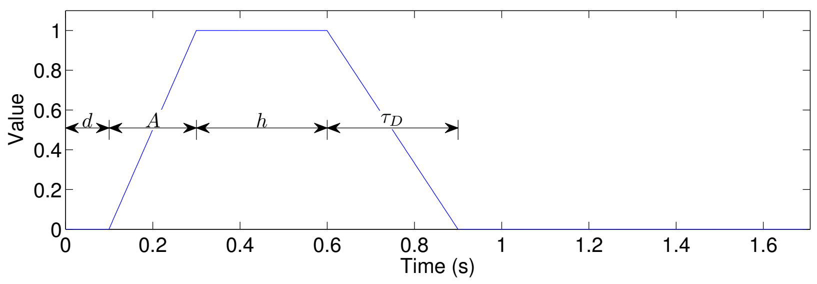
and

$$EG2(t) = \begin{cases} 0 & t \le d + A + h \\ \frac{t-(d+A+h)}{\tau_{\mathrm{D}}} & d + A + h < t \le d + A + h + \tau_{\mathrm{D}} \\ 1 & d + A + h + \tau_{\mathrm{D}} \times (1 - S) < t \le l - \tau_{\mathrm{R}} \\ \frac{l-t}{\tau_{\mathrm{R}}} & n_{off} - \tau_{\mathrm{R}} < t \le l \\ 0 & l < t\ . \end{cases} \tag{5.3}$$

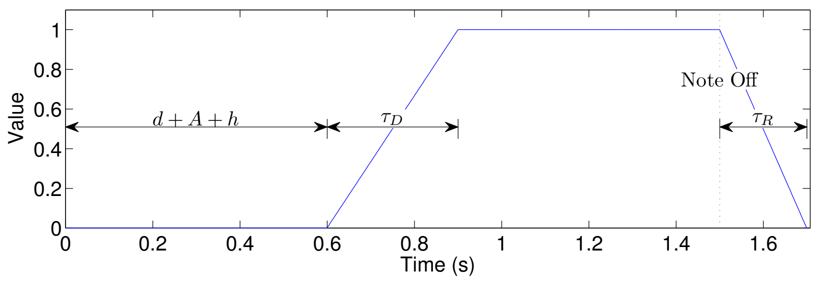
each with its own depth (d_1 and d_2) parameter, in pitch cents, subject to the condition that $d_2 < d_1$ (see figure 5.2).

This allows the overall pitch trajectory to be represented as a linear combination of four components:

- a constant contribution of the base pitch value;
- a contribution from the *dAhD* envelope, $\mathbf{p} = (p_1, \ldots, p_n)$ where $p_i = EG1(t_i)$;



(a) dAhD Envelope



(b) (d+A+h)DSR Envelope

Figure 5.2: Representing EG output as two subenvelopes

- a contribution from the $(d + A + h)DSR$ envelope, $\mathbf{q} = (q_1, \ldots, q_n)$ where $q_i = EG1(t_i)$;
- the LFO depth, l_i.

where t_i is the time offset from the start of the envelope to the ith YIN pitch trajectory estimate.

The overall pitch trajectory estimate, $\mathbf{g} = (g_1, \ldots g_n)$, is then given by:

$$\begin{pmatrix} g_1 \\ \vdots \\ g_i \\ \vdots \\ g_n \end{pmatrix} = \begin{pmatrix} 1 & p_1 & q_1 & l_1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & p_i & q_i & l_i \\ \vdots & \vdots & \vdots & \vdots \\ 1 & p_n & q_n & l_n \end{pmatrix} \begin{pmatrix} b \\ d_{\text{EG1}} \\ d_{\text{EG2}} \\ d_{\text{LFO}} \end{pmatrix} . \tag{5.4}$$

where the base pitch of the segment is b; the EG depth is $d_{\text{EG}} = d_{\text{EG1}}$; the sustain depth is $d_{\text{EG2}} < d_{\text{EG1}}$; the sustain level of the envelope, $S = \frac{d_{\text{EG1}}}{d_{\text{EG2}}}$; and the LFO depth is d_{LFO}, all in pitch cents.

Minimising the cost function (Equation 5.1) minimises the sum-square-error between the pitch trajectory estimate, $\mathbf{g}$, and the YIN data, $\mathbf{e}$. We can calculate values for b, d_{EG1}, d_{EG2} and d_{LFO} that minimise this error from equation 5.4 – given the time-based parameters for the EG and LFO (i.e. $d, A, h, \tau_{\mathrm{D}}, \tau_{\mathrm{R}}, \delta, \frac{1}{f}$).

Noting that $d_{\mathrm{EG1}} = d_{\mathrm{EG}}$ and $d_{\mathrm{EG2}} = d_{\mathrm{EG}} \times S$ and that the expressions for $EG1$ and $EG2$ only use the actual decay time (τ_{D}) and release time (τ_{R}) rather than the individual parameters, we can: take values for the 7 time domain parameters (dAh, τ_{D}, τ_{R}, f, δ); find best-fit values for the pitch domain parameters $d_{\mathrm{EG1}} = d_{\mathrm{EG}}$, d_{EG2}, d_{LFO} and b; and hence calculate the remaining pitch domain parameter $S = \frac{d_{\mathrm{EG2}}}{d_{\mathrm{EG1}}}$.

We have therefore reduced the problem from finding 11 parameters across the pitch and time domains to a problem of finding the best 7 time domain parameters – from which we can calculate the appropriate 4 best-fit pitch domain parameters.

Assuming continuity for the original pitch trajectories, small changes in the time-based parameters will produce small changes in the best-trajectory depths, and small changes in the cost function. Local search abilities (exploiting local features) will allow minima to be found, however the problem includes local optima (Section 3.4) as well as the global optimum. The local optima will need to be avoided using exploration of the search space. An optimisation technique that balances exploration and exploitation will therefore be appropriate – such as CHC with BRGC encoding (as discussed in Chapter 4).

5.3 Assessing CHC+BRGC for EG+LFO Parameter Estimation

5.3.1 Method

We found that CHC with BRGC coding was an effective optimisation technique on test problems (Chapter 4), and have reduced the parameter estimation problem to finding the best time domain parameters (as we can calculate the best-fit pitch domain parameters from these, Section 5.2). We now wish to test the performance of CHC with BRGC to estimate EG+LFO pitch parameters from the YIN pitch trajectories.

To assess the ability of CHC+BRGC to estimate suitable pitch EG and LFO parameters (Section 5.2), 60 files from the RWC Musical Instrument Sounds database [Goto et al., 2003] were selected. Each file consists of a chromatic sequence of notes covering the complete range of an instrument (i.e. notes from the lowest to highest pitch produced by the instrument at 1 semitone intervals). Each instrument has

Instrument (RWC ID)	Number of files	Number of articulations	Dynamics	Number of segments
Piano (011)	12	4	P/M/F	2,079
Organ (061)	8	8	M	382
Violin (151)	2	2	M	134
Trumpet (211)	25	9	P/M/F	844
Clarinet (311)	12	4	P/M/F	526
Total	59	n/a	n/a	3,965

Table 5.1: Instrument Summary.

files covering various articulations (styles of playing) and dynamics (how loudly the instrument is played). The files selected covered various instruments (piano, pipe organ, violin, trumpet and clarinet) and multiple articulations and dynamics (see Table 5.1). The selected instruments have a variety of timbres:

- the trumpet is a conical open tube oscillator including all harmonics;
- the clarinet is a cylindrical closed tube with only odd harmonics;
- the violin uses driven strings connected to a resonating body which emphasises specific frequencies;
- the piano has struck strings of varying stiffness (the increased stiffness of lower strings introducing inharmonicities in the sounds);
- the pipe organ can include both flue pipes (open and closed tubes) and reed pipes giving a variety of timbres from the one instrument.

Using YIN (Section 3.2.3), pitch, power and aperiodicity were estimated for each file. The YIN output was then segmented into individual pitch trajectories based on the aperiodicity (Section 3.3) – 3,965 individual segments were created from the YIN data.

Pitch EG and LFO parameters were estimated by running CHC (Section 4.2.2) with a population of 50 candidate solutions for $10,000$ generations. Each candidate solution consisted of a string of 224 bits (consisting of a 32 bit codeword for each of the 7 time-domain EG+LFO parameters). The initial population was random – 224×50 uniform random numbers were generated and bits were set to 1 for random numbers > 0.5 and zero otherwise.

In order to evaluate the candidate solutions, each codeword was decoded into a 32-bit integer using BRGC (Section 4.4.1, pp.74) and mapped to the appropriate EG+LFO value based on the DLS 32-bit data types (Section 3.1.4). The optimal best-fit pitch parameters (base, EG depth, sustain level, LFO depth) were then calculated (Section 5.2) and the cost function evaluated based on the full set of time and pitch parameters. The estimated pitch trajectories, $\mathbf{p}$, were then compared with the YIN pitch trajectories, $\mathbf{e}$.

As an alternative to using the EG+LFO pitch trajectory estimate, for segment k a constant pitch estimate pitch ρ_c can be used. We can then calculate the cost function using this pitch:

$$\mathrm{f}(\rho_c, k) = \sqrt{\frac{\sum_i (e_i - \rho_c)^2}{\sum_i e_i^2}} \tag{5.5}$$

where $\mathbf{e} = (e_1, \ldots, e_n)$ is the pitch trajectory given by YIN for segment k.

Considering this constant pitch cost function (Equation 5.5), we can find the constant pitch that minimises the function:

$$\begin{aligned} \frac{\partial \mathrm{f}(\rho_c, k)}{\partial \rho_c} &= \frac{\partial}{\partial \rho_c} \sqrt{\frac{\sum_i (e_i - \rho_c)^2}{\sum_i e_i^2}} \\ &= \frac{1}{2} \left(\frac{\sum_i (e_i - \rho_c)^2}{\sum_i e_i^2} \right)^{-\frac{1}{2}} \frac{2}{\sum_i e_i^2} \left(\left(\sum_i e_i \right) - n\rho_c \right) \end{aligned} \tag{5.6}$$

which is 0 when $\rho_c = \frac{1}{n}\sum_i e_i$, i.e. the constant pitch trajectory at the mean YIN pitch minimises the cost function. We will refer to the fitness calculated using a constant trajectory positioned at the mean value as the "mean valued trajectory".

The EG+LFO trajectories found using CHC+BRGC were compared with the original trajectories, and with the cost function value given using the mean valued trajectories.

5.3.2 Overview of Results

Overall Summary

Of the $3,965$ segments processed, $2,824$ (71.22%) had a better fit using the EG+LFO trajectory than by using the mean valued trajectory (Table 5.2). Allowing CHC to run for more generations should allow it to explore the solution space further and could increase the likelihood of finding good solutions. Rerunning CHC for $100,000$ generations on a selection of files showed that there were cases when the fitness of the

Instrument	RWC ID	% $f_{EG+LFO} <= f_{mean}$
Piano	011	74.12%
Pipe Organ	061	84.55%
Violin	151	67.91%
Trumpet	211	61.26%
Clarinet	311	66.92%
Overall		71.22%

Table 5.2: Percentage of segments with lower EG+LFO cost (f_{EG+LFO}) than cost for the mean valued trajectories (f_{mean}) for each instrument and overall.

best candidate solution for a segment improved after 10,000 generations had passed (Figure 5.3).

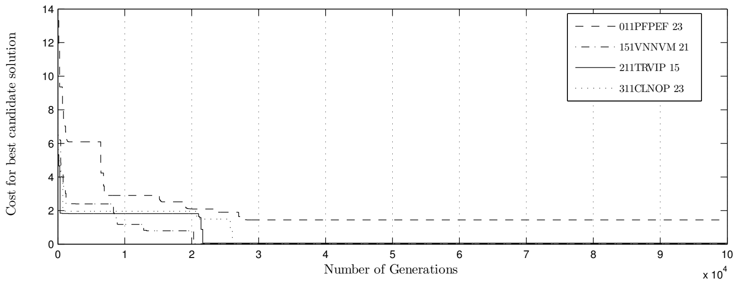


Figure 5.3: Variation of cost of the best candidate solution with the number of generations that CHC has run. The selected segments show that the cost of the best candidate solution can improve beyond 10,000 generations of CHC.

By Instrument

We found that the EG+LFO estimate gave a lower cost function value than the constant mean valued trajectories for most segments (Table 5.2). Overall, for each instrument over 60% of pitch estimates had lower costs using the EG+LFO model than the mean valued trajectories – over 84% being better for the pipe organ segments. Noting that the mean valued trajectories are the constant valued trajectories that minimise the cost function, CHC + BRGC has effectively minimised the cost function for many segments.

By File

Examining the results in detail by file (Table 5.3), we discover a large variation in our ability to estimate $EG + LFO$ parameters for each file. In general, the parameter

File	Number of Segments	% $f_{EG+LFO} < f_{mean}$
011PFNOF	143	83.92%
011PFNOM	125	85.60%
011PFNOP	114	64.91%
011PFPEF	343	81.34%
011PFPEM	156	75.64%
011PFPEP	151	83.44%
011PFREF	28	75.00%
011PFREM	331	66.16%
011PFREP	259	62.55%
011PFSTF	112	80.36%
011PFSTM	134	73.13%
011PFSTP	183	69.40%
151VNNOM	64	78.13%
151VNNVM	70	58.57%
311CLNOF	40	75.00%
311CLNOM	40	67.50%
311CLNOP	40	50.00%
311CLSTF	40	72.50%
311CLSTM	41	65.85%
311CLSTP	40	62.50%
311CLTLF	63	77.78%
311CLTLM	58	72.41%
311CLTLP	44	72.73%
311CLVIF	40	65.00%
311CLVIM	40	55.00%
311CLVIP	40	57.50%

(a) Piano, Violin and Clarinet

File	Number of Segments	% $f_{EG+LFO} < f_{mean}$
061ORNOM	56	91.07%
061ORP1M	30	90.00%
061ORP2M	31	96.77%
061ORR1M	40	77.50%
061ORR2M	59	86.44%
061ORR3M	56	66.07%
061ORR4M	56	83.93%
061ORSTM	54	90.74%
211TRM7F	31	58.06%
211TRM7M	31	48.39%
211TRM7P	31	61.29%
211TRM8F	31	51.61%
211TRM8M	34	55.88%
211TRM8P	32	50.00%
211TRM9F	31	61.29%
211TRM9M	31	64.52%
211TRM9P	31	54.84%
211TRNOF	35	54.29%
211TRNOP	34	55.88%
211TRSTF	45	57.78%
211TRSTM	35	71.43%
211TRSTP	37	59.46%
211TRVIF	31	45.16%
211TRVIM	31	54.84%
211TRVIP	32	28.13%
211TRW1F	34	82.35%
211TRW1M	31	61.29%
211TRW1P	31	80.65%
211TRW2F	30	66.67%
211TRW2M	30	66.67%
211TRW2P	31	74.19%
211TRW3F	33	78.79%
211TRW3M	30	73.33%
211TRW3P	31	77.42%

(b) Pipe Organ and Trumpet

Table 5.3: Percentage of segments with lower EG+LFO cost (f_{EG+LFO}) than cost for the mean valued trajectories (f_{mean}) by RWC file.

estimation was successful for the pipe organ files (061OR*), EG+LFO estimates having a lower cost for 96.77% of segments in the 061ORP2M file and only 061ORR3M improving over the mean valued trajectories for less than 75% of segments. For the piano files, 5 of the 12 had a lower EG+LFO cost for more than 80% of segments and at least 62.55% of segments were better with EG+LFO than using the mean. Of the violin files, parameter estimation was largely effective on the *vibrato* violin (151VNNOM at 78.13%) but less so for the *non-vibrato* file (151VNNVM at 58.57%). The trumpet files showed a large variation – quiet, vibrato trumpet (211TRVIP) produced extremely poor results with only 28.13% of segments better and loud, normally articulated trumpet with a wow-wow mute (211TRW1F) achieved improvements for 82.35% of segments. Clarinet files were neither particulary good nor exceedingly bad (ranging from 50.00% to 77.78% with EG+LFO better than using the mean).

Example Output

For several segments from each instrument, including both vibrato and non-vibrato trumpet and violin, the following figures (5.4 to 5.10) plot the YIN pitch trajectories (paler, background lines) and the EG+LFO pitch trajectories (the bolder smoother lines) showing the offset of both trajectories from the median pitch for each segment – the median pitch being unaffected by outliers in the YIN pitch trajectories. For each figure, we consider the performance of the EG+LFO pitch estimation.

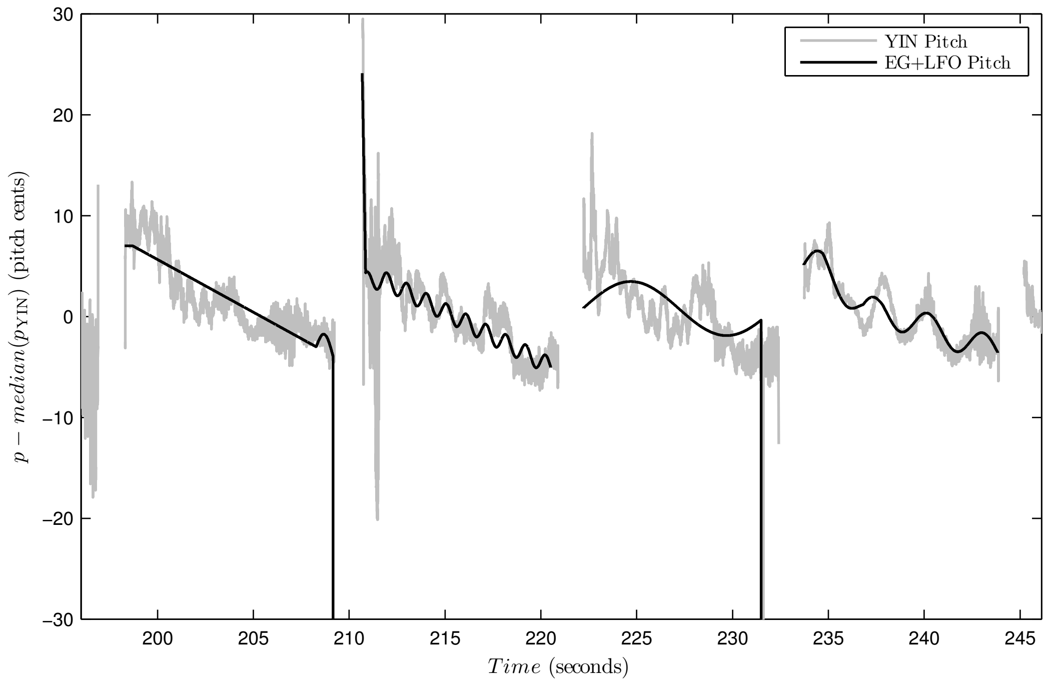


Figure 5.4: *011PFPEF: Piano (with pedal) pitch estimation.* The YIN piano pitch estimates (grey lines) show a similar decrease in pitch for each segment and variations during the segment. The 4 EG+LFO trajectories (black lines) are quite different providing smoothed approximations of the trajectories that bear little resemblance to the "basic" EG+LFO trajectory (Section 3.1.3). The first segment uses the EG to approximate the slope and only applies the LFO to add a small kick at the end of the segment; the second uses the EG to provide both an initial pitch drop and the slope, using the LFO to provide variation through the segment; the third uses the EG to provide a final pitch drop and attempts to match the overall shape using the LFO; and the fourth provides an initial attack and a downward slope using the EG and closely matches the variations in pitch using the LFO. The YIN piano pitch variation is mainly in the ±10 cent range.

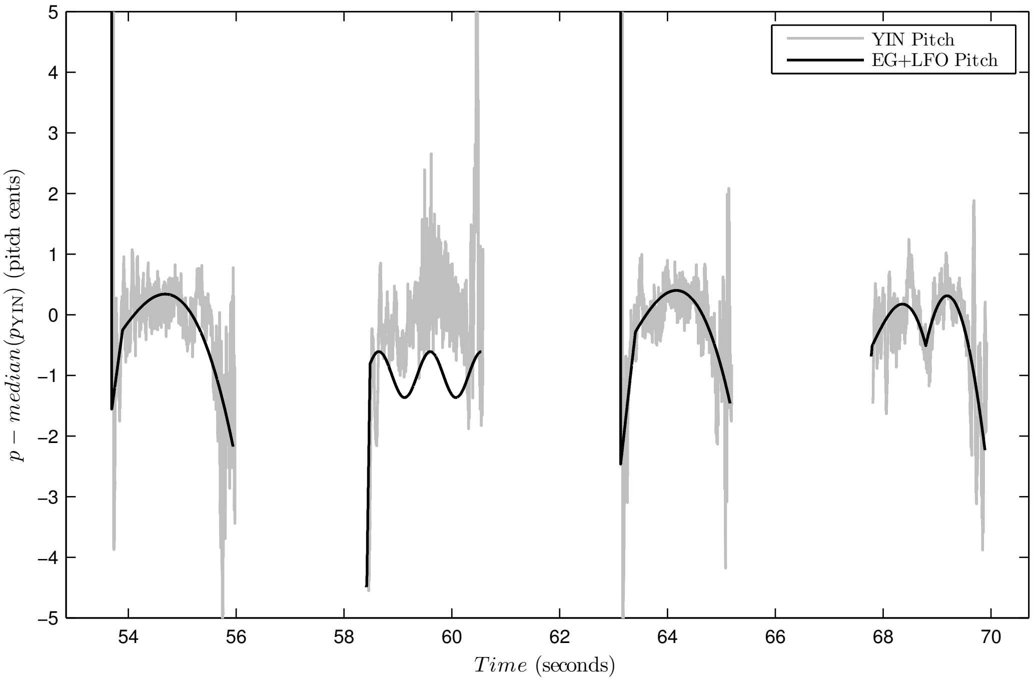


Figure 5.5: *061ORNOM: Pipe Organ pitch estimation.* Noting the smaller scale of the graph, the YIN pitch trajectories (grey lines) for the pipe organ segments show less variation than the hammered-string sounds of the piano (Figure 5.4) – the main variation in the pipe organ notes being in the range ± 2 cents. However, the first three notes have large pitch changes at the start and/or end. These are artifacts of the segmentation procedure. The EG+LFO parameters produced similar trajectories for the first and third segments using the EG to provide the initial pitch drop rise and modelling the shape to the end of the trajectory using the LFO. The second segment used the EG for the initial pitch rise and applied the LFO to create peaks and troughs across the segment, matching the alignment (but not the depth) of the overall YIN pitch trajectory peaks and dips. The fourth trajectory provides a surprising combination of the EG and LFO, closely matching the shape of the YIN pitch trajectory.

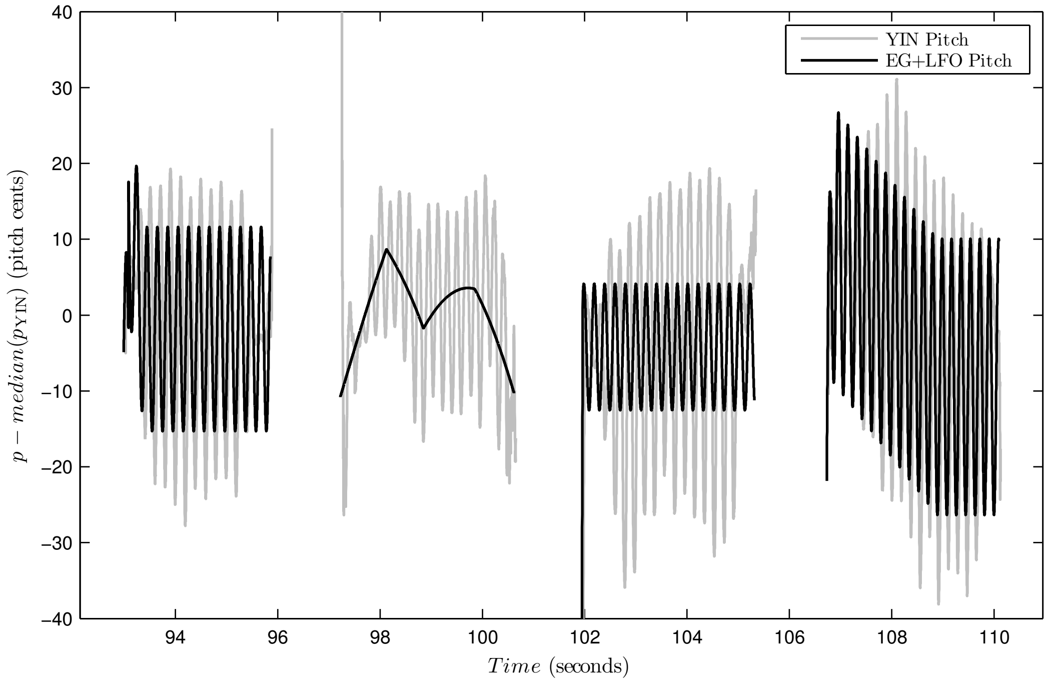


Figure 5.6: *151VNNOM: Violin pitch estimation.* The standard violin articulation shows distinct vibrato. For three of the four segments, both the phase and the frequency of the vibrato has been closely matched, additional characteristics such as the initial shape of the trajectory, also being matched. For these segments, the "best-fit" constant depth for the vibrato also appears good. For the second segment, gross features of the YIN pitch trajectory (the initial increase, the central dip, and the final fall) were reasonably matched – the LFO being used to provide the second rise in the trajectory. We note that the vibrato depth for this more is greater than ±10 cents.

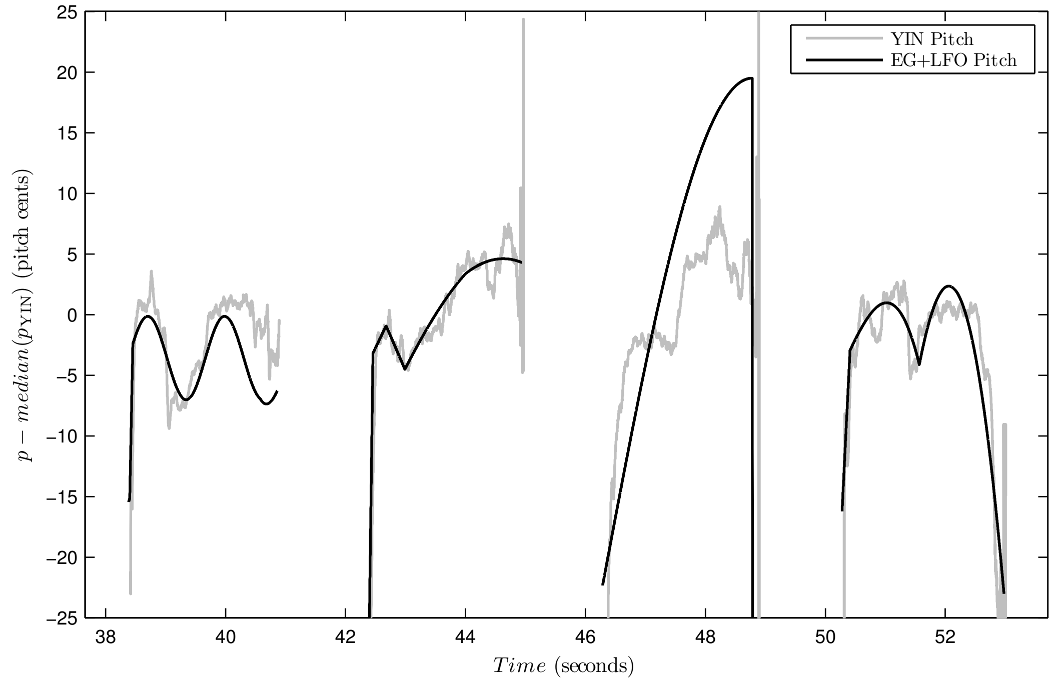


Figure 5.7: *151VNNVM: Violin pitch estimation (no vibrato).* Violin segments were also processed from samples without vibrato. A range of approximately ±5 cents covers the YIN pitch variation excluding the starts and ends of the segments. Given the EG+LFO model, the first two segments and the last segment are well matched – only the first showing the expected use of the LFO, it being used for the overall trajectory shape for the second and fourth segments. The third segment starts from a low value and ends with a large pitch drop and subsequent rise, possibly as result of poor segmentation, and the EG+LFO trajectory attempts to capture this shape (the square error based cost function applying pressure to reduce the largest pitch errors). In doing so, the EG is used to provide the final pitch drop and the LFO provides the shape across the segment.

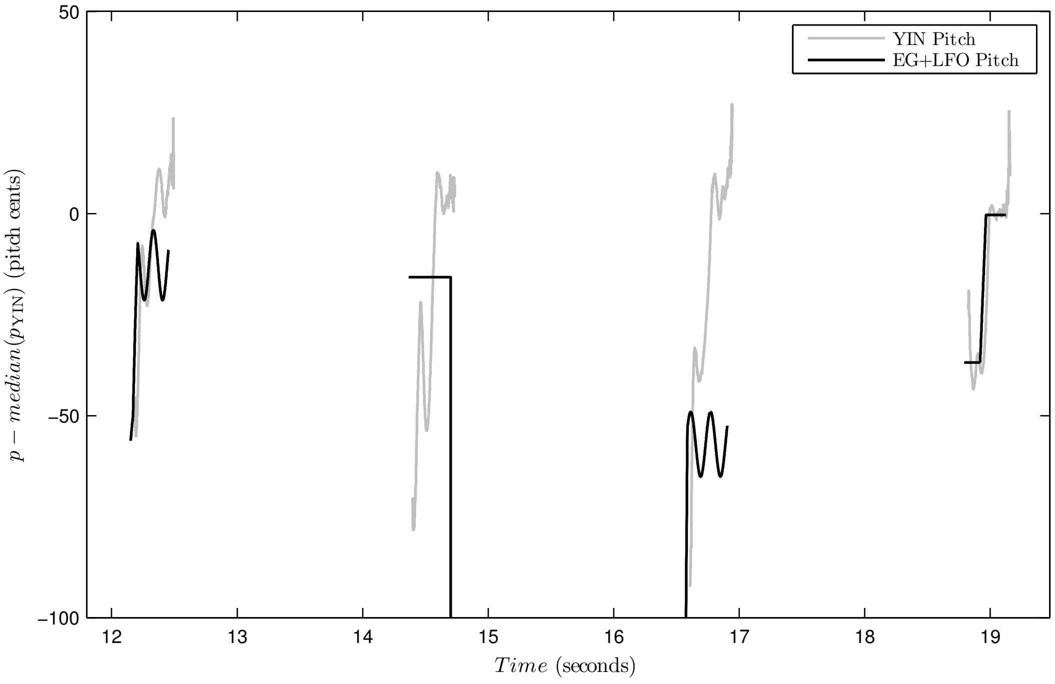


Figure 5.8: *211TRSTF: Staccato trumpet pitch estimation.* Staccato trumpet segments offer little data from which to estimate parameters. None of the four segments capture the rise in pitch throughout the segment, although the fourth example successfully matches the two plateaux in the pitch. The first and third segments, however, almost match the timings of the peaks at the start of and half-way through the trajectory using the LFO – although the pitch at the peaks is wrong. CHC was unsuccessful in finding suitable parameters for the second segment. It is notable that the YIN pitch trajectories for these segments cover a larger pitch range than other instruments – the pitch rises from ca. 50 cents (0.5 semitones) below the median pitch to 25 cents above the median with the pitch for the final half of each segment being relatively constant.

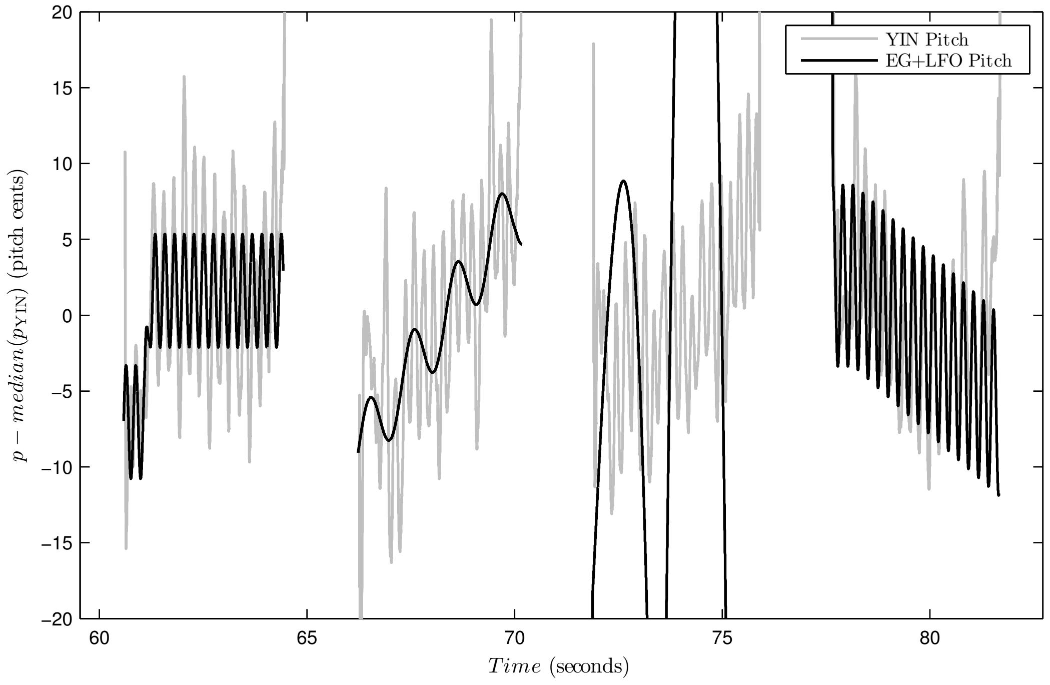


Figure 5.9: *211TRVIP: Vibrato trumpet pitch estimation.* For vibrato trumpet segments we again managed to match frequency and phase in some segments. Additionally, the major features of the trajectories were also represented (low then higher for the first segment, increasing for the second, decreasing for the fourth). However, for the second segment the LFO was not matched – the square error cost function matching the larger pitch variations within the note (e.g. the pitch drops at each trough in the LFO). For the third segment CHC completely failed to find a reasonable estimate of the trajectory in $10,000$ generations. The good performance on some segments is notable as this audio file generally had better pitch estimates using the mean YIN pitch than using the EG+LFO model (pp.110).

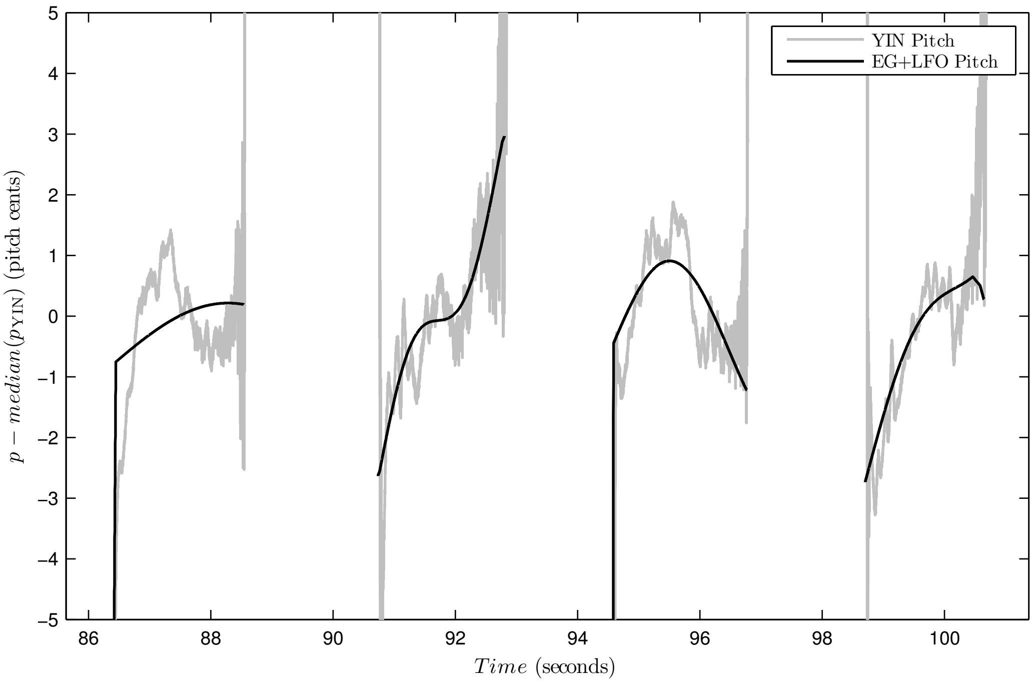
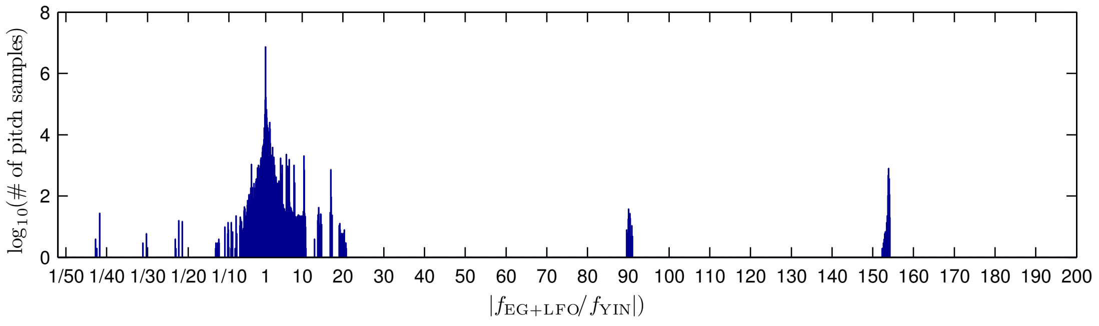
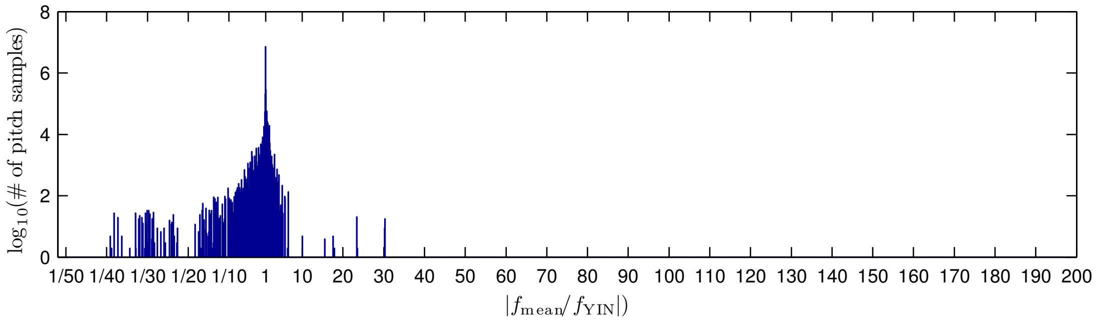


Figure 5.10: *311CLNOP: Clarinet pitch estimation.* The final instrument processed was the clarinet. The pitch variation is less than 4 cents apart from at the segment ends and the overall shape of the pitch trajectories is reasonably represented. The segments did not have a vibrato element, and the LFO was used to provide the gross pitch trajectory across each segment. Although the EG+LFO trajectory for the first segment appears a poor match for the YIN pitch trajectory, the range of the pitch variation during the segment (excluding the start and end of the segment) is only ca. ± 1 cent.

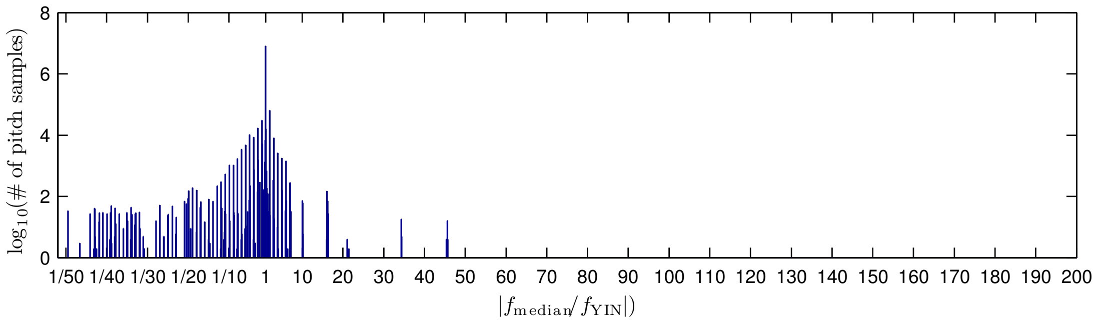
5.3.3 Distributions of Errors



(a) EG+LFO pitch estimate



(b) Mean pitch estimate



(c) Median pitch estimate

Figure 5.11: All errors using EG+LFO, mean and median pitch estimates showing the frequency ratio vs. $\log_{10}$ of the number of pitch estimates.

Although the mean pitch minimises the cost function in Equation 5.5, it is affected by outliers – large errors caused by poor segmentation can cause the mean to differ from most of the values in the pitch trajectory. Therefore, as an alternative to using the mean YIN pitch estimate we looked at the median YIN pitch estimate which is less affected by the values of outliers. We found that 74.65% of EG+LFO estimates had a lower cost than the median pitch estimate – however, the cost function is not particularly suitable for the median estimate – we already know that the mean YIN pitch estimate minimises this function. We therefore considered an alternative cost function:

$$\hat{\mathrm{f}} = \sum_{i=1}^{\hat{n}} |g_i - e_i| \ . \tag{5.7}$$

For this cost function, although the EG+LFO estimates were designed to minimise f, for 64.82% of the segments the EG+LFO estimate had a lower cost than the median YIN pitch estimate.

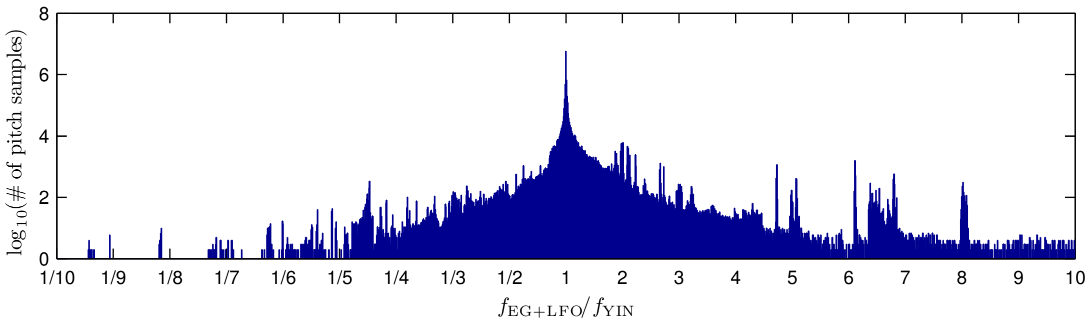
Examining the ratio of the EG+LFO, mean and median pitch estimates ($f_{\mathrm{EG+LFO}}$, f_{mean}, f_{median} respectively) to the initial YIN output (f_{YIN}), we see (Figure 5.11) that the EG+LFO estimates produced fewer errors in the range $\frac{1}{50} f_{\mathrm{YIN}} \leq f_{\mathrm{est}} \leq \frac{1}{10} f_{\mathrm{YIN}}$ than the mean and median pitch estimates. However, there are occasions when EG+LFO fails, resulting in estimates $f_{\mathrm{EG+LFO}} \approx 90 f_{\mathrm{YIN}}$ and $f_{\mathrm{EG+LFO}} \approx 154 f_{\mathrm{YIN}}$. We hypothesise that these errors occur when outliers cause the EG+LFO algorithm to fail (as for the third vibrato trumpet segment in Figure 5.9).

Both the mean and median valued constant pitch trajectories also show large differences to the YIN pitch trajectory. As each file in the RWC database consists of a series of notes at semitone intervals, we expect segments, which are either individual notes or portions of notes, to not be varying over more than one semitone. In this case, all pitch estimates in the YIN pitch trajectory should be around the note pitch ±1 semitone – and the mean and the median values should also be in this range. Given the relationship between pitch and frequency (Section 2.4):

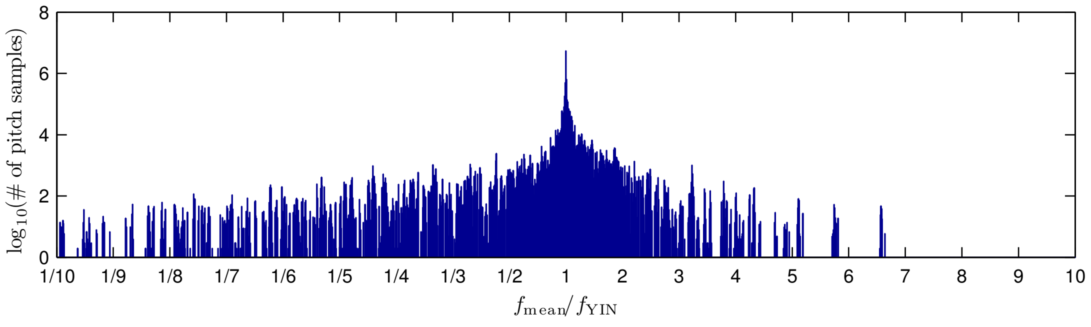
$$\frac{f_{est}}{f_{YIN}} = 2^{\frac{p_{est} - p_{YIN}}{12}} \tag{5.8}$$

implying that $2^{-\frac{2}{12}} \leq \frac{f_{est}}{f_{YIN}} \leq 2^{\frac{2}{12}}$ for a range of ±1 semitone in both the estimated pitch trajectory and the YIN pitch trajectory. The large variation of the mean and median pitch relative to the YIN pitch suggests YIN pitch estimates vary by more than one semitone either as a result of errors in the YIN pitch estimation or as a result of poor segmentation of the notes.

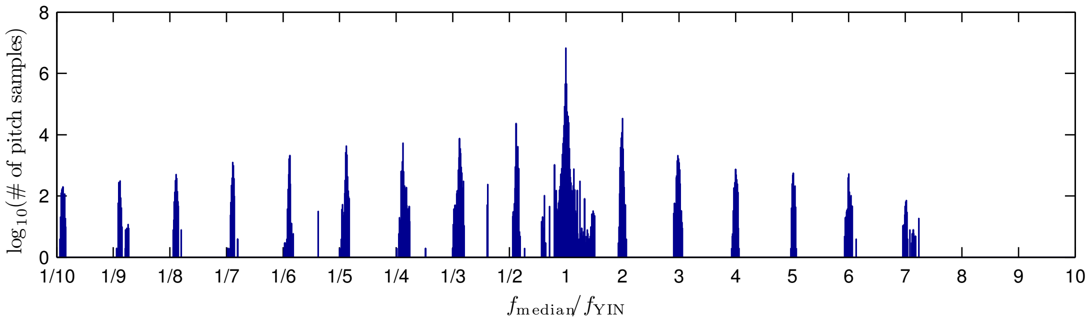
As most errors occur in the region $\frac{1}{10} \leq \frac{\mathrm{f}_{.}}{\mathrm{f}_{YIN}} \leq 10$, we examined the smaller errors using the EG+LFO, mean and median pitch estimates (Figure 5.12 shows this region from Figure 5.11 in more detail). We see that the median pitch estimate mainly results in a frequency ratio ($\frac{f_{\mathrm{median}}}{f_{\mathrm{YIN}}}$) either close to 1 or close to a an integer or fraction – i.e. errors using this estimate are either small or have a *harmonic* relationship to the median YIN pitch estimate. The harmonic relationship between the YIN pitch estimates and the median pitch estimate therefore suggests harmonic errors in the



(a) EG+LFO pitch estimate



(b) Mean pitch estimate



(c) Median pitch estimate

Figure 5.12: Details of smaller errors (in the range $f_{\mathrm{YIN}}/10 \leq f_{\mathrm{est}} \leq 10 f_{\mathrm{YIN}}$) using EG+LFO, mean and median pitch estimates showing the frequency ratio vs. $\log_{10}$ of the number of pitch estimates, the median pitch estimate showing clearly that YIN is making octave / harmonic errors (see text).

output of the YIN pitch estimation. Such harmonic errors are discontinuities in the pitch trajectory and will mean that it may not be possible to find suitable parameters cannot be found for the almost continuous EG+LFO parameter model – the EG+LFO model allows three discontinuities if the Attack, Decay or Release rates are zero.

The EG+LFO and mean pitch estimates are less closely tied to the original pitch estimates, outliers in the segment pitch moving the estimates away from the actual pitch trajectory, this results in a less "spiky" distribution of errors.

5.4 Conclusions

Using the EG+LFO estimate of the pitch trajectory from the Expressive MIDI model, we produced similar trajectories to those estimated from the audio, using the LFO to represent both vibrato effects and more complex non-vibrato trajectories. This trajectory can more closely approximate the original audio than simply applying a constant pitch and represents the trajectory in a small number of parameters per segment. The EG+LFO estimate provides a smoothed approximation of the pitch trajectory and often bore little resemblance to the "basic" EG+LFO trajectory (Figure 3.1.3) – the LFO often being used for the gross trajectory shape (e.g. the no-vibrato violin notes in Figure 5.7).

With vibrato examples, both frequency and phase can be matched (as in Figures 5.6 and 5.9). However, with more complex pitch variation, the minimum error found may use the LFO to represent coarser features than the vibrato and omit representing any vibrato that is present (as seen in the second vibrato violin segment in Figure 5.6). In order to capture the vibrato effect, we propose a two stage process: first using CHC to estimate gross features of the pitch trajectory with the EG; and then separately finding the LFO parameters to best represent the residual pitch trajectory unaccounted for by the EG (e.g. using autocorrelation on the residual pitch trajectory after the EG estimate is used to estimate the LFO frequency and subsequently estimating the LFO delay using cross-correlation of the residual trajectory and a "test" LFO based on a zero-padded sine wave).

Segmentation errors and problems estimating the pitch at the ends of segments result in sudden pitch changes which either: are poorly represented using the EG+ LFO model; or are estimated and disturb the estimation of the remaining pitch trajectory features. Additional work is therefore required to improve the segmentation of the pitch trajectories. This is unsurprising given the basic segmentation used (Section 3.3). For the RWC Musical Instrument Sounds database, each individual note is separated by a period of silence. Better segmentation can therefore be achieved by: splitting the audio into segments at the silences and then refining those segments using power and/or aperiodicity thresholds to refine the note boundaries.

The combination of square error cost function and large variations in the YIN pitch estimates produce pressure on the system to match the large variations in pitch. Ignoring the outliers or using an absolute difference (L_1 norm, $|\cdot|$) cost function would reduce the pressure to match these variations and may produce a more accurate estimate of the smaller variations in pitch across the note.

Comparing a constant valued pitch trajectory at the median YIN pitch with the YIN pitch trajectory revealed the presence of harmonic errors in the YIN output. The approximate MIDI pitch for each note in the RWC Musical Instrument Sounds database can be found based on the instrument range and the notes being at semitone intervals. The YIN pitch estimates could therefore be post-processed using this data to remove some of these harmonic errors / outliers.

For most segments, the EG+LFO model produces a lower cost than using a constant trajectory at the mean pitch – the constant trajectory that minimises the cost function under consideration. We therefore believe that Expressive MIDI using the EG+LFO model will produce a better pitch trajectory estimate than a simple constant pitch for many notes. Although we have seen problems with the estimated EG+LFO pitch trajectories, we next examine the final step in an Expressive MIDI system – resynthesising audio from estimated EG+LFO pitch trajectories – and consider whether, given suitable estimates of the EG+LFO parameters, audio can be produced which matches a parameterised pitch trajectory.

Chapter 6

(Re)Synthesis

Having extracted segments of pitch trajectories from source audio using the YIN pitch estimator (Chapter 3), we used the CHC optimisation algorithm with a BRGC encoding (as described in Chapter 4) to estimate pitch trajectory parameters based on an EG+LFO model for note pitch trajectories (Chapter 5)[1].

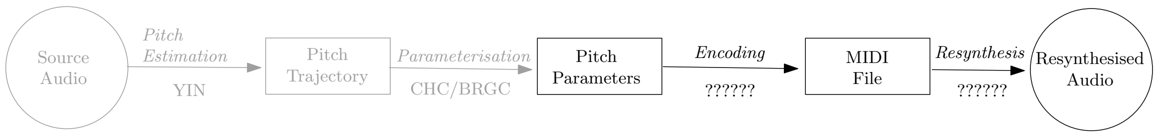


Figure 6.1: System Model: With YIN and CHC+BRGC in place, the remaining components are a suitable encoding and a resynthesis technique

In order to create the Expressive MIDI system for creating audio based on the source audio pitch trajectories, we need to resynthesise audio using the found parameters. The parameters are to be passed to the synthesiser using a Standard MIDI File (SMF, Section 2.5.1). We therefore require: a suitable synthesiser to resynthesise the audio using the pitch trajectory parameters; and a definition of the MIDI data which will be used to pass the parameters (Figure 6.1). The appropriate MIDI data will allow the estimated parameters to be passed to the chosen resynthesis technique. In order to define the MIDI data, we must know how we will resynthesise the audio. Therefore, we next consider selecting a suitable synthesiser for the resynthesis.

[1]Parts of this work were originally presented at the 127th Convention of the Audio Engineering Society [Welburn and Plumbley, 2009c].

Manufacturer	Software Sampler	SoundFont Import ?	Envelope Format
Steinberg	HALion 3	✓	ADSR / Up to 32 points
Native Instruments	Kontakt 3	✓	AhDSR / DBD / Up to 32-points
TASCAM	Gigastudio 4	via converter	ADSR
E-mu	Emulator X3	via converter	"EOS"
FL Studio	DirectWave	✓	ADSR
	SoundFont Player	✓	ADSR
Ableton	Sampler	✓	ADSR
Mark Of The Unicorn	MachFive 2	✓	AhDSR / Multipoint

Table 6.1: Software Samplers. The E-mu "EOS" envelope model is an ADSR style envelope with two-part Attack, Decay and Release phases.

6.1 Synthesiser Selection

Having selected the MMA DLS specification as the basis for the parameters, we need a suitable synthesiser to apply the parameters to produce audio. However, the DLS standard only supports 6 controllers and there are 9 parameters in our EG+LFO pitch model. As DLS is a "samples + synthesis" model, we looked for a software synthesiser that could play back samples in a manner similar to DLS.

Continuing the aim of building a system from existing components, we chose to use an existing commercial synthesiser rather than developing our own. Software samplers were considered – these allow sample playback to be modulated using envelope generators, oscillators, filters etc. Of the software samplers considered (Table 6.1), HALion 3, Kontakt 3, Emulator X3 and MachFive 2 support envelope generators more complex than "ADSR". However, only Native Instruments[2] Kontakt supports scripting to allow control of sampler parameters via MIDI messages.

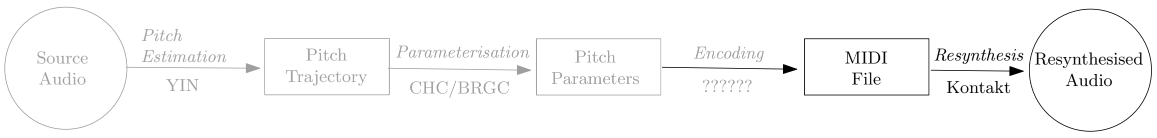


Figure 6.2: System Model: Only encoding the audio data is undefined

The Kontakt 3 modulators [Marinic et al., 1994] do not match the DLS specification (Chapter 3):

[2]http://www.native-instruments.com

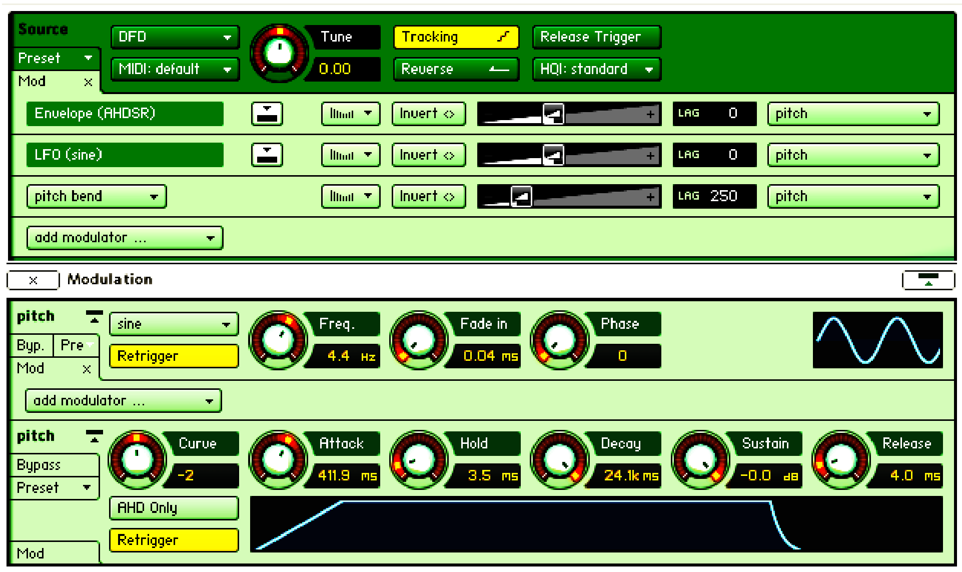
- the envelope generator model is AhDSR rather than dAhDSR (i.e. it doesn't support an initial delay segment);
- envelope generator timing parameters are supported up to 10, 15 or 25 seconds rather than 40 or 50 seconds for DLS;
- the output of the envelope generator is not linear in shape, although the attack slope can be set to linear;
- and the LFO supports a "fade" parameter (which gradually applies the LFO over the specified time) and an initial phase rather than an initial delay.

Nevertheless, we believe that the Kontakt specification and the ability to script its behaviour should allow a close approximation of the pitch of the source audio using the parameters previously estimated and that any errors resulting from necessary modifications should be observable in the output. As Kontakt is the commercial synthesiser with the closest specification to DLS we chose to adopt it as the synthesis engine (Figure 6.2) and to bear in mind the variations from the original system when examining the resynthesised audio.

6.2 Implementation in Kontakt

Kontakt can be used as either a stand-alone piece of software or as a sequencer plug-in. In order to develop a custom synthesiser using Kontakt we can: load samples into Kontakt voices; attach the required modulators to each voice; and add Kontakt Script Processor (KSP) scripts [Marinic et al., 1994] to each voice to control the modulators.

Kontakt allows the import of samples from Creative Labs SoundFonts (a similar technology to DLS, Section 3). The SCC1t2 GM SoundFont is a GM/GS compatible SoundFont based on samples from the Roland Sound Canvas. The audio content comprises small 16 bit 44.1 kHz samples, the entire SoundFont being approximately 3.2 megabytes, a small number of cycles of the waveform being used as the loop region of each sample. These basic samples are appropriately characterless allowing the Expressive MIDI parameters to add expression during resynthesis. The SoundFont was imported into Kontakt assigning instruments to Kontakt voices according to the General MIDI program numbers (Section 2.5.3). Each instrument has several samples across the pitch range reducing how far individual samples need pitch-shifting to produce appropriately pitched output (see multisamples, p.40). The SCC1t2 import included data on the base pitch for samples, and the MIDI note numbers each sample should respond to.


Source: DFD
Preset: MIDI: default
Tune: 0.00
Tracking
Reverse
Release Trigger
HQI: standard
Envelope (AHDSR) — Invert — LAG 0 — pitch
LFO (sine) — Invert — LAG 0 — pitch
pitch bend — Invert — LAG 250 — pitch
add modulator ...

Modulation
pitch: sine — Retrigger
Freq. 4.4 Hz
Fade in 0.04 ms
Phase 0
add modulator ...
pitch: Curve -2
Attack 411.9 ms
Hold 3.5 ms
Decay 24.1k ms
Sustain -0.0 dB
Release 4.0 ms
AHD Only
Retrigger

Figure 6.3: Kontakt Modulations.

Pitch modulators were added to the required Kontakt instruments (Violin, Trumpet and Clarinet) for: a sine-wave based LFO ("LFO (Sine)"); an AHDSR envelope generator ("Envelope (AHDSR)"); and for the pitch bend controller ("Pitch Bend") (Figure 6.3). A Kontakt script was then attached to each individual instrument to interpret the MIDI parameters and to provide the appropriate Kontakt modulator settings (Appendix A.2).

As the Kontakt LFO and EG (Section 6.1) differ from the DLS specification, parameters either needed to be re-estimated, or we needed to adapt the given values for the new model. We chose to use the previously estimated parameter values (Chapter 5) and to set the Kontakt to approximate their behaviour:

- the Kontakt AhDSR EG does not have a delay time – if the delay time was less than half the note length, then the Kontakt EG attack time was set to cover both the estimated delay and attack times (just EG no delay), otherwise, the EG depth was set to zero (just delay no EG);

- the Kontakt AhDSR EG only allows attack times up to 15 seconds rather than the 40 seconds supported by DLS – if the total attack and delay time was more than 15 seconds (which is greater than the note lengths examined), then the Kontakt EG attack time was set to 15 seconds and the EG depth was scaled to reach the same pitch that the longer attack would have reached after 15 seconds;

Kontakt Parameter	Formula	Units for EG+LFO variables
LFO Frequency	$2.31800\log_{10}(f) + 4.59800$	f frequency in Hz
Sustain depth	$1000000\left(\frac{s}{100}\right)^{\frac{1}{3}}$	s sustain depth as a % of the total envelope generator depth
Times	$1000000(\log_2(t+2)-1)/\alpha$	t time in ms
		$\alpha = \log_2(t_{\max}+2)-1)$
		$t_{\max}$ is the maximum time allowed (10, 15 or 25 seconds)
Pitch	$500000\left(1+\left(\frac{p}{1200}\right)^{\frac{1}{3}}\right)$	p in pitch cents

Table 6.2: Formulae to relate Kontakt parameter values (0–1000000) to EG+LFO parameters in units shown

- the Kontakt AhDSR EG has a maximum hold value of 15 seconds and decay and release times of 25 seconds rather than the 40 seconds supported by DLS – times longer than 15 / 25 seconds were set to 15 / 25 seconds;
- the Kontakt LFO does not have a delay time – if the delay time was less than half the note length, then the Kontakt LFO fade time was set to cover the delay time, otherwise, the LFO depth was set to zero (just delay, no LFO);
- the Kontakt LFO fade time has a maximum value of 10 seconds rather than the 40 seconds supported by DLS – times longer than 10 seconds were set to 10 seconds.

We believe that, if Kontakt accurately reproduces the pitch trajectory based on these modified parameter values, then the results of these approximations should be apparent in the pitch trajectories output from Kontakt. Any differences observed between the pitch of the output audio and the expected trajectories, which can be explained by these approximations would then suggest that reestimating parameters specifically for the Kontakt parameters would be an appropriate step to improve the results.

Kontakt parameters are specified as integers in the range 0 to $1,000,000$ and their relationships to modulator values (e.g. times in seconds, frequencies in Hz) are not given by Native Instruments. However, reverse engineered formulae exist for the these relationships in the "KSP Math Library Technical Manual" [Villwock, 2009]. We validated these functions against various parameter values (Appendix A.1) and produced the formulae in Table 6.2 to map EG+LFO parameter estimates to Kontakt parameter values.

Given these mappings between the EG+LFO parameter values and internal Kontakt parameters, we next consider how to use a Standard MIDI File (SMF, Section 2.5.1) to pass the parameters into Kontakt.

6.3 Parameter Encoding

Using MIDI, the basic pitch of a note is provided by the Note On and Note Off messages, conventional MIDI tuning (p.35) specifying the pitch to the nearest semitone based on a value of 69 representing 440Hz. In order to provide finer control of the pitch, we used the 14-bit pitch bend controller to provide an offset from the base pitch to the nearest semitone. The pitch bend range was set to ±0.5 semitones (±50 pitch cents) allowing the full range of pitch bend values to be used to specify the "fractional" part of the pitch. The base pitch, b, is therefore:

$$b = n + \left(\frac{c_{\text{pb}} - 8192}{16384} \right) \tag{6.1}$$

where n is the pitch used in the Note On and Note Off messages, and c_{pb} the value of the pitch bend controller. In line with the usual MIDI convention, the expected frequency of the output is then $f = 440 \times 2^{\frac{b-69}{12}}$ (Section 2.4).

Note timings are based on the Note On and Note Off messages and the EG release time. As the MIDI 1.0 specification (Section 2.5.1) is targeted at real-time control, event timings are not part of the specification – the encoding of timings was therefore examined when we defined the file format for passing parameters to Kontakt (Section 6.4).

Voice	GM Program Number
Piano	1
Pipe Organ	20
Violin	41
Trumpet	57
Clarinet	72

Table 6.3: General MIDI (GM) Program Numbers for Voices used.

In order to select an instrument timbre, we adopted the General MIDI (GM) program number assignments for the voices (Section 2.5.3). By supplying the appropriate "program number", it is therefore possible to select the required timbre (Table 6.3).

MIDI provides three methods for sending arbitrary data to a synthesiser:

- Control Change messages (CCs) – for providing live performance parameters, but not for providing tone/voice parameters [MMA, 2000b, p. 9];
- System Exclusive message (sysex) – for synthesiser configuration, but not for sending real-time performance information [MMA, 2000b, p. 34];
- and Non-Registered Parameter Numbers (NRPNs) – to represent sound or performance parameters [MMA, 2000b, p. 17].

Therefore, as we are specifying sound parameters, we used NRPNs to pass the pitch trajectory parameters to the synthesiser. NRPN values are specified using a sequence of Control Change messages specifying the most significant byte (MSB) and least significant byte (LSB) for the NRPN and the MSB and LSB for the required value. As the first bit of each MIDI data byte is zero (to distinguish between status bytes and data bytes) NRPNs can be used for 14-bit values.

NRPN	Parameter
1001	LFO delay
1002	LFO frequency
1003	LFO depth

(a) LFO Parameters

NRPN	Parameter
1004	EG attack
1005	EG hold
1006	EG decay
1007	EG sustain
1008	EG release
1009	EG depth

(b) EG Parameters

Table 6.4: NRPNs used for EG and LFO parameters

The selected NRPN numbers (Table 6.4) allow the same MSB to be used for all the NRPNs, allowing the NRPNs to be sent by initially specifying the MSB and then specifying just the LSB before providing the NRPN values. In conjunction with using MIDI *running status* (in which data received is assumed to use the same status byte as preceding messages if a new status is not provided), the messages used to specify the pitch parameters are therefore:

`Bxh`	`63h`	`NNh`	*NRPN MSB*
	`62h`	`MM`$_i$h	*NRPN LSB*
	`38h`	`PP`$_i$h	*Data Entry LSB*
	`06h`	`QQ`$_i$h	*Data Entry MSB*

The initial 3 bytes are a once-per-note overhead: the status byte `Bxh` indicates that Control Change data for channel `x` will be sent and, using running status, is used for subsequent messages until a new status is received; the following 2 bytes specifying the MSB for the NRPNs. The subsequent three 2 byte data messages specify the LSB for the NRPN and enter the NRPN value as an LSB and an MSB. These are repeated for each NRPN that needs to be modified (i.e. they set NRPN $128\mathtt{NN} + \mathtt{MM}_i$ to NRPN $128\mathtt{QQ}_i + \mathtt{PP}_i$). NRPN values are therefore provided using 6 bytes per parameter plus a 3 byte overhead.

For each note, 64 bytes of MIDI messages are therefore required consisting of:

3 bytes	Pitch Bend
1 byte	Initial status byte for NRPN settings
54 bytes	Setting 9 NRPNs (one per parameter) using 6 bytes each
3 bytes	Note On
3 bytes	Pitch Off

As these settings are based on Channel messages and affect all output from the channel, they must be sent after the previous note has completed i.e. between the previous Note Off message (if any) and the Note On to which they apply.

6.4 Input File Format

We have now selected both the file format, SMF, and the necessary MIDI messages for our Expressive MIDI system (Figure 6.4).

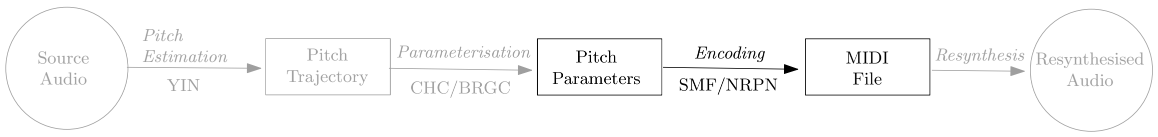


Figure 6.4: System Model: SMF/NRPN selected

We wish to provide a Standard MIDI file (SMF) (Section 2.5.1) which will play the notes from the original audio with a pitch trajectory approximating that of the original audio. As the audio considered consisted of monophonic audio (i.e. a single note played at any one time) the SMF single-track format (format 0) was used. Within an SMF, header information describes the file format and a series of MIDI messages are provided, each with a *delta time* since the preceding message. Delta times are stored as variable length quantities [MMA, 1996, p. 2]. A precision of 1ms was adopted for MIDI event timings. In order to support this resolution the "tempo" of the track was set to 1 second per quarter note and the "tick" resolution was set to 1000 TPQN – these values being unrelated to the actual musical tempo of the track and simply allowing the desired resolution for timings. Hence, delta-times were made to be the time in milliseconds since the previous MIDI event. At the start of the track, a program change message was included to select the appropriate instrument (instruments being assigned to programs according to the General MIDI standard [MMA, 1991b]).

As NRPN values are provided as 14 bit numbers and, internally, Kontakt parameters take integer values from 0 to $1,000,000$ it was necessary to scale the Kontakt parameters associated with the parameter estimates to pass them as NRPNs. Kontakt parameter values, k, were passed as NRPN value n:

$$n = \begin{cases} \left[\frac{8192k}{500000}\right] & \text{for } k \leq 500000 \\ \left[1 + 8191\frac{k}{500000}\right] & \text{for } k > 500000 \ . \end{cases} \tag{6.2}$$

Hence, the minimum, maximum and centre values (0, 1000000 and 500000) were passed through to Kontakt via NRPN values 0, 8192 and 16383. This allowed timings of 0 seconds, the minimum and maximum sustain levels of 0% and 100%, and a pitch bend of 0 pitch cents to avoid rounding errors. Within Kontakt, a script was attached to each instrument to take the MIDI NRPN values and set the internal Kontakt parameter values appropriately (Listings in Appendix A.2)

For each note, prior to the MIDI Note On event, the required NRPN values and pitch bend values were given to set the synthesiser parameters appropriately. The NRPN messages and the pitch bend message were evenly spaced across the time between the preceding Note Off event and the Note On event that the setting applied to. As the notes in the RWC database are separated by silences, this was sufficient to allow parameters to be set. With small intervals between notes, multiple MIDI channels could be used, with notes being assigned to channels in "round robin" style (the first segment being assigned to channel 1 and subsequent segments being assigned to successive channels 2 . . . 16, assigning to channel 1 again after channel 16).

In this way an SMF was produced to play notes with timings from the original audio and a pitch trajectory intended to approximate that of the original audio.

6.5 Evaluation of Expressive MIDI Resynthesis Using Kontakt

6.5.1 Method

Using the preceding format (Section 6.4) we created Standard MIDI files (SMFs) (Section 2.5.1) from the estimated parameters (Chapter 5) using Matlab 7.2. The SMFs were played back using the Kontakt 3 synthesiser and the defined voices, hosted as a VST plug-in in Max/MSP 5[3]. This allowed the audio produced by Kontakt to be captured to a file. In order to evaluate the synthesis performance, we used YIN to calculate pitch trajectories for the output audio and compared these with the expected pitch trajectories calculated from the estimated parameters. As the precise timing of start of the recording and the start of the MIDI file differed, the output audio pitch trajectories were manually aligned with the trajectories based on the pitch parameters (the timing adjustments varying from 0.04 seconds to 0.5 seconds).

6.5.2 Results

As an initial assessment of the results of the resynthesis system, we examined the output for: quiet clarinet (311CLNOP); loud staccato trumpet (211TRSTF); quiet vibrato trumpet (211TRVIP); and non-vibrato violin (151VNNVM).

We found that the clarinet notes closely matched the intended output (Figure 6.5). The pitch of the notes was, however, slightly flat (ca. 2.5 cents). Similarly, staccato trumpet notes (Figure 6.6) closely matched the intended pitch trajectories. However, during the initial attack (approx. first quarter of a second), artifacts are observed (Figure 6.7) and the some notes were truncated (e.g. the third note, examined in detail in Figure 6.8).

Vibrato trumpet notes (Figure 6.9) closely modelled the estimated vibrato, producing output of the appropriate frequency and amplitude, and similar phase to the initial estimate. Some EG+LFO estimates used both the delay and attack phases and the output pitch trajectories for these notes (e.g. the first three notes in Figure 6.9) differed from the EG+LFO pitch trajectories. These differences agree with the approximations made (Section 6.2) e.g. the second note has no delay and just an attack, the third note has just a delay and no attack. Examining the first note

[3]http://www.cycling74.com

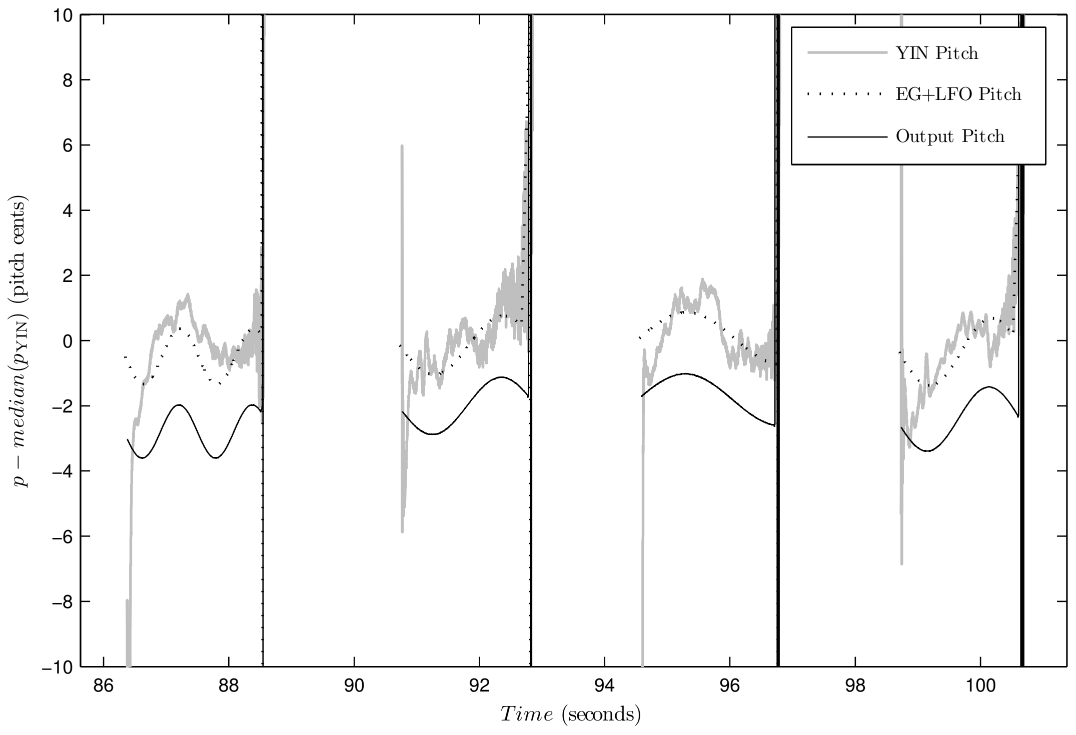


Figure 6.5: *Clarinet.* The YIN pitch of the original audio is shown as a grey line. The dotted line shows the estimated pitch trajectory according to the parameters found using CHC. The solid line shows the pitch trajectory of the audio generated from those parameters in Kontakt. The output pitch trajectories for all four notes are approximately 2.5 cents flatter than desired, otherwise they closely match the estimated pitch trajectories. This error is less than the 10 cents threshold that we hoped to achieve.

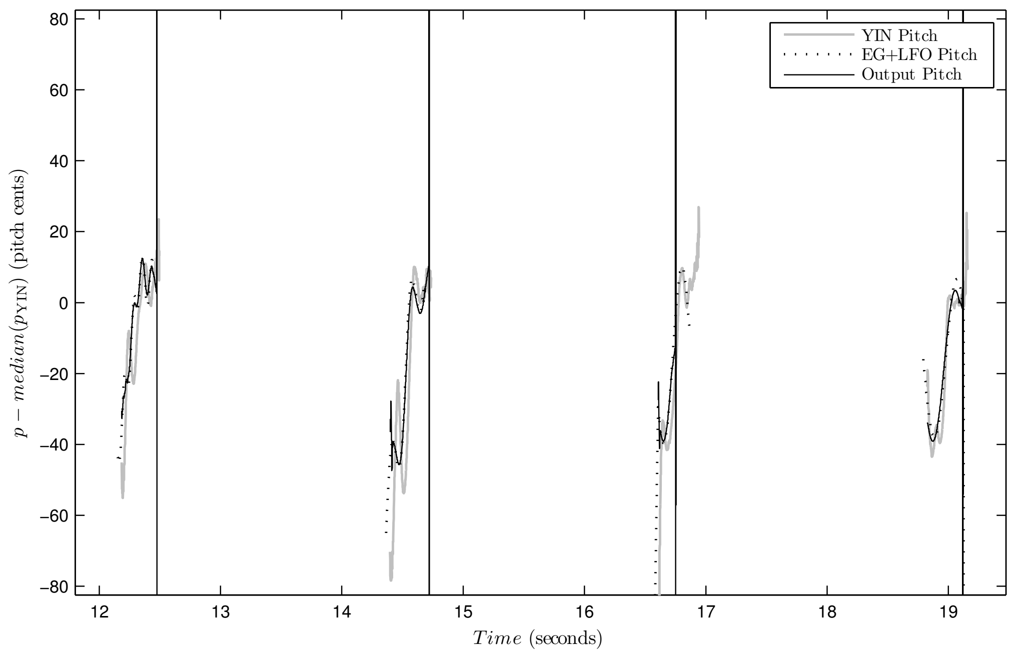


Figure 6.6: *Staccato Trumpet.* The YIN pitch of the original audio is shown as a grey line. The dotted line shows the estimated pitch trajectory according to the parameters found using CHC. The solid line shows the pitch trajectory of the audio generated from those parameters in Kontakt. Although the staccato notes have very short durations (< 0.5 seconds), the pitch trajectories are seen to resemble the expected trajectories. Although the output trajectories look to closely approximate the EG+LFO estimate, the short durations of the notes and the wide pitch variation make it difficult to asses the actual quality of the output trajectories. We therefore examined two of the notes in more detail (the first note is in Figure 6.7, the third note in Figure 6.8)

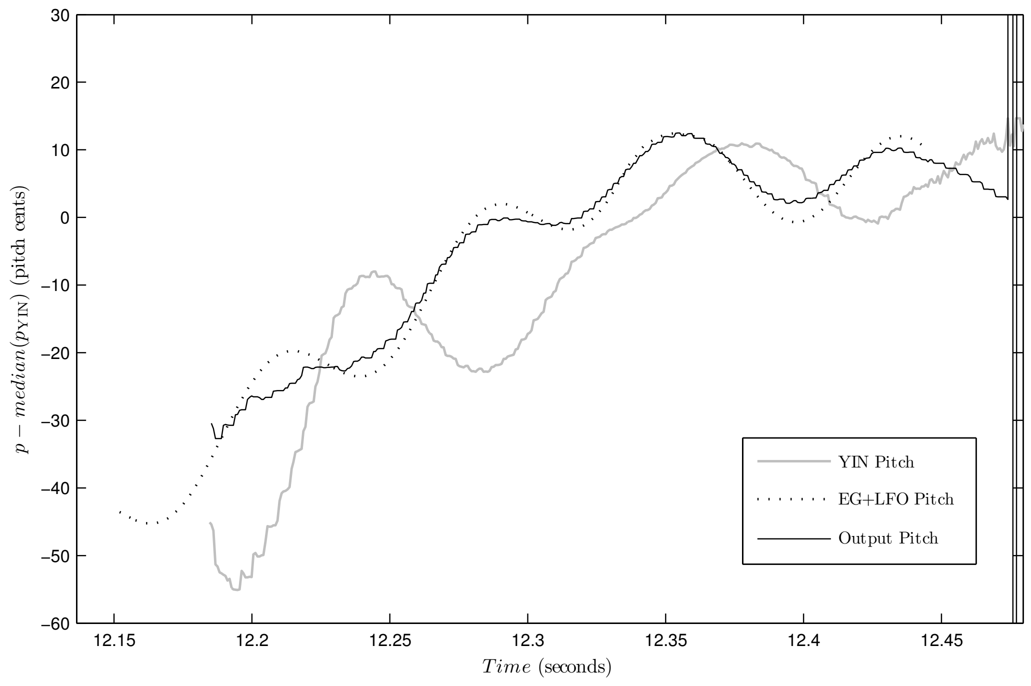


Figure 6.7: *Detail of first Staccato Trumpet note from Figure 6.6.* The YIN pitch of the original audio is shown as a grey line. The dotted line shows the estimated pitch trajectory according to the parameters found using CHC. The solid line shows the pitch trajectory of the audio generated from those parameters in Kontakt. During the first 0.05 seconds, pitch artefacts are observed above the desired pitch trajectory, after the first 0.1 seconds the trajectory closely follows that produced by the parameters. The output pitch trajectory closely matches the expected trajectory but is not aligned with the source pitch trajectory – there is a difference of approximately 0.02 seconds between the estimated and output trajectories and the source pitch trajectory. Again, output pitch trajectories are within 10 cents of the EG+LFO estimate.

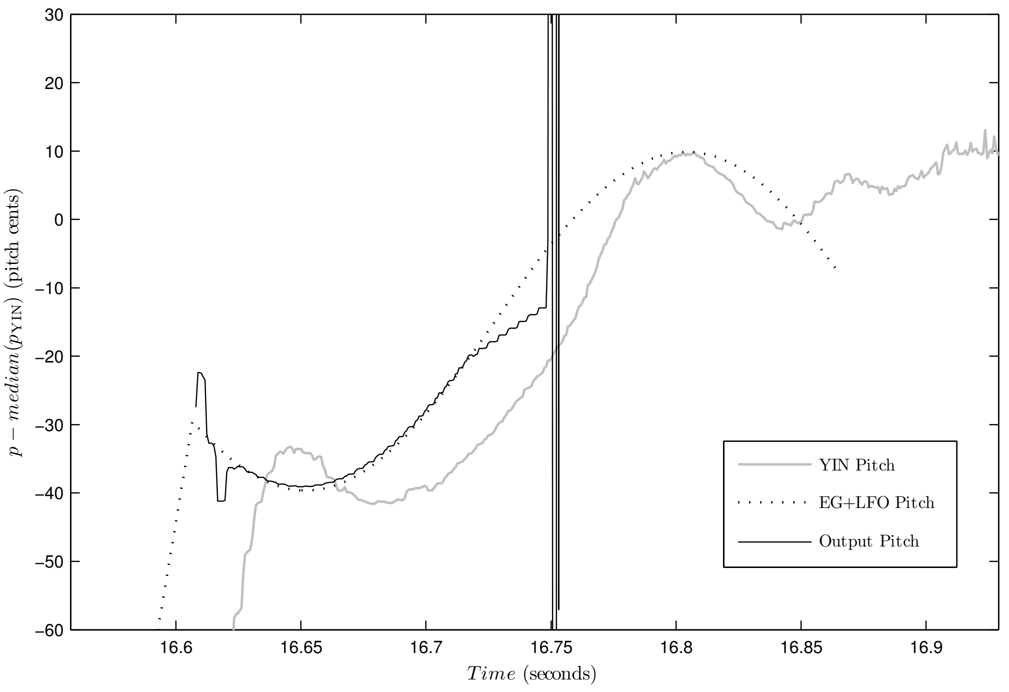


Figure 6.8: *Detail of third Staccato Trumpet note from Figure 6.6.* The YIN pitch of the original audio is shown as a grey line. The dotted line shows the estimated pitch trajectory according to the parameters found using CHC. The solid line shows the pitch trajectory of the audio generated from those parameters in Kontakt. The output audio is truncated at approximately 16.75 seconds. The output pitch trajectory otherwise closely matches the expected trajectory. Again, the output pitch trajectory and the expected trajectory are not aligned with the source pitch trajectory – with a difference of approximately 0.04 seconds. Output pitch trajectories are within 10 cents of the EG+LFO estimate.

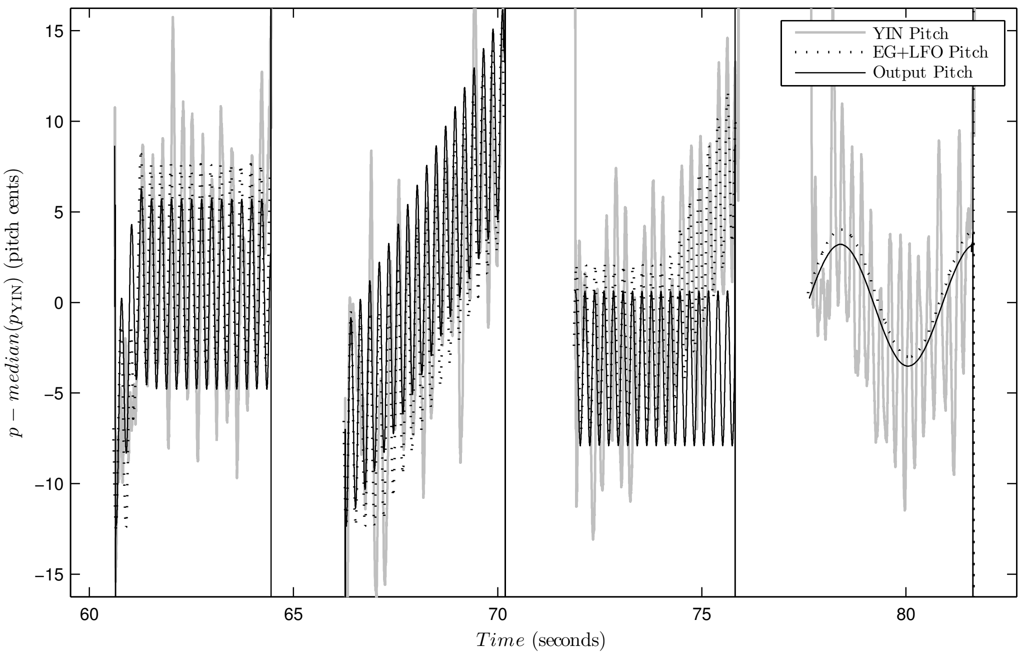


Figure 6.9: *Vibrato Trumpet.* The YIN pitch of the original audio is shown as a grey line. The dotted line shows the estimated pitch trajectory according to the parameters found using CHC. The solid line shows the pitch trajectory of the audio generated from those parameters in Kontakt. The phase, amplitude and frequency of the output vibrato closely match the EG+LFO trajectory. The second note trajectory increases across the notes as a result of the Kontakt EG not having a delay phase – the overall shape is simply a combination of a long EG attack phase and the LFO. Similarly, the long delay on the third note has led to the EG not being used. Otherwise, all four notes are within 10 cents of the estimated trajectories. We examine the first of these notes in more detail in Figure 6.10.

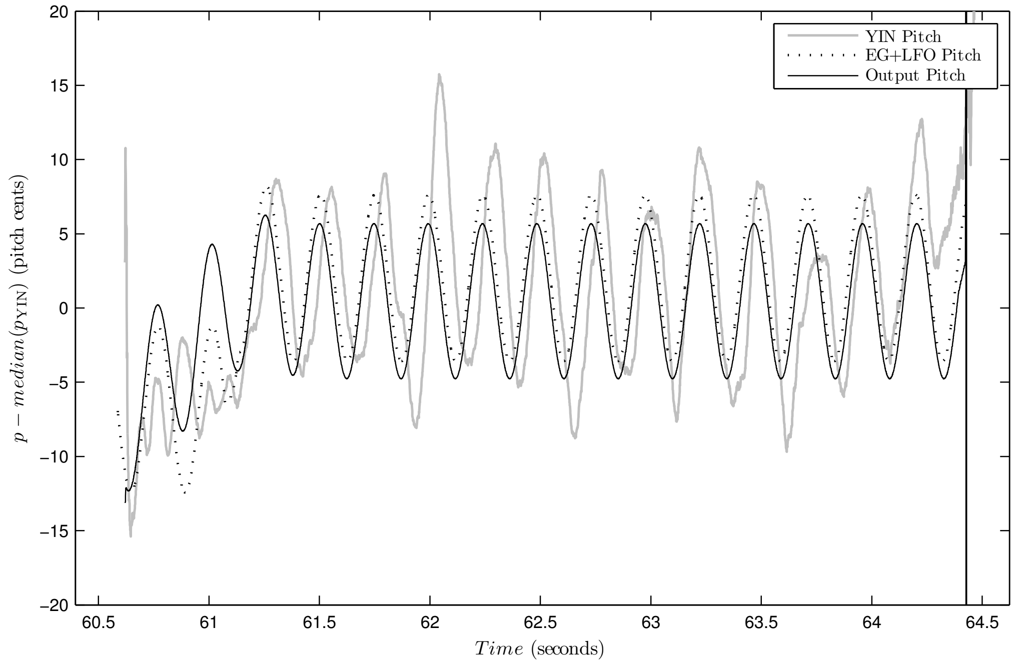


Figure 6.10: *Detail of first Vibrato Trumpet note from Figure 6.9.* The YIN pitch of the original audio is shown as a grey line. The dotted line shows the estimated pitch trajectory according to the parameters found using CHC. The solid line shows the pitch trajectory of the audio generated from those parameters in Kontakt. The amplitude and frequency of the vibrato closely match the estimated pitch trajectory. At the start of the note, the lack of a delay phase in the Kontakt EG can be seen – the output pitch trajectory simply increasing across the combined times of the delay and attack phases. After the initial attack (from approx. 61.25 seconds onwards), the output pitch trajectory is 1–2 cents flat relative to the expected trajectory.

in detail (Figure 6.10) we see that the output pitch trajectory steadily increases at 61.25 seconds rather than maintaining a level to 61 seconds and then increasing, again agreeing with the approximations made. The EG+LFO attack time for the second note was 19.59 seconds (i.e. longer than 15 seconds) and was therefore reduced to 15 seconds with the EG depth being scaled to reach an appropriate level. Generating parameters to work with the Kontakt model (i.e. based on an EG with no delay phase and with the appropriate upper limit on attack time) should allow consistency between the output audio and the expected pitch.

Non-vibrato violin notes (Figure 6.11) were close to the expected output, the four notes differing from the expected EG+LFO output by less than 5 cents. The first and fourth notes were approximately 1 cent different from the EG+LFO estimates and the third note approximately 3 cents. The second note used a delay phase for the EG for the constant level over the first two seconds followed by a negative attack for the final decrease. Approximating this using only the attack phase led to a continuous decrease across the note.

Small differences in pitch were observed for each instrument but were better than the target difference of 10 cents. Possible causes of the pitch errors include:

- incorrect playback using Kontakt;
- incorrect pitch assignments for the Kontakt voices;
- incorrect pitch estimation of the output audio.

The variety of errors observed suggest that the system for producing the output worked – i.e. that Kontakt attempted to play back the specified notes correctly – and that the errors are either issues with the Kontakt voices or the result of the pitch estimation. We therefore examined the tuning of the Kontakt voices used to see whether that could explain the pitch differences.

6.5.3 Kontakt Sample Tuning

In order to play a sample at a specific pitch p_{out}, the pitch adjustment required Δ_{out} is given by:

$$\Delta_{out} = p_{out} - (p_{base} - \Delta_{base}) \tag{6.3}$$

where p_{base} is the base MIDI pitch associated with the sample and Δ_{base} a tuning offset indicating how the actual sample pitch differs from that MIDI pitch.

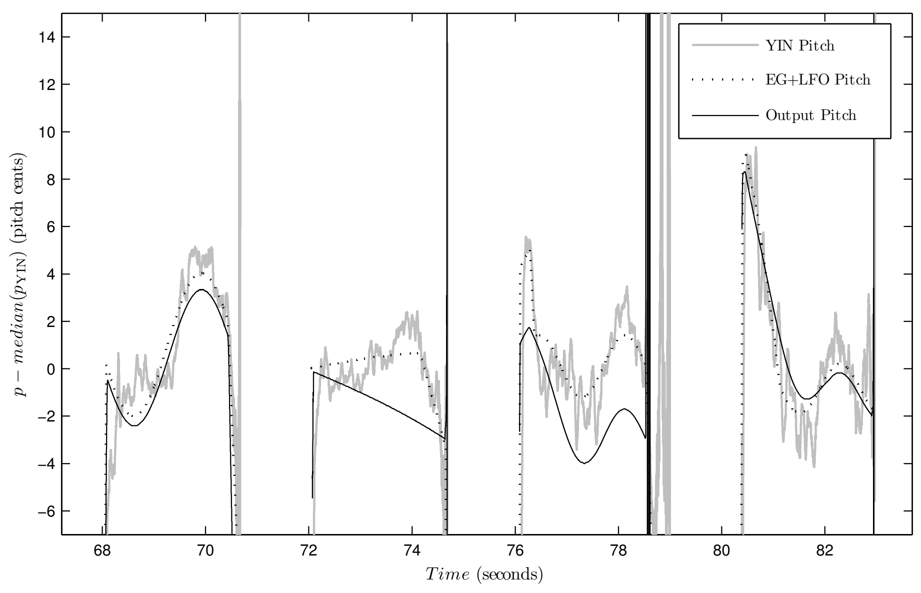


Figure 6.11: *Non-Vibrato Violin.* The YIN pitch of the original audio is shown as a grey line. The dotted line shows the estimated pitch trajectory according to the parameters found using CHC. The solid line shows the pitch trajectory of the audio generated from those parameters in Kontakt. The first and fourth notes closely matches the expected trajectory, being correct to less than 1 pitch cent. However, the third note is approximately 3 cents flatter than the estimate. For the second note, the initial increase was approximated using a slowly increasing LFO, with a negative depth EG "attack" being used to provide the drop after approximately 2 seconds. Combining the delay and attack phases has led to a decrease across the entire note. However, the overall difference between the estimated and output pitch trajectories is still only ca. 2 cents, comfortably less than the hoped for 10 cents.

Instrument	MIDI Pitch	Sample
Clarinet	70	SYNCL64
Clarinet	71–73	SYNCL70
Vibrato Trumpet	63–65	TRMPT65
Vibrato Trumpet	66	TRMPT73
Staccato Trumpet	57	TRMBN55A
Staccato Trumpet	58–60	TRMPT65
Violin	65–68	VIOLN68

Table 6.5: Notes played and the SCC1t2 samples used based on the MIDI pitch

The sample selected for a note depends upon the base pitch being played. For each note played, we examined the base MIDI pitch and found which samples were used (Table 6.5). The samples consist of start and release segments and a middle segment, consisting of a small number of cycles of the waveform (often just one cycle), that is repeated to extend the duration of the audio to the required length (a *loop* segment). For a loop segment containing n cycles, whilst the loop segment is playing, the loop length, l samples, defines the period of the sample, $\omega = \frac{l}{n}$ samples, and hence the MIDI pitch, p_0, is then:

$$p_0 = 12\log_2\left(\frac{s_\mathrm{R} n}{440 l}\right) + 69 \tag{6.4}$$

where s_R the sample rate in Hz.

Sample	Base pitch p_{base} (MIDI Pitch)	Pitch offset Δ_{base} (MIDI Pitch)	Source pitch $p_{base} - \Delta_{base}$ (MIDI Pitch)	Loop Start (sample #)	Loop End (sample #)	Loop Length l (samples)	Pitch p_0 (MIDI Pitch)
SYNCL64	64	+0.28	63.72	70	138	68	63.72
SYNCL70	70	-0.11	70.11	48	95	47	70.11
TRMBN55A	55	+0.08	54.92	2251	2364	113	54.92
TRMPT65	53	+0.10	52.90	1275	1402	127	52.90
TRMPT73	61	+0.10	60.90	1210	1290	80	60.90
VIOLN68	56	-0.03	56.03	1341	1447	106	56.03

Table 6.6: Samples used, their base MIDI pitch p_{base} and tuning offsets Δ_{base}, and the loop-based pitch of the underlying sample p_0 (see equation 6.4).

Table 6.6 lists the individual clarinet, trumpet and violin samples from the SCC1t2 SoundFont that were used to resynthesise the audio. The tuning offsets, Δ_{base}, are seen to adjust the tuning accurately to agree with the specified base MIDI pitch e.g. if we required sample TRMBN55A to be played at pitch $p_{out} = 54.92$ then it would need no pitch adjustment as, from (6.3), $\Delta_{out} = 54.92 - (55 - 0.08) = 0$. The pitch offsets are therefore not caused by the tunings of the samples and must, therefore, be produced either by YIN or by Kontakt itself.

6.6 Conclusions

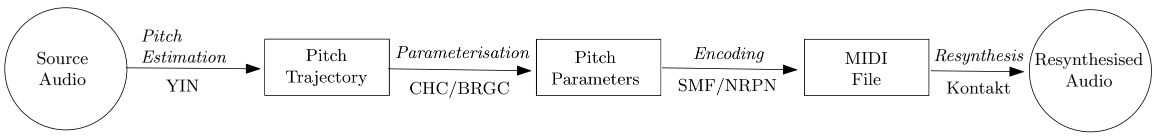


Figure 6.12: The Complete System Model

In Chapters 3 to 6, we have derived an Expressive MIDI system (Figure 6.12) to transmit a "MIDI sketch" from audio. This divides the audio into a series of notes and represents the pitch variations within notes using envelope generators (EGs) and low frequency oscillators (LFOs). Having extracted segments of pitch trajectories from the audio (Chapter 3) we estimated parameters for EGs and LFOs using a CHC (Chapter 5), and have shown that it is possible to render audio using those parameters (this chapter). However, the current output sometimes truncates notes, as for the staccato trumpet, and exhibits small pitch errors of less than 10 cents. Improving the segmentation algorithm (Section 3.3) and including constraints on the nature of the pitch trajectory (e.g. limiting the trajectory to start and end at the same pitch) may solve truncation errors. Additionally, errors in the YIN pitch trajectories (e.g. octave errors) need to be solved in order for valid EG+LFO pitch parameters to be created for resynthesis.

The pitch is modelled as base pitch plus EG plus LFO, and the EG and LFO parameters and the offset from an exact MIDI pitch are specified using 14 bit NRPN values – the MMA specifying that NRPNs should be used for sound and performance parameters (Section 6.3). Additionally, note event timings were restricted to a precision of 0.001 seconds. These quality constraints apply to all outputs from the MIDI file. However, the observed output closely matched the expected pitch trajectories based on the parameter values.

Although Kontakt was the closest match to the DLS model, Kontakt's synthesis model is not the same as DLS (e.g. the EG lacks a delay, the maximum EG attack duration is shorter and the LFO supports a "fade" rather than a delay). The resynthesis model therefore differs from that used in the CHC analysis stage. The analysis stage can be updated to agree with the actual resynthesis stage i.e. using CHC to produce AhDSR EG parameters, and to include the fade parameter for the LFO rather than the delay. Additionally, the parameters are represented as 14-bit NRPN values and the relationship between those values and the Kontakt settings is known. The representation of the parameters in CHC can therefore be reduced from 32-bits, and can optimise Kontakt parameter values directly. This reduction in dimensionality will reduce the size of the search space and should improve CHC performance. Applying these changes should match analysis results more precisely with the final Kontakt output.

Complex samples are often used with samplers to create a more realistic sound. However, repetitive use of these samples reveals an underlying lack of expressiveness as each note sounds the same. Adding expressive control to samples offers a sound with more character. In this resynthesis work, small samples were used with little character of their own rather than using longer "natural" sounding samples for the notes. More complex samples may produce "better" sounding audio. However, if the original samples have their own pitch trajectories these will affect the output and the pitch will not match that of the source material. In order to use samples with more complex timbral features, the pitch trajectory would need to be removed from them – this should be possible by pitch shifting the sample to a fixed pitch before importing it to Kontakt, but this process may introduce additional artefacts.

The close match between the output pitch trajectories produced using Kontakt and the expected trajectories based on the EG+LFO parameters give confidence that, by estimating parameters that match the Kontakt model, we can produce audio from Expressive MIDI which closely matches the parameterised pitch trajectory. Differences between the EG+LFO model used by Expressive MIDI and the model used in Kontakt can be overcome by adopting the Kontakt model in Expressive MIDI, and improved segmentation should remove some timing artifacts from the system. Alternatively, a new synthesiser could be created to match the Expressive MIDI model, however this "bottom up" approach to the problem does not fit with the original intention of using existing components.

Expressive MIDI produced a low bit-rate representation of the audio data. An 11Mb uncompressed audio file consisting of 31 trumpet notes played over approximately 2 minutes produced a MIDI file of 2533 bytes[4] (which could be further compressed, a ZIP archive of the file being 1458 bytes). Using the readily available "`lame`"[5] MP3 converter to compress the audio as a 32kbit/s MP3 file produced a file of 505kB – 190kbit/s MP3 being regarded as approximating CD quality audio. The SMF size would be expected to be similar for 31 notes over any period of time (the only difference being the lengths of the delta times which would decrease if the notes

[4]64 bytes of data per note accounting for 1954 bytes, plus the overhead for the MIDI file header information, plus variable length delta-times for each MIDI message

[5]http://lame.sourceforge.net

were closer together). This suggests that it would require at least 190 notes to be played per minute for the 32kbit/s MP3 to result in a smaller file.

Our aim was to use Expressive MIDI to resynthesise audio based on the pitch trajectory of source audio material. However, resynthesis from Expressive MIDI only depends on the pitch trajectory – it is therefore possible to use this system for other purposes:

- As the final sample rate and the precision of the EG and LFO are features of the resynthesis engine, analysing low sample-rate audio can allow high resolution audio to be produced during the rendering;
- the pitch trajectory from a recorded performance can be used to control resynthesis using a different instrument timbre (i.e. using samples from an instrument);
- as Expressive MIDI uses high-level parameters with semantic meaning, it is possible to modify the sound by adjusting the Expressive MIDI parameters (e.g. to reduce vibrato).

Small pitch differences observed between the output audio and the expected values could either be: actual differences between the pitch of the Kontakt output and the expected pitch; or artifacts produced by YIN. In order to test whether the Kontakt output is producing the expected pitch, we need to estimate the pitch of the output. Whichever is the actual cause of the pitch difference, Kontakt or YIN, we therefore need to assess the quality of pitch estimation. In the next chapter, we examine the performance of the YIN pitch estimation algorithm in detail.

Chapter 7

(Re)Evaluating YIN

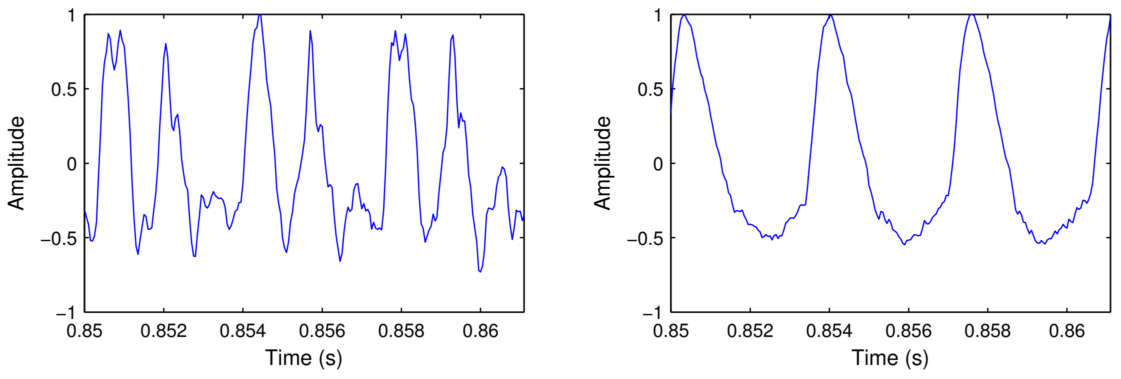
In chapters 3 to 6, we produced a system for extracting pitch trajectories from audio and resynthesising audio based on those pitch trajectories. However, differences were observed between the parameterised pitch trajectories and the pitch estimates extracted from the resynthesised audio. Additionally, we saw evidence of harmonic errors in the YIN pitch estimates (Chapter 5). The question arises of how many observed pitch differences are a result of the resynthesis and how many are artifacts of the pitch estimation process. In order to answer this question, we present a novel detailed analysis of the performance of the YIN pitch estimation algorithm.

In order to produce an accurate, high resolution pitch track from audio, we need to be able to extract accurate, high-resolution pitch trajectories from the original audio. To do this, we require an accurate, high resolution pitch estimator. The JND for two successive pure tones is ca. 10 cents (Section 3.2). In order to synthesise audio with the same perceived pitch, a pitch error of less than 10 cents is therefore required – pitch differences may be noticeable below that threshold, but differences greater than that will be noticeable.

7.1 Evaluating Pitch Estimators

7.1.1 Pitch Data

In order to evaluate the performance of pitch estimation algorithms, it is necessary to have a suitable corpus of audio labelled with ground-truth pitch values. For vocal data, it is possible to simultaneously record both audio and electroglottograph (EGG, also known as a laryngograph) data. Several such databases are readily available online; some of them including pitch estimates created by their authors (e.g. see Section 7.1.2). When pitch estimates are not provided, pitch estimation can be performed based on EGG data.



(a) Audio (b) EGG

Figure 7.1: Example of EGG and audio data

An EGG measures the electrical resistance across the larynx, and this is related to the contact area between the vocal folds. EGG data therefore indicates the frequency of the opening and closing of the larynx. EGG data is a relatively simple signal compared with the audio as it excludes the resonances associated with the chest, mouth and nasal cavities. Examining example audio and EGG data for a speech sample (Figure 7.1), the audio shows a strong third harmonic (periodicity at three times the frequency, visible as two large peaks and a small peak in each cycle e.g. from 0.85 to 0.854 seconds). However, the EGG signal clearly shows the underlying frequency at the larynx. Glottal closure instants (GCIs) – the moments when the laryngeal folds close – provide the highest rate of change of the EGG signal [Henrich et al., 2004]. Estimating GCIs from the EGG data allows a ground-truth pitch estimate to be calculated – each consecutive pair of GCIs defining one period of the oscillation of the larynx. The number of pitch estimates thus calculated is therefore defined by the number of GCIs in the dataset.

We next introduce a selection of available pitch databases which we will use to evaluate the performance of YIN.

Dataset	Content Type	Audio data	EGG data	Pitch Supplied?	MIDI Pitch Range
Bagshaw (FDA) [Bagshaw et al., 1993]	Speech	20kHz 16bits	20kHz 12bits	✓	34.58 to 67.35
Keele [Plante et al., 1995]	Speech	20kHz 16bits	20kHz 16bits	✓	32.05 to 67.35
VOQUAL'03					
Brian (English)	Speech	44.1kHz 16bits	44.1kHz 16bits	✗	33.63 to 73.93
Singing 1 (English)	"Belting" Singing	24kHz 16bits	24kHz 16bits	✗	51.94 to 80.54
Singing 2 (French) [VOQUAL'03, 2003]	Speech, Shout and Singing	44.1kHz 16bits	44.1kHz 16bits	✗	40.58 to 71.45
MIREX QBSH [Jang, 2010]	Singing and Humming	8kHz 8bits	✗	✓	30.20 to 85.52

Table 7.1: Readily Available Pitch Datasets

7.1.2 Pitch Databases

Bagshaw

The Bagshaw speech database [Bagshaw et al., 1993] contains 100 audio files, 50 each from one male and one female speaker, totalling 5.5 minutes. Pitch estimates are included with the database, estimated from laryngograph (EGG) data – giving $24,246$ pitch estimates between pairs of glottal closure instants (GCIs). Ground-truth pitch estimates are provided in terms of the period of the EGG signal to the nearest sample.

The Bagshaw database is available from: http://www.cstr.ed.ac.uk/research/projects/fda/.

Keele

The Keele pitch database [Plante et al., 1995] contains an audio file for each of 10 speakers (5 male, 5 female) with EGG data. Pitch estimates based on applying autocorrelation to the EGG (with a 10ms hop-size and 25.6ms window size) are included with the database. In total, 5 minutes of audio are available, and $16,960$ pitch estimates are supplied (one for each voiced frame).

The Keele database is available from: http://www.liv.ac.uk/psychology/hmp/projects/pitch.html.

VOQUAL'03

For the 2003 ISCA workshop on "Voice Quality: Functions, Analysis and Synthesis" [VOQUAL'03, 2003]), five databases were created – of American English, Japanese and French speech, and of English and French singing. The American English speech database and the singing databases include EGG data but have no pre-calculated pitch estimates provided. The datasets with EGG data total 4.5 minutes. Using the method for estimating GCIs given by Henrich et al. [2004] resulted in $20,533$ period estimates (to the nearest sample) from the EGG data.

The VOQUAL'03 database is available from: `http://archives.limsi.fr/VOQUAL/voicematerial.html`.

MIREX QBSH

Query By Singing And Humming (QBSH) systems attempt to identify music based on a low bit-rate sung or hummed segment. Since 2006, the Music Information Retrieval Evaluation eXchange (MIREX), has been evaluating the performance of QBSH systems. As part of this evaluation, a corpus of annotated audio has been created [Jang, 2010]. Using 48 example pieces of music, some $4,431$ recordings have been created by about 195 subjects, with individual pieces being recorded from 10 to 170 times. The resulting audio files have all been annotated with manually labelled pitch estimates, albeit with no guarantee of the quality of these estimates. In total, this database provides 10 hours of annotated audio, with some $800,000$ pitch annotations. Although it is the largest corpus in terms of number of files, total file size and duration, as it is intended for low bit-rate applications, it is only recorded at 8kHz and 8 bits.

The MIREX QBSH corpus is available from: `http://neural.cs.nthu.edu.tw/jang/`.

7.1.3 Metrics for Evaluating Pitch Estimators

(i) Gross Error Percentage (GEP)

The Gross Error Percentage (GEP) (sometimes referred to as the Gross Error Rate (GER)) is used in many pitch estimation papers as the overall measure of the quality of the algorithm. It is the percentage of test points at which the algorithm gave a frequency estimate within a specified percentage of the ground-truth value. The specified percentage is usually 20% [Bagshaw et al., 1993, de Cheveigné and Kawahara, 2002, Camacho and Harris, 2008, Signol et al., 2008].

Denoting the frequency estimate as f^*, the ground-truth frequency as f_0 and the associated pitches as p^* and p_0, the relative error is then:

$$\epsilon = \frac{\Delta f}{f_0} \tag{7.1}$$

where $\Delta f = f^* - f_0$, and the pitch error is:

$$p^* - p_0 = 12 \log_2(1 + \epsilon) \ . \tag{7.2}$$

Hence, with a GEP of 20%, the pitch range is then $12 \log_2 0.8 \leq p^* - p_0 \leq 12 \log_2 1.2$ i.e. $-3.87 < p^* - p_0 < 3.16$. The Gross Error Percentage with a 20% threshold therefore treats pitch errors of 3 semitones as "correct". Pitch errors of greater than 10 cents (approx. the JND for pure tones) will, however, be noticeable to a listener. Error thresholds other than the GEP therefore need to be considered.

(ii) 10 Cent Threshold (10c)

The JND for pure tones is less than 10 cents, and the JND for complex tones is less than that for pure tones. Any pitch estimates which are out by more than 10 cents will therefore have a noticeably different pitch to the pitch of the original audio (for a listener with normal hearing). The proportion of pitch errors above a threshold of 10 cents will provide a measure of the noticeable difference between the pitch estimates and the ground-truth pitch of the source – there will also be noticeable differences in pitch at less than 10 cents difference, but the 10 cents threshold will provide more information than the GEP.

(iii) Period Mismatch Percentage (PMP)

The Keele [Plante et al., 1995] and Bagshaw [Bagshaw et al., 1993] pitch databases include EGG data and period estimates estimated directly from the EGG waveform to the nearest sample. Similarly, the manual pitch labels for the MIREX QBSH dataset [Jang, 2010] are provided based on estimates of the period at sample-level accuracy. From this "ground-truth" data, the best performance we can hope to achieve is to match the performance of the "ground-truth" estimation – i.e. to estimate the period to the nearest sample. The proportion of period estimates, rounded to the nearest sample, that agree with the provided ground-truth periods provides an estimate of how well a pitch estimation algorithm performs relative to the ground-truth pitch estimation. We refer to this metric as the Period Mismatch Percentage (PMP)

Dataset	Sample Rate	Frequency Range	Pitch Accuracy (cents)
Bagshaw	20kHz	60 Hz to 400 Hz	5.19 to 34.63
Keele	20kHz	52 Hz to 400 Hz	4.50 to 34.63
VOQUAL'03	44.1kHz	57 Hz to 585 Hz	2.24 to 22.97
VOQUAL'03	24kHz	164 Hz to 858 Hz	11.83 to 61.9
MIREX QBSH	8kHz	46 Hz to 1.14 kHz	9.95 to 247.75

Table 7.2: Accuracy of pitch dataset ground-truth based on sample rate and frequency range

The accuracy of this in terms of *frequency* depends upon the sample rate of the underlying data and the ground-truth pitch estimate (a one sample change in period is a small change to a slowly varying signal, a large change at high frequencies).

With a sample rate of s_{R}, the pitch, p, for a given period, τ, is given by:

$$p = 12\log_2\left(\frac{s_{\mathrm{R}}}{440 \times \tau}\right) + 69 \ . \tag{7.3}$$

The change in pitch for a change to the period of ± 0.5 samples is therefore:

$$\Delta p = 12\log_2\left(\frac{\tau + 0.5}{\tau - 0.5}\right) \tag{7.4}$$

and, as frequency $f = \frac{s_{\mathrm{R}}}{\tau}$,

$$\Delta p = 12\log_2\left(\frac{s_{\mathrm{R}} + 0.5f}{s_{\mathrm{R}} - 0.5f}\right) \ . \tag{7.5}$$

Considering the range of frequencies in each dataset and the sampling frequencies, equation (7.5) indicates that the precision of sample resolution period estimation varies with the sample rate (Table 7.2). Hence, depending upon the dataset being considered, the 10c metric may be more precise than the precision supported by the ground-truth pitch estimates. However, a low PMP should be possible given ground-truth pitch estimates accurate to the nearest sample. Given the low sample rate and the high maximum pitch estimate, the QBSH dataset has the least precise pitch estimates.

(iv) MIDI Mismatch Percentage (MMP)

Another application for pitch estimation is transcription of a musical score from audio. For this purpose, less time resolution is typically required than for our resynthesis as a single pitch is assigned to each note. For Western music, this is typically based on the 12-tone equal-temperament scale, i.e. to integer MIDI pitch values. Comparing pitch estimates and ground-truth pitch values rounded to the nearest MIDI pitch gives an indicator of how well this task can be performed. The proportion of MIDI pitch estimates that do not match the MIDI ground-truth pitch offers a pitch estimation performance metric between the coarse GEP and the fine detail of matching the period to the nearest sample. We refer to this metric as the MIDI Mismatch Percentage (MMP).

7.2 Appraising YIN

YIN is a pitch estimation algorithm inspired by autocorrelation techniques (introduced in Chapter 3). YIN estimates the pitch to minimise the square-error difference function between a window of audio and a window a period later. It is a time-domain pitch estimation algorithm. Sub-sample pitch estimates are made by quadratic interpolation of the difference function values. YIN provides three outputs: pitch estimates; power estimates; and aperiodicity values. The aperiodicity is the average square error across a single period window.

7.2.1 YIN Test Implementation

We have pitch estimates with timings for each dataset (provided by the authors for the Bagshaw, Keele and QBSH data and estimated from the EGG data using Henrich's technique for the VOQUAL'03 data). These were taken as ground-truth estimates of pitch, and the assumption was made that the pitch estimates provided were associated with pitched parts of the audio signal. The default range of pitch estimates output by YIN is from 30 Hz to a quarter of the sample rate. Considering the sample rates of each database, the ground-truth pitch estimates for all four databases are within this pitch range.

YIN pitch estimates were linearly interpolated to estimate the pitch at the same timings as the ground-truth data i.e. pitch estimates were only tested where there was a ground-truth pitch. We relied upon the ground-truth data for "pitched"/"unpitched" decisions – where such information was provided it was used, otherwise we assumed that all pitch estimates provided related to pitched parts of the signal. No attempt was made to make additional "pitched" / "unpitched" decisions using the aperiodicity and power outputs from YIN. In order to allow tests to be rerun easily, pitch estimates

	Bagshaw	Keele	VOQUAL'03	MIREX QBSH	Notes
10c	75.25%	62.37%	36.82%	53.34%	*% Pitch errors above 10 cents – lower is better*
GEP	3.37%	4.47%	3.01%	3.68%	*% frequency errors above 20% of frequency – lower is better*
PMP	77.13%	70.50%	46.33%	17.74%	*% Period mismatches – lower is better*
MMP	33.74%	26.87	18.81%	16.30%	*% MIDI pitch mismatches – lower is better*

Table 7.3: Summary pitch metrics (10 cents threshold, Gross Error Percentage, Period Mismatch Percentage and MIDI Mismatch Percentage) for YIN on the Bagshaw, Keele, VOQUAL'03 and MIREX QBSH datasets.

were saved to files and were reloaded when appropriate rather than recalculated.

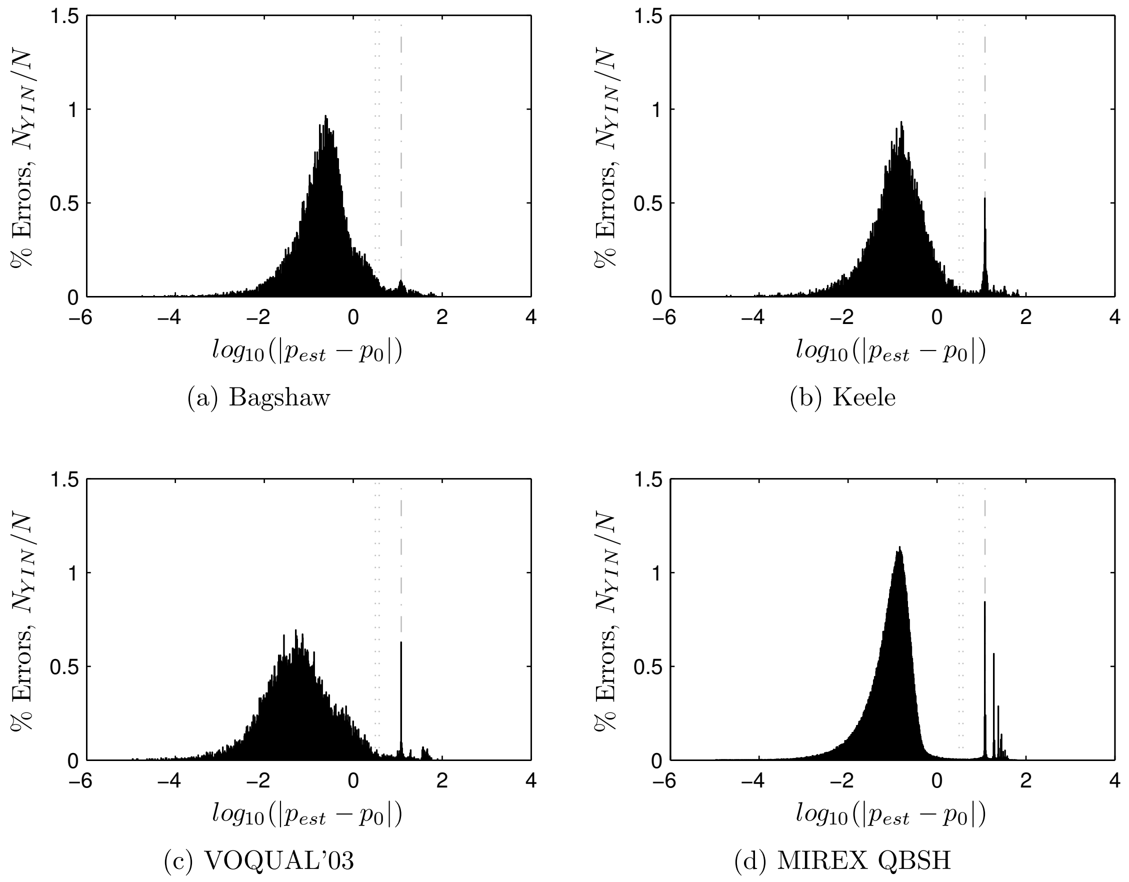
7.2.2 YIN Pitch Metric Results

Examining the summary pitch metrics for YIN on the four datasets (Table 7.3) we find that the Gross Error Percentage (GEP) for each dataset is similar (3.0-4.5% in each case) – YIN apparently working best on the VOQUAL'03 dataset with a GEP of 3.01% and worst on the Keele dataset with a GEP of 4.47%. However, the alternative metrics present different pictures of the results: the "10 cent" pitch threshold also suggesting YIN worked best on the VOQUAL'03, but indicating that the worst performance was on the Bagshaw dataset; the Period and MIDI metrics indicating that YIN performed best on the QBSH data and worst on the Bagshaw dataset. From this, we see that a simple fixed metric gives a partial picture of the performance of a pitch estimator and that pitch estimator performance varies across thresholds.

In the next section, we therefore consider the distribution of errors produced by YIN.

7.2.3 YIN Pitch Error Distributions

10c, GEP, PMP and MMP give snapshots of the performance of the algorithms at specific error thresholds. In order to examine the performance across error thresholds, histograms were created based on the logarithm of the error value, $\log_{10}(|p_{est} - p_0|)$, at an arbitrary small interval of 0.05.



(a) Bagshaw (b) Keele (c) VOQUAL’03 (d) MIREX QBSH

Figure 7.2: Distributions of YIN pitch errors for the Bagshaw, Keele, VOQUAL’03 and MIREX QBSH datasets. Note x axis is $\log_{10}|p_{\text{est}} - p_0|$. Dotted lines indicate the error rates for the up/down GEP. Dash-dot line indicates octave errors ($|p_{\text{est}} - p_0| = 12$)

Using the Bagshaw, Keele and VOQUAL’03 datasets, error histograms were produced for YIN (Figure 7.2). YIN shows a significant number of octave errors (the spike at the dash-dot line indicating errors of approximately 12 semitones), on the Keele, VOQUAL’03 and MIREX QBSH datasets. The dotted lines in the figure show the two thresholds for the 20% GEP, for errors below and above the ground-truth pitch. Most of the errors that YIN shows above this threshold are octave errors ($f_{est} = 2f_0$ or $f_{est} = \frac{1}{2}f_0$) (the dash-dot line in the figure). Excluding the octave errors, YIN performed well on the Keele and VOQUAL'03 databases, with most errors being less than 10^{-1}. YIN analysis of the Bagshaw database produced fewest octave errors, but most errors were between 10^{-1} and 10^{0}. The MIREX QBSH dataset is largest with the largest number of individual files and the longest overall duration and most YIN errors were around 10^{-1}. As well as octave errors, additional harmonic errors appear in the distribution (peaks to the right of the dash-dot line).

7.2.4 Harmonic Errors and YIN

Having examined the magnitude of pitch errors, it is also of interest to know whether YIN gives frequencies which are too high or too low. We therefore examined the error distributions against the relative frequency of the data $\frac{f_{est}}{f_0}$ (Figure 7.3). Some harmonic errors are observed for each database. However, the MIREX QBSH database produces a wide range of harmonic "down" errors in which the estimated period is a multiple of the ground-truth – these errors causing the spikes to the right of the dash-dot line in Figure 7.2.

A significant number of the larger errors are around the harmonics, and the output of YIN could be improved if these harmonic errors could be removed. We next look into how much performance could be improved if a technique was found to remove the harmonic errors.

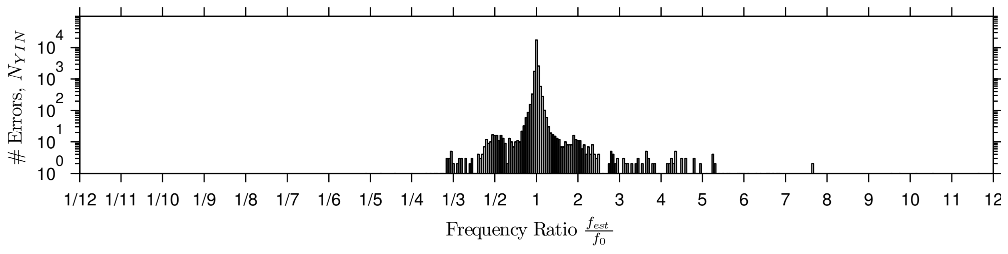
7.2.5 Harmonic vs. "Other" Errors

We have observed that harmonic errors occur whilst using YIN, both in testing against the test databases (Figure 7.3) and previously (Chapter 5) against the RWC data (Figure 5.12). However, if the only errors were harmonic errors then a series of distinct spikes should be apparent at the harmonic ratios. As we observe errors between the harmonic ratios other errors must also occur.

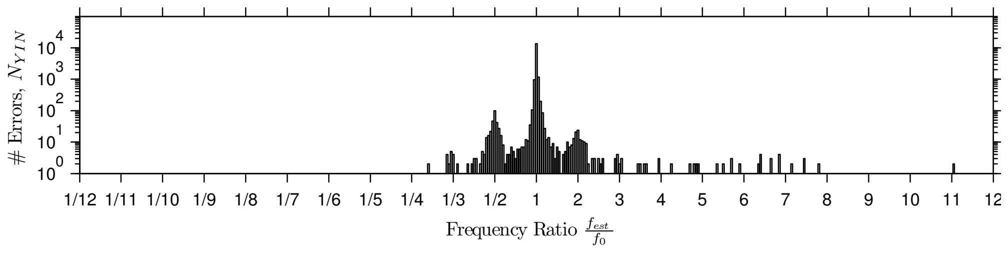
For harmonic "up" errors, the estimated frequency, f, is based on a positive integer multiple, the harmonic factor $k \in \mathbb{N}$, of the ground-truth frequency, f^*, with some other error factor, ϵ, accounting for the residual difference between the estimate and the ground-truth:

$$f = k \times f^* \times \epsilon \tag{7.6}$$

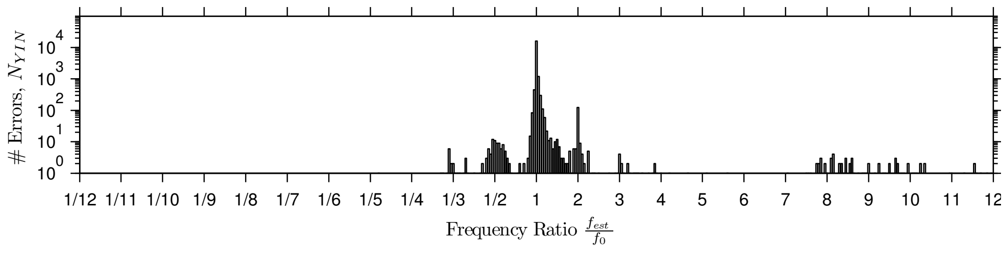
and $f \geq f^*$. The best pitch estimation performance that could occur after removing the harmonic error will minimise the effect of ϵ on the estimated frequency – this



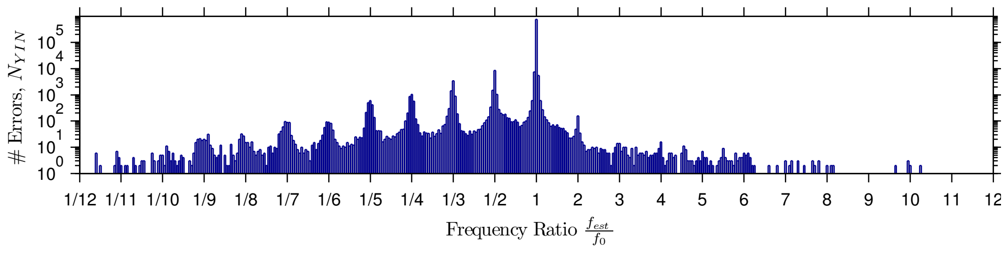
(a) Bagshaw



(b) Keele



(c) VOQUAL'03



(d) MIREX QBSH

Figure 7.3: YIN error distribution vs. relative frequency $\frac{f_{est}}{f_0}$ on the Bagshaw, Keele, VOQUAL'03 and MIREX QBSH datasets.

level of improvement may not be possible, but will provide an upper bound on the performance that can be achieved by removing harmonic errors from the estimator output. We therefore require ϵ, the residual error factor to be as close as possible to 1. From (7.6), $\epsilon = \frac{f}{kf^*}$ and the candidate values of ϵ that are closest to 1 (i.e. produce the smallest variation from the ground-truth) are given when $k = \left\lfloor \frac{f}{f^*} \right\rfloor$ and when $k = \left\lceil \frac{f}{f^*} \right\rceil$ (as $k \in \mathbb{N}$).

In order to minimise the error based on a perceptual scale (i.e. to minimise how audible the errors are), we consider the squared pitch error after removing the effect of the harmonic error, k:

$$\begin{aligned} E(k) &= (\hat{p} - p^*)^2 \\ &= \left(12 \log_2 \left(\frac{f}{kf^*}\right)\right)^2 \end{aligned} \tag{7.7}$$

as the underlying pitch $p^* = 12 \log_2 \left(\frac{f^*}{440}\right) + 69$ and the estimated pitch after harmonic error removal is $\hat{p} = 12 \log_2 \left(\frac{f}{440k}\right) + 69$.

Theorem 1 *If $f \geq f^* > 0$ then $\frac{f}{f^*} \leq \sqrt{\left\lfloor \frac{f}{f^*} \right\rfloor \left(\left\lfloor \frac{f}{f^*} \right\rfloor + 1\right)}$ implies that $E\left(\left\lfloor \frac{f}{f^*} \right\rfloor\right) \leq E\left(\left\lceil \frac{f}{f^*} \right\rceil\right)$ where $E(k) = \left(12 \log_2 \left(\frac{f}{kf^*}\right)\right)^2$*

Proof 1 *In order to choose the value for k that minimises this function, we can compare $E(k)$ for the two candidate solutions, . Let $\eta = \frac{f}{f^*}$, the candidate values for k are then $\lfloor \eta \rfloor$ and $\lceil \eta \rceil$. Then, if $\eta \notin \mathbb{I}$,*

$$\begin{aligned} E\left(\lfloor \eta \rfloor\right) &= \left(12 \log_2 \frac{\eta}{\lfloor \eta \rfloor}\right)^2 \\ E\left(\lceil \eta \rceil\right) &= \left(12 \log_2 \frac{\eta}{\lceil \eta \rceil}\right)^2 \quad . \end{aligned} \tag{7.8}$$

For $\eta \notin \mathbb{I}$, $1 < \lfloor \eta \rfloor < \eta < \lceil \eta \rceil$. Hence, as $a^2 \leq b^2$ implies $a \leq b$ if $a, b \geq 0$, $E(\lfloor \eta \rfloor) \leq E(\lceil \eta \rceil)$ implies

$$\log_2 \frac{\eta}{\lfloor \eta \rfloor} \leq \log_2 \frac{\lceil \eta \rceil}{\eta} \quad . \tag{7.9}$$

As $\eta \notin \mathbb{I}$, $\lceil \eta \rceil = \lfloor \eta \rfloor + 1$ and, as $\log(a) \leq \log(b)$ implies $a \leq b$,

$$\eta^2 \leq \lfloor \eta \rfloor \left(\lfloor \eta \rfloor + 1\right) \quad . \tag{7.10}$$

As $f \geq f^* > 0$, we have $1 \leq \eta \leq \sqrt{\lfloor \eta \rfloor \left(\lfloor \eta \rfloor + 1\right)}$ ∎

Hence, for harmonic "up" errors, such that $f \geq f^* > 0$, the harmonic factor, k, that gives the residual error factor closest to 1 is

$$k = \begin{cases} \left\lfloor \frac{f}{f^*} \right\rfloor & \text{if } \frac{f}{f^*} \leq \sqrt{\left\lfloor \frac{f}{f^*} \right\rfloor \left(\left\lfloor \frac{f}{f^*} \right\rfloor + 1\right)} \\ \left\lceil \frac{f}{f^*} \right\rceil & \text{otherwise.} \end{cases} \tag{7.11}$$

Similarly, for "too low" harmonic errors, the pitch estimate can be considered to include a harmonic error factor $k = \frac{1}{k^\dagger}$, $k^\dagger \in \mathbb{N}$, and a residual error factor ϵ such that

$$f = \frac{1}{k^\dagger} \times f^* \times \epsilon \ . \tag{7.12}$$

As $\epsilon = k^\dagger \frac{f}{f^*}$, the values of ϵ that are closest to 1 (i.e. produce the smallest variation from the ground-truth) are given when $k^\dagger = \left\lceil \frac{f^*}{f} \right\rceil$ and when $k^\dagger = \left\lfloor \frac{f^*}{f} \right\rfloor$. As above, we consider the square pitch error function

$$E\left(\frac{1}{k^\dagger}\right) = \left(12 \log_2 \frac{k^\dagger f}{f^*}\right)^2 \ . \tag{7.13}$$

Theorem 3 again holds, and hence, for harmonic "down" errors, such that $f^* \geq f > 0$, the harmonic error factor, k, that gives the residual error factor closest to 1 is

$$k = \begin{cases} 1/\left\lfloor \frac{f^*}{f} \right\rfloor & \text{if } \frac{f^*}{f} \leq \sqrt{\left\lfloor \frac{f^*}{f} \right\rfloor \left(\left\lfloor \frac{f^*}{f} \right\rfloor + 1\right)} \\ 1/\left\lceil \frac{f^*}{f} \right\rceil & \text{otherwise.} \end{cases} \tag{7.14}$$

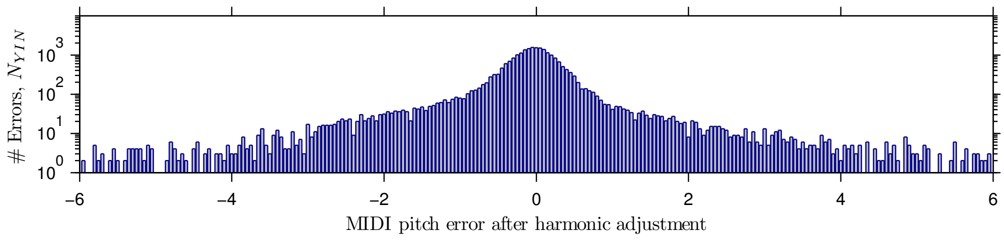
Summarising, given an estimated frequency, f, and the ground-truth frequency, f^*, the harmonic error factor, k such that the residual error factor, ϵ, minimises the difference with the ground-truth pitch estimate is:

$$k = \begin{cases} 1/\left\lceil \frac{f^*}{f} \right\rceil & \text{if} \quad 1 < \sqrt{\left\lfloor \frac{f^*}{f} \right\rfloor \left(\left\lfloor \frac{{}^*f}{f} \right\rfloor + 1\right)} \leq \frac{f^*}{f} \\ 1/\left\lfloor \frac{f^*}{f} \right\rfloor & \text{if} \quad 1 \leq \frac{f^*}{f} < \sqrt{\left\lfloor \frac{f^*}{f} \right\rfloor \left(\left\lfloor \frac{f^*}{f} \right\rfloor + 1\right)} \\ \left\lfloor \frac{f}{f^*} \right\rfloor & \text{if} \quad 1 < \frac{f}{f^*} \leq \sqrt{\left\lfloor \frac{f}{f^*} \right\rfloor \left(\left\lfloor \frac{f}{f^*} \right\rfloor + 1\right)} \\ \left\lceil \frac{f}{f^*} \right\rceil & \text{if} \quad 1 < \sqrt{\left\lfloor \frac{f}{f^*} \right\rfloor \left(\left\lfloor \frac{f}{f^*} \right\rfloor + 1\right)} < \frac{f}{f^*} \quad . \end{cases} \tag{7.15}$$

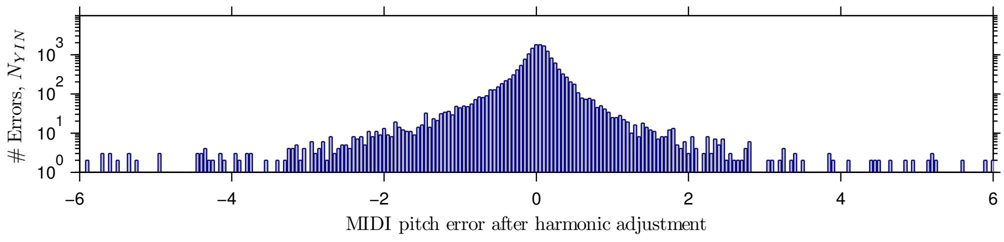
and the frequency estimate that minimises the difference with the ground-truth pitch estimate is:

$$\hat{f} = \frac{f}{k} \ . \tag{7.16}$$

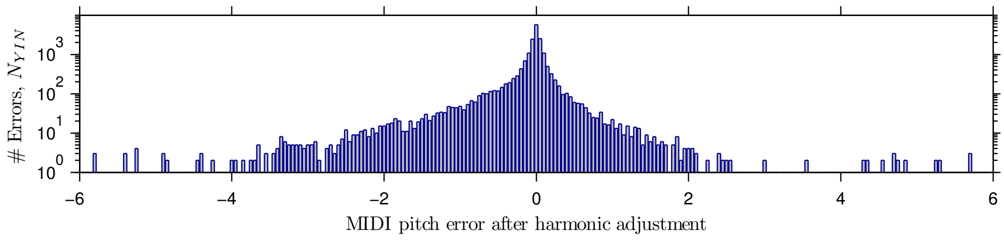
Theorem 2 *After removing harmonic errors, the maximum absolute pitch error is 6 semitones.*



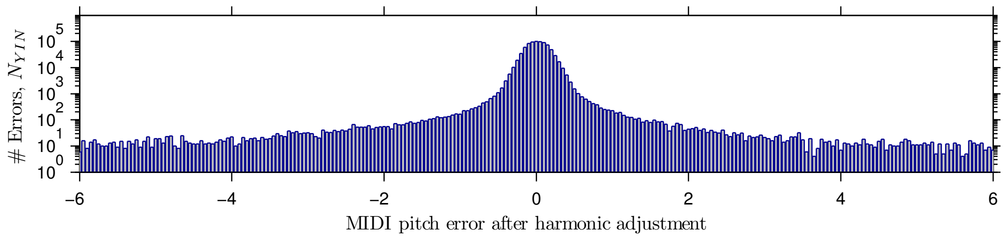
(a) Bagshaw



(b) Keele



(c) VOQUAL'03



(d) MIREX QBSH

Figure 7.4: Number of errors vs. minimal possible pitch error after best harmonic error removal ($\hat{p} - p^*$, in semitones) on the Bagshaw, Keele, VOQUAL'03 and MIREX QBSH datasets.

Proof 2 *From Equation 7.15, the residual pitch error is:*

$$
\begin{aligned}
\hat{p} - p^* &= 12\log_2\left(\frac{f}{kf^*}\right) \\
&= \begin{cases}
12\log_2\left(\left\lceil\frac{f^*}{f}\right\rceil\frac{f}{f^*}\right) & \textit{if} \quad 1 < \sqrt{\left\lfloor\frac{f^*}{f}\right\rfloor\left(\left\lfloor\frac{{}^*f}{f}\right\rfloor+1\right)} \le \frac{f^*}{f} \\
12\log_2\left(\left\lfloor\frac{f^*}{f}\right\rfloor\frac{f}{f^*}\right) & \textit{if} \quad 1 \le \frac{f^*}{f} < \sqrt{\left\lfloor\frac{f^*}{f}\right\rfloor\left(\left\lfloor\frac{f^*}{f}\right\rfloor+1\right)} \\
-12\log_2\left(\left\lfloor\frac{f}{f^*}\right\rfloor\frac{f^*}{f}\right) & \textit{if} \quad 1 < \frac{f}{f^*} \le \sqrt{\left\lfloor\frac{f}{f^*}\right\rfloor\left(\left\lfloor\frac{f}{f^*}\right\rfloor+1\right)} \\
-12\log_2\left(\left\lceil\frac{f}{f^*}\right\rceil\frac{f^*}{f}\right) & \textit{if} \quad 1 < \sqrt{\left\lfloor\frac{f}{f^*}\right\rfloor\left(\left\lfloor\frac{f}{f^*}\right\rfloor+1\right)} < \frac{f}{f^*} \quad .
\end{cases}
\end{aligned}
\tag{7.17}
$$

Considering each case in turn, if $1 < \sqrt{\left\lfloor\frac{f^*}{f}\right\rfloor\left(\left\lfloor\frac{{}^*f}{f}\right\rfloor+1\right)} \le \frac{f^*}{f}$ *then, as* $\lceil x\rceil \ge x$,

$$0 \le 12\log_2\left(\left\lceil\frac{f^*}{f}\right\rceil\frac{f}{f^*}\right) \tag{7.18}$$

and $0 < \hat{p} - p^*$. *Also,*

$$12\log_2\left(\left\lceil\frac{f^*}{f}\right\rceil\frac{f}{f^*}\right) \le 12\log_2\left(\left(\left\lfloor\frac{f^*}{f}\right\rfloor+1\right)/\left\lfloor\frac{f^*}{f}\right\rfloor\right)^{\frac{1}{2}} \tag{7.19}$$

giving $\hat{p} - p^* \le 6$.

If $1 \le \frac{f^*}{f} < \sqrt{\left\lfloor\frac{f^*}{f}\right\rfloor\left(\left\lfloor\frac{f^*}{f}\right\rfloor+1\right)}$ *then,*

$$-12\log_2\left(\left(\left\lfloor\frac{f^*}{f}\right\rfloor+1\right)/\left\lfloor\frac{f^*}{f}\right\rfloor\right)^{\frac{1}{2}} < 12\log_2\left(\left\lfloor\frac{f^*}{f}\right\rfloor\frac{f}{f^*}\right) \tag{7.20}$$

giving $-6 < \hat{p} - p^*$ *and*

$$12\log_2\left(\left\lfloor\frac{f^*}{f}\right\rfloor/\frac{f^*}{f}\right) \le 0 \tag{7.21}$$

as $\lfloor x\rfloor \le x$ *giving* $-6 < \hat{p} - p^* \le 0$.

Following the same procedure for the remaining two cases, $-6 \le \hat{p} - p^* \le 6$ *in all cases* ▮

Applying this to the YIN data for the test datasets, we can calculate the minimal possible pitch errors remaining after post-processing to remove harmonic errors (Figure 7.4). Results appear across the full range of possible values (±6 semitones) indicating that, as expected from the previous results, the errors are not simply harmonic, and that there is a residual error factor ϵ. Hence, even if harmonic errors could

Threshold (pitch cents)	Bagshaw	Keele	VOQUAL'03	MIREX QBSH
5	81.07%	70.20%	42.97%	63.46%
10	69.41%	54.27%	31.53%	41.84%
15	59.14%	42.58%	25.24%	25.36%
20	50.39%	34.63%	21.23%	15.04%
25	42.82%	28.60%	18.50%	9.07%
30	36.63%	24.32%	16.35%	5.71%
40	27.51%	18.16%	13.34%	2.82%
50	21.62%	13.97%	11.18%	1.94%
75	14.16%	8.60%	7.24%	1.19%
100	10.79%	5.88%	5.21%	0.88%
200	4.49%	2.03%	1.69%	0.40%
300	1.94%	0.92%	0.76%	0.22%
400	0.92%	0.52%	0.36%	0.13%
500	0.41%	0.24%	0.17%	0.06%
600	0%	0%	0%	0%

Table 7.4: Percentage of absolute pitch errors greater than specified threshold (in pitch cents) after best harmonic error removal. As expected, all pitch errors greater than 6 semitones (600 pitch cents) have been removed.

be successfully removed, additional work is required to improve the underlying pitch estimation – pitch errors due to the residual error factor being greater than the pitch JND and hence too large for our purposes.

Examining the numbers of errors at various thresholds after applying the above "best harmonic error removal" procedure (Table 7.4), the Bagshaw, Keele and VOQUAL'03 datasets produce significant numbers of residual (non-harmonic) errors greater than half a semitone, Bagshaw having over 20%. On the MIREX QBSH dataset, following YIN with harmonic error removal gives much better results, over 98% of pitch estimates having an error less than half a semitone. Higher harmonic errors give *better* subsample period estimates if the harmonic errors can be removed – an nth harmonic error allows the period to be estimated to the nearest $\frac{1}{n}$th of a sample. As MIREX QBSH data gave more high harmonic errors, this will give better pitch estimates after the harmonic errors are removed.

We note that the procedure as given requires both pitch estimates and ground-truth pitch values and is therefore not a technique for removing harmonic errors – it is a procedure for evaluating the possible improvements *if* a suitable technique could be found to remove harmonic errors.

Examining the percentage decrease in the number of errors by applying the harmonic error removal to the YIN results (Table 7.5), we see that it could be possible to remove some 60% of MIREX QBSH errors greater than 50 pitch cents if a procedure was found to remove of harmonic errors. However, at the 50 cent level, smaller improvements are seen for the other datasets (harmonic error removal being most effective at reducing the number of errors greater than 2–3 semitones). Given the more harmonic nature of the errors on the MIREX QBSH data (as in Figure 7.3) it is unsurprising that removing harmonic errors was most effective on this dataset.

Examining the summary statistics after applying this procedure, (Table 7.6) we see that the GEP is reduced to $< 1.5\%$ for all four datasets. The other statistics show some reduction, but still show a significant number of errors.

7.3 Conclusions

We introduced several pitch database and pitch estimator performance metrics (Section 7.1) and presented a novel analysis of the performance of the YIN pitch estimation algorithm. Using histograms of the magnitude of the pitch errors (Section 7.2.3), we found that most errors occur at approximately the 10 pitch cent level, the vast ma-

Threshold (pitch cents)	Bagshaw	Keele	VOQUAL'03	MIREX QBSH
5	-7.10%	-11.94%	-20.55%	-16.45%
10	-7.76%	-12.99%	-14.37%	-21.55%
15	-7.79%	-12.92%	-12.34%	-26.55%
20	-7.84%	-11.82%	-10.99%	-30.41%
25	-8.10%	-11.82%	-11.04%	-35.06%
30	-8.34%	-11.82%	-11.30%	-40.70%
40	-8.22%	-13.74%	-13.33%	-52.70%
50	-7.22%	-16.48%	-15.17%	-60.65%
75	-8.06%	-23.23%	-22.00%	-72.04%
100	-9.94%	-31.85%	-28.63%	-78.30%
200	-26.42%	-62.51%	-57.05%	-89.29%
300	-49.63%	-79.93%	-75.30%	-93.97%
400	-69.06%	-88.06%	-86.96%	-96.57%
500	-84.29%	-94.34%	-93.51%	-98.46%
600	-100%	-100%	-100%	-100%

Table 7.5: Percentage change in absolute pitch errors greater than specified threshold (in semitones) after best harmonic error removal. For the QBSH dataset more than half the pitch errors greater than 40 cents could be removed if the best harmonic error removal was achievable. However, it has less effect on the other datasets.

	Bagshaw	Keele	VOQUAL'03	MIREX QBSH
10c	75.18%	61.84%	36.10%	52.14%
GEP	1.41%	0.73%	0.67%	0.18%
PMP	77.10%	70.24%	88.41%	15.70%
MMP	33.28%	25.28%	17.46%	13.80%

Table 7.6: Summary pitch statistics for Bagshaw, Keele, VOQUAL'03 and MIREX QBSH datasets after removing harmonic errors.

jority of errors being in the range 1 cent to 100 cents (1 semitone). However, octave errors also occurred and the nature of the harmonic errors produced was examined (Section 7.2.4). Given the presence of harmonic errors, we considered the effect of harmonic error removal on YIN (Section 7.2.5) and found that even if the harmonic errors could be removed, YIN would still produce errors that were larger than we desire.

In order to successfully produce an Expressive MIDI representation of audio, we require a better pitch estimator than YIN. We next consider alternative pitch estimation algorithms.

Chapter 8

Alternatives to the YIN Pitch Estimator

We examined the performance of the YIN pitch estimation algorithm (Chapter 7) and found that it made both harmonic errors (pitch estimates at multiples and fractions of the correct frequency) and smaller residual errors. We therefore examined alternative pitch estimators which may improve the performance of our Expressive MIDI system. Many pitch estimation algorithms have been proposed in the literature. For example, 13 algorithms, including YIN, are compared in [Camacho and Harris, 2008][1].

In general terms, most pitch estimation algorithms use a similar procedure: windows of samples are taken at equally spaced time intervals; a pitch detection function is then applied to each window of data, indicating how likely each pitch estimate is to be correct; and the "best" pitch candidate for each frame is then selected (e.g. using a peak picking algorithm).

We noted that YIN makes *harmonic* errors (Section 7.2.4), we therefore considered two alternative pitch estimation algorithms which take specific steps to avoid these errors: the PRAAT[2] autocorrelation based pitch estimator (AC-P) [Boersma, 1993] (Section 8.1.1); and SWIPE$'$, the "prime harmonics" variant of the Sawtooth Wave Inspired Pitch Estimator (SWIPE) [Camacho, 2007, Camacho and Harris, 2008] (Section 8.1.2) – a time-frequency domain pitch algorithm.

[1]Parts of this work were originally presented at the 128th Convention of the Audio Engineering Society [Welburn and Plumbley, 2010].

[2]PRAAT[Boersma and Weenink, 2010] is an application for "doing phonetics by computer" and features both a graphical user-interface and a command-line tool. It includes facilities for: spectral analysis; pitch analysis; formant analysis; intensity analysis; annotating audio; and the manipulation of pitch, duration, intensity, and formants.

Pitch estimation performance varies with different data, as we saw for YIN (Section 7.2.2). Given a known type of audio data, it may be possible to optimise algorithm parameters to improve algorithm performance, but for the Expressive MIDI system we wish to be able to apply a pitch estimator on an arbitrary example of source audio material. Without ground-truth pitch data, it is not possible to assess the performance of a pitch estimator to optimise parameter values. Given unlabelled data, parameter values can only be refined based on known features of the data (e.g. sample rate) and according to the problem at hand (e.g. the expected pitch range). In line with our aim of using available tools, we assumed that the default parameters had been tuned to give "good" results for typical audio data and used these default values in our tests – we note, however, that the ground-truth pitch estimates for all four databases are within the default pitch ranges of the pitch estimation algorithms.

We next examine the PRAAT AC-P and SWIPE′ pitch algorithms.

8.1 Alternative Pitch Estimation Algorithms

8.1.1 PRAAT AC-P

The PRAAT tool for phonetics [Boersma and Weenink, 2010] includes an autocorrelation based pitch estimator (AC-P)[Boersma, 1993]. The algorithm initially upsamples the signal, and frames of 3 times the period of the minimum pitch are then selected. These frames are set to zero mean and a windowing function is applied. The autocorrelation function is then calculated using the FFT of the frame. For each frame, peaks are selected that represent the strongest pitch candidates. Finally, a "best" pitch path is calculated that minimises jumps in pitch and the number of voiced/unvoiced switches.

We next give in detail the algorithm used by PRAAT AC-P.

A frame of data centred at time t, of length $2T$, is selected, set to zero mean, and a windowing function, $w(\cdot)$ is applied:

$$a(t,i) = (x(t+i-T) - \mu_x)\, w(i-T) \tag{8.1}$$

where: i is an index into the window of data; μ_x is the mean of the data across the window, $\mu_x = \frac{1}{2T}\sum_{i=0}^{2T-1} x(t+i-T)$; and $w(\cdot)$ is a Hann windowing function of length $2T$ centred at T. The Hanning window function is

$$w(t,L) = \frac{1}{2}\left(1 - \cos\left(2\pi\left(\frac{t}{L-1} + \frac{1}{2}\right)\right)\right) \tag{8.2}$$

where L is the window length.

The pitch detection function is based on the normalised autocorrelation of the windowed data:

$$r_a(t,\tau) = \frac{\sum_{i=0}^{2T-1} a(t,i+\tau)a(t,i+\tau)}{\sum_{i=0}^{2T-1} a(t,i)^2} \tag{8.3}$$

In order to approximate the discrete-time autocorrelation of the signal, $r_x(n,\tau)$, from the normalised autocorrelation of the *windowed* signal, $r_a(t,n,\tau)$, we divide this by the normalised autocorrelation of windowing function:

$$r_x(t,\tau) = \frac{r_a(t,\tau)}{r_w(\tau)} \tag{8.4}$$

where the normalised autocorrelation of window is calculated in the continuous domain by integration:

$$\begin{aligned} r_w(\tau) &= \frac{\int_{t=-T}^{T} w(t)w(t+\tau)dt}{\int_{t=-T}^{T} w(t)^2 dt} \\ &= \left(1 - \frac{|\tau|}{T}\right)\left(\frac{2}{3} + 13\cos\left(\frac{2\pi\tau}{T}\right)\right) + \frac{1}{2\pi}\sin\left(\frac{2\pi|\tau|}{T}\right) \ . \end{aligned} \tag{8.5}$$

It is possible to produce a bandwidth limited continuous-time version, $\hat{x}(\hat{t})$, of a sampled function, $x(t)$, using sinc interpolation:

$$\hat{x}(\hat{t}) = \sum_{i=-\infty}^{\infty} x(i)\frac{\sin\left(\pi s_{\mathrm{R}}(\hat{t}-i)\right)}{\pi s_{\mathrm{R}}(\hat{t}-i)} \tag{8.6}$$

where $\hat{\cdot}$ indicates values in the continuous time domain and s_{R} is the sample rate. AC-P approximates $\hat{r}_x$, the continuous-time version of the discrete-time normalised signal autocorrelation (r_x, Equation 8.4), by using a finite number of terms, W, either side of the given time, and using a Hann window to taper the function to zero at $\pm W$:

$$\hat{r}_x(t,\hat{\tau}) = \sum_{i=-W}^{W-1} r_x(t,\tau+i)w(i,2W)\frac{\sin\left(\pi s_{\mathrm{R}}(\hat{\tau}-(\tau+i))\right)}{\pi s_{\mathrm{R}}(\hat{\tau}-(\tau+i))} \tag{8.7}$$

where $w(\cdot)$ is, again, a Hann window function this time of width $2W$.

Given this function, the places and the heights of the maxima are found using a suitable algorithm (Boersma [1993] suggests Brent's algorithm [Brent, 1973, Ch. 4], but does not specify the algorithm used in the implementation of AC-P).

For each maximum of $\hat{r}_x(t,\hat{\tau})$, a local strength value is calculated:

$$\hat{R}(t,\hat{\tau}) = \hat{r}_x(t,\hat{\tau}) - c_{\mathrm{oct}}\log_2(p_{\mathrm{min}}\hat{\tau}) \tag{8.8}$$

where c_{oct} is an "octave cost" which makes the strength selection favour higher frequencies (in order to avoid "too low" errors) and p_{min} is the minimum pitch to be accepted.

Some number of Period / Strength pairs (m a parameter for the algorithm) with the highest local strength are remembered for each frame n: $\left\langle \hat{\tau}_{n,i}, \hat{R}(t(n), \hat{\tau}_{n,i}) \right\rangle$ ($i = 1 \ldots m$), where $t(n)$ is the centre timing for the window. Dynamic programming is then used to find a path through these candidate solutions to minimise the cost function, $c_{\text{path}}(\{i_n\})$:

$$c_{\text{path}}(\{i_n\}) = \sum_{t=2}^{N} c_{\text{tx}}(\hat{\tau}_{n-1,i_{n-1}}, \hat{\tau}_{n,i_n}) - \sum_{t=1}^{N} \hat{R}(t(n), \hat{\tau}_{n,i_n}) \tag{8.9}$$

where i_n is index of the selected candidate Period / Strength pair for frame n and $c_{\text{tx}}(\hat{\tau}_{n-1,i_{n-1}}, \hat{\tau}_{n,i_n})$ is the transition cost from the pitch estimate for frame $n-1$ to the pitch estimate at frame n.

$$c_{\text{tx}}(\hat{\tau}_{n-1,i_{n-1}}, \hat{\tau}_{n,i_n}) = \begin{cases} 0 & \text{if both } \hat{\tau}_{n-1,i_{n-1}} = 0 \text{ and } \hat{\tau}_{n,i_n} = 0 \\ c_{\text{vuv}} & \text{if either } \hat{\tau}_{n-1,i_{n-1}} = 0 \text{ or } \hat{\tau}_{n,i_n} = 0 \\ c_{\text{oj}} \left| \log_2 \left(\frac{\hat{\tau}_{n-1,i_{t-1}}}{\hat{\tau}_{n,in}} \right) \right| & \text{if neither } \hat{\tau}_{n-1,i_{n-1}} = 0 \text{ nor } \hat{\tau}_{n,i_n} = 0 \end{cases} \tag{8.10}$$

where c_{vuv} is a cost value for changes between voiced and unvoiced parts of the signal and c_{oj} is a cost value for octave jumps in the signal – used to reduce harmonic errors. The pitch estimates, $\hat{\tau}_{n,i}$, the pitch estimate timings, $t(n)$, and the period estimate strengths, $\hat{R}(t(n), \hat{\tau}_{n,i})$ from the lowest cost path are then returned for each window.

Boersma [1993] tested AC-P against sampled sine waves and pulse trains with a sample rate of 10kHz and reported pitch determination errors, $\Delta F/F$, of $< 5 \times 10^{-4}$ and $< 5 \times 10^{-5}$ respectively i.e. MIDI pitch errors of $< 8.65 \times 10^{-3}$ and $< 8.66 \times 10^{-4}$. The sine wave and pulse train were selected as "maximally spectral different" signals – the pulse train containing all harmonics at equal strength.

In PRAAT, AC-P is called using the "To Pitch..." command with parameters of a "Time step" in seconds, a "Pitch floor" in Hz and a "Pitch ceiling" in Hz. AC-P can also be called using "To Pitch... (ac)" in which case additional parameters can be set affecting the windowing function and the "best path" estimation (i.e. W, c_{oct}, c_{vuv} and c_{oj}).

8.1.2 SWIPE and SWIPE′

SWIPE [Camacho and Harris, 2008] is based on the amplitude of the Fourier spectrum calculated over individual frames from the audio signal (i.e. it operates in the time-frequency domain). Considering pitch bins from the spectrogram and their harmonically related components, it is designed to avoid many of the harmonic errors that are seen with YIN. Sub-sample pitch estimates are made by cubic spline interpolation of the FFT output. SWIPE returns frequency estimates, timings for the estimates, and the strength of the pitch estimate. Camacho and Harris provide a Matlab implementation of SWIPE.

The SWIPE pitch estimate at time τ is

$$f_{\mathrm{SWIPE}}(\tau) = \underset{f}{\operatorname{argmax}} \left(S(\tau, f) \right) \tag{8.11}$$

i.e. SWIPE gives the frequency, f Hz, that maximises the pitch candidate strength function, $S(\cdot)$. The SWIPE strength output is $S(\tau, f_{\mathrm{SWIPE}}(\tau))$.

Given a frequency range of $f_{\min}$ to $f_{\max}$ Hz, the pitch candidate frequencies considered are $f_\alpha = f_{\min} \times 2^{\alpha \Delta p}$, $\alpha \in \mathbb{N}$, $0 \le \alpha \le \frac{\log_2\left(\frac{f_{\max}}{f_{\min}}\right)}{\Delta p}$ i.e. at intervals of $12\Delta p$ semitones from the minimum frequency to the largest value of $f_{\min} \times 2^{\alpha\Delta p}$ that is less than the maximum frequency.

For a given pitch candidate, f, the pitch candidate strength is estimated for a window of ρ periods of the waveform i.e. over a window of $\Omega(f) = \rho \frac{f_{\mathrm{S}}}{f}$ samples where f_{S} is the sample rate in Hz. The pitch candidate strength estimate for a window of size $\Omega(f)$ is estimated by interpolating pitch candidate strength estimates at the closest power-of-two window sizes to $\Omega(f)$:

$$S(\tau, f) = (1 - \lambda(f, \Omega(f)))S_{L^{-}_{\Omega(f)}(f)}(\tau, f) + \lambda(f, \Omega(f))S_{L^{+}_{\Omega(f)}(f)}(\tau, f) \tag{8.12}$$

where

$$\begin{aligned} L^{-}_{\Omega}(f) &= L : 2^L \le \Omega \text{ and } 2^{L+1} > \Omega,\ L \in \mathbb{N}, \\ L^{+}_{\Omega}(f) &= L : 2^{L-1} < \Omega \text{ and } 2^L \ge \Omega,\ L \in \mathbb{N} \\ \lambda(f, \Omega) &= \log_2(\Omega) - L^{-}(f)\ . \end{aligned} \tag{8.13}$$

For a window of size 2^L, the pitch candidate strength at time τ is given by

$$S_L(\tau, f) = \left\langle \frac{\mathbf{k}(f)}{|\mathbf{k}^{+}(f)|}, \frac{\hat{\mathbf{s}}_L(\tau)}{|\hat{\mathbf{s}}_L(\tau)|} \right\rangle \tag{8.14}$$

the inner product between a kernel function based on the frequency candidate, $\frac{\mathbf{k}(f)}{|\mathbf{k}^+(f)|}$, and the normalised spectral strengths, $\frac{\hat{\mathbf{s}}_L(\tau)}{|\hat{\mathbf{s}}_L(\tau)|}$, over the window of length 2^L centred at time τ. The SWIPE kernel function weights the spectral strengths depending upon the pitch estimate being considered.

The kernel function and the normalised spectral strengths are calculated at intervals of $\Delta\epsilon$ ERBs. Given a frequency of f_{min} to f_{max} Hz, the frequencies considered are $\eta(m\Delta\epsilon)$, $m = 0, \ldots, N_{\text{k}}$ where $N_{\text{k}} = \left\lfloor \frac{ERBs(f_{\max})}{\Delta\epsilon} \right\rfloor$, $\eta(\cdot)$ is a conversion function from ERBs to Hz and $ERBs(\cdot)$ is a conversion function from Hz to ERBs.

The SWIPE kernel consists of the core kernel function, $\mathbf{k}(f)$ and a normalising factor, $|\mathbf{k}^+(f)|$. The core kernel function, for pitch candidate f, is

$$\begin{aligned}
\mathbf{k}(f) &= (k_0(f), \ldots, k_{N_{\text{k}}}(f)) \\
k_i(f) &= \frac{1}{\sqrt{\eta(i\Delta\epsilon)}} \sum_{h \in \mathbf{H}} K_h(f, \eta(i\Delta\epsilon)) \qquad i = 0, \ldots, N_{\text{k}} \\
K_h(f, f') &= \begin{cases} \cos\left(2\pi \frac{f'}{f}\right) & \text{if } \left|\frac{f'}{f} - h\right| < \frac{1}{4} \\ \frac{1}{2}\cos\left(2\pi \frac{f'}{f}\right) & \text{if } \frac{1}{4} < \left|\frac{f'}{f} - h\right| < \frac{3}{4} \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{8.15}$$

where $\mathbf{H}$ is the set of harmonics, $h_i \in \mathbb{N}$, to consider. The normalising factor, $|\mathbf{k}^+(f)|$, considers only the positive components of the kernel function

$$\begin{aligned}
\mathbf{k}^+(f) &= (k_0^+(f), \ldots, k_{N_{\text{k}}}^+(f)) \\
k_i^+(f) &= \max(0, k_i(f)) \qquad i = 0, \ldots, N_{\text{k}} \ .
\end{aligned} \tag{8.16}$$

The spectral strengths are calculated by interpolating the spectrogram of size 2^L centred at time τ

$$\begin{aligned}
\hat{\mathbf{s}}_L(\tau) &= (s_{L,0}(\tau), \ldots, s_{L,N_{\text{k}}}(\tau)) \\
s_{L,m}(\tau) &= \left|\hat{X}_{2^L}(\tau, \eta(m\Delta\epsilon))\right|^{\frac{1}{2}} \\
\hat{X}_{2^L}(\tau, f') &= I\left(\{0, \ldots, 2^L - 1\}, X_{2^L}[\tau, \{0, \ldots, 2^L - 1\}], 2^L \frac{f'}{f_{\text{S}}}\right) \\
X_{2^L}(\tau, \phi) &= \sum_{\tau' = -\infty}^{\infty} w_{2^L}(\tau' - \tau) x(\tau') e^{-2\pi i \frac{\phi\tau'}{2^L}}
\end{aligned} \tag{8.17}$$

where $N = 2^L$, $I(\Phi, F(\Phi), \phi)$ is an interpolating function to predict $F(\phi)$ from $F(\Phi)$ (the Matlab implementation uses the `interp1` function with cubic spline interpolation), and $w_N(\tau)$ is a Hann window of size N centred at 0

$$w_N(\tau) = \begin{cases} 0.5\left(1 - \cos\left(2\pi\left(\frac{\tau}{N} + \frac{1}{2}\right)\right)\right) & \text{for } -\frac{N}{2} \leq \tau < \frac{N}{2} \\ 0 & \text{otherwise} \end{cases}. \tag{8.18}$$

The actual implementation of SWIPE reduces the time taken by calculating the required 2^l size FFTs with a specified overlap for the Hann windows, w_{H}, giving spectral strengths at intervals of 2^{L-1} samples. These spectral strengths are then linear interpolated to estimate the spectral strengths at the specified time resolution, Δt (seconds).

As a final step, the maxima of the pitch candidate strength function are found and parabolic interpolation is used to provide a pitch estimate to the nearest pitch cent.

SWIPE allows parameters to be specified to control:

- Output frequency range, $f_{\min}$ to $f_{\max}$ in Hz (default 30Hz to 5kHz i.e. MIDI pitch 22 to 111);
- Hop size between output pitch estimates, Δt in seconds (Default 0.001 seconds);
- Hann window overlap w_{H} as a proportion of the FFT window size (in the range $[0, 1]$) (Default 0.5 i.e. 50% overlap);
- Spectrum sample granularity $\Delta\epsilon$ (the spectrum is sampled uniformly using the ERB scale [Moore, 1997]) (default 0.1 ERBs);
- Pitch granularity, Δp in fraction of an octave (default 48 pitch estimates per octave, i.e. 0.25 semitones).

Although the pitch estimation occurs at a granularity specified in the parameters, the pitch is "fine-tuned" using parabolic interpolation to output pitch estimates with 1 pitch cent resolution (i.e. 0.01 semitones).

The standard SWIPE algorithm considers all harmonics, $\mathbf{H} = \{1, \dots\}$, up to the highest pitch candidate. Camacho and Harris [2008] noted that SWIPE′ (pronounced "Swipe Prime"), which only considers the prime harmonics, $\mathbf{H} = \{1, 2, 3, 5, 7, 11, \dots\}$, up to the highest pitch candidate, further reduces the number of harmonic errors. In our evaluations, we use SWIPE′.

8.1.3 Previous Evaluations

There have been many previous evaluations of pitch estimators. We examine several of these and compare their methodologies and their results on the datasets under consideration (Section 7.1.2).

	Bagshaw OG	Bagshaw YV	Keele
PRAAT AC-P [Boersma, 1993]	7.3%	9.2%	5.1%
YIN [de Cheveigné and Kawahara, 2002]	2.2%	1.4%	2.4%

Table 8.1: GEP results from YIN vs. PRAAT AC-P assessment from de Cheveigné and Kawahara [2002] on Bagshaw and Keele datasets. The Bagshaw dataset results are given using both the original ground-truth data (OG) and ground-truth data generated using a YIN variant (YV).

De Cheveigné and Kawahara [2002] evaluated the YIN algorithm against 10 other algorithms. The databases used included Bagshaw and Keele databases (Section 7.1.2). GEPs were given for YIN on both datasets based on the original ground-truth data. Additionally, YIN was tested on the Bagshaw dataset using alternative ground-truth data. The alternative ground-truth estimates were created by: applying a YIN variant to the EGG data; manually removing obviously incorrect results; and only keeping remaining results if there was evidence of glottal vibration. In order to compensate for time-differences between the EGG and audio data and differences in the time-alignment of the algorithms, pitch results were aligned to minimise the error rate. YIN gave a lower GEP than AC-P over each dataset (Table 8.1). Using the original ground-truth data, the GEP for YIN on each dataset is less than the value we found using default YIN parameter settings and estimated timings for the YIN pitch estimates (Table 7.3).

	Bagshaw	Keele
PRAAT AC-P [Boersma, 1993]	0.73%	2.90%
YIN [de Cheveigné and Kawahara, 2002]	0.33%	1.40%
SWIPE [Camacho and Harris, 2008]	0.15%	0.87%
SWIPE′ Camacho and Harris [2008]	0.13%	0.83%

Table 8.2: GEP results from Camacho [2007] on Bagshaw and Keele datasets. The ground-truth data was restricted to those pitch estimates at times at which all 12 algorithms tested agreed the signal was pitched.

Camacho [2007] evaluated the SWIPE and SWIPE′ algorithms against 12 other algorithms (including YIN and PRAAT AC-P) over 4 databases using the GEP. Algorithms were set to produce pitch estimates every millisecond and the results were associated with the time of the centre of the analysis window. Pitch timings were adjusted to give the best alignment with the ground-truth. Results use only the pitch estimates for those times at which both the ground-truth and all the algorithms agreed that the sound was pitched. Again, the Bagshaw and Keele databases were included. GEPs on the Bagshaw database were less than 1% for 8 of the 12 algorithms tested ($< 0.5\%$ for both YIN and SWIPE/SWIPE′), the mean GEP being 1.70%. For the Keele database, the mean GEP was 3.00% ($< 1.5\%$ for YIN and SWIPE/SWIPE′). The very low error rates (Table 8.2) suggest that the restriction to clearly pitched ground-truth estimates has resulted in an easier problem. The YIN GEP reported for YIN by Camacho is much lower than that published by de Cheveigné. On both the Keele and Bagshaw databases, SWIPE′ had a lower GEP than SWIPE.

	Combined Dataset
PRAAT AC-P [Boersma, 1993]	3.7%
YIN [de Cheveigné and Kawahara, 2002]	5.1%
SWIPE [Camacho and Harris, 2008]	3.0%

Table 8.3: GEP results for YIN vs. PRAAT AC-P assessment from Signol et al. [2008] on a combined dataset based on the Bagshaw, Keele and VOQUAL'03 datasets. The ground-truth data was generated from the EGG data.

Signol et al. [2008] presented a methodology for evaluating pitch estimators based on both the pitch estimates and Voiced-Unvoiced (VuV) hypotheses - in order for a pitch estimate to be correct, the pitch estimator must judge whether the signal is voiced and then produce a correct pitch estimate. In order to remove biases caused by the accuracy of the VuV decision, the VuV decision threshold was adjusted for each algorithm to produce equal numbers of false positives and false negatives. Pitch estimates were linearly interpolated to agree with the timings of the ground-truth. This was tested on YIN, SWIPE and PRAAT AC-P. For PRAAT AC-P, the “pitch post-processing” was “turned off” by setting parameter values (presumably deactivating the best-path search by setting the costs for changing between voiced and unvoiced segments. c_{vuv}, and the cost for octave jumps, c_{oj}, to zero). Three databases were used: Bagshaw; Keele; and “Daless”, the French speech examples from the VOQUAL'03 database (Section 7.1.2). Reference VuV decisions and pitch estimates were generated from the EGG data[3] in order to remove variation in the quality of the ground-truth data. The GEPs reported (Table 8.3) are larger than the values reported

[3]See Section 7.1.1 for more information on pitch estimation from EGG

by Camacho on any of the datasets. Additionally, the ranking of the three algorithms differs – although both Camacho and Signol et al. reported that SWIPE produced the best results, Signol et al. found PRAAT AC-P was better than YIN and Camacho found YIN to be better than PRAAT AC-P. The performance differences between the algorithms being less than those in the original publications, Signol suggested the need for further analysis the strengths and weaknesses of individual algorithms.

We have already seen (Chapter 7) that matching the ground-truth data may be a different problem for each dataset and from these three evaluations, we again see that algorithm performance reported using the GEP varies greatly depending upon the ground-truth data used. We are interested in evaluating the algorithms on both easy and difficult data and do not wish to simplify the problem by modifying the datasets to contain only the most strongly pitched parts of the signal (as done by Camacho [2007]). Similarly, uniform data quality (as produced by Signol et al. [2008] by generating ground-truth data) is not the same as *high* data quality. In addition to the Keele, Bagshaw and VOQUAL'03 databases, we consider the MIREX QBSH dataset which includes manually entered pitch estimates. We next compare the performance of the YIN, PRAAT AC-P and SWIPE$'$ algorithms separately on each dataset with the provided ground-truth data.

8.1.4 Comparing YIN with SWIPE$'$ and AC-P

Design

In Chapter 7, we saw that fixed threshold pitch metrics gave a partial picture of pitch estimator performance. In order to compare the quality of the SWIPE$'$ and AC-P pitch estimators with YIN, we looked to examine the fixed threshold error metrics and pitch error distributions as we did for YIN (Chapter 7). We also considered comparisons between the distributions.

Implementation

In order to generate the results for SWIPE$'$ and AC-P, wrapper Matlab functions were created which allowed parameters and results to be passed using similar Matlab structures to those used by YIN. The SWIPE$'$ and AC-P output was then evaluated using the same Matlab procedures as used for YIN (Section 7.2.1) but calling the wrapper functions rather than YIN. Therefore, the exact same evaluations were applied – the only difference being the pitch data under consideration.

```
1  form Process Pitch
2      text infile
3      text outfile
4  endform
5  printline Processing file 'infile$'
6  Read from file... 'infile$'
7  # Estimate the pitch
8  To Pitch... 0.005 20 15000
9  printline Writing file 'outfile$'
10 Write to text file... 'outfile$'
```

Listing 8.1: PRAAT script to: accept “infile” and “outfile” parameters (lines 1–4); read in an audio file (line 6); generate pitch estimates using the AC-P pitch estimator (line 8); and write the estimates to an output file (line 10).

In order to use AC-P, we created a PRAAT script (Listing 8.1) which was run from Matlab via a system call to the `praatcon` Windows PRAAT console. PRAAT loaded the specified audio file and processed it calling AC-P using the “To Pitch...”. The AC-P pitch estimates were calculated at 0.005 second intervals and for the pitch range 20Hz to 15kz. PRAAT then saved the resulting pitch estimates to a text file. When the PRAAT script completed, the output text file was loaded into Matlab to give pitch estimates and their timings.

Results

Considering the number of period and MIDI pitch matches and the GEP (Table 8.4), AC-P produced the worst results on all datasets. On the Bagshaw, Keele and VOQUAL’03 datasets, SWIPE′ was better than YIN (which was, in turn, better than AC-P). However, on the MIREX QBSH database, YIN and SWIPE′ were equally effective at matching the MIDI pitch but YIN achieved 5% more period matches than SWIPE′ whilst SWIPE′ produced fewer gross errors (GEP). Neither YIN nor SWIPE′ can therefore be considered conclusively “better” for this dataset – the better algorithm depending upon the specific pitch estimation requirements.

We previously examined the YIN pitch error distributions (Section 7.2.3). Examining the difference between the AC-P and YIN error distributions (Figure 8.1), the number of errors produced by AC-P is greater than that for YIN for larger error values (the “right hand side” of the plot is positive showing that AC-P produced more errors in that region than YIN) whilst the number of errors produced by AC-P is less than that for YIN for small error values (the “left hand side” of the plot is negative, YIN producing more small errors than AC-P). On all four datasets, AC-P produces

	YIN	AC-P	SWIPE′
10c	75.25%	78.08%	74.57%
PMP	77.13%	79.45%	76.22%
MMP	33.74%	39.55%	31.89%
GEP	3.37%	11.88%	2.57%

(a) Bagshaw

	YIN	AC-P	SWIPE′
10c	62.37%	66.44%	59.07%
PMP	70.50%	73.46%	66.48%
MMP	26.87%	33.99%	25.15%
GEP	4.47%	12.11%	2.70%

(b) Keele

	YIN	AC-P	SWIPE′
10c	36.82%	46.55%	30.95%
PMP	46.33%	52.44%	42.76%
MMP	18.81%	26.31%	13.49%
GEP	3.01%	10.28%	1.02%

(c) VOQUAL'03

	YIN	AC-P	SWIPE′
10c	53.34%	61.05%	56.04%
PMP	17.74%	30.65%	22.14%
MMP	16.30%	24.98%	16.30%
GEP	3.68%	10.06%	1.51%

(d) MIREX QBSH

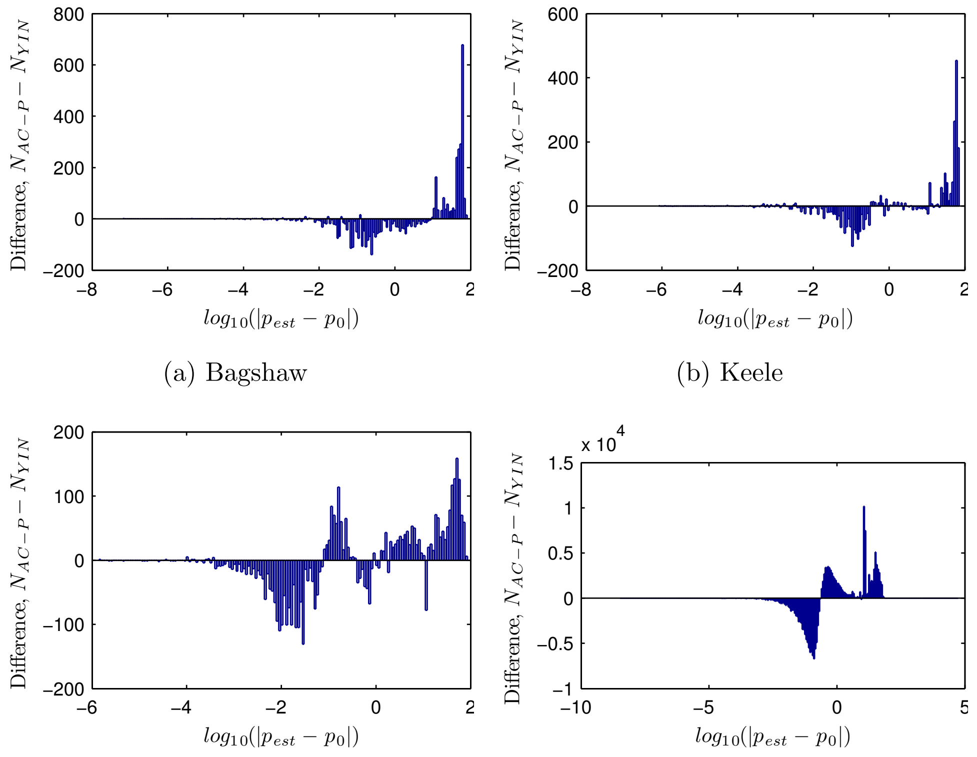
Table 8.4: 10 cent, Period Mismatch, MIDI Mismatch and GEP error percentages by dataset for YIN, AC-P and SWIPE′. Larger values are worse.

more large errors than YIN at the expense of a reduced number of small errors. On these datasets then, YIN produces better pitch estimates than AC-P and the AC-P steps for reducing harmonic errors do not give better results than YIN.

The picture is different for SWIPE′ (Figure 8.2), SWIPE′ producing fewer large errors than YIN on all datasets (negative "right hand side"), however, there are also fewer *small* errors on the VOQUAL'03 and MIREX QBSH datasets (negative "left hand side") – SWIPE′ making more "mid-range" errors. With these signals SWIPE′ appears to reduce harmonic errors more successfully than YIN.

In Figure 8.3, details of the performance of SWIPE′ and YIN pitch estimation on the MIREX QBSH dataset across threshold values are compared: Figure 8.3a emphasising the performance for low error values, YIN providing lower errors than SWIPE′ i.e. YIN is "better" if low error values are required; Figure 8.3b emphasising the performance for high error values, YIN producing more high error values than SWIPE′, introducing a significant number of octave errors, as marked, and a small number of cases in which it failed to find a reasonable pitch estimate and produced an error greater than 100 semitones. YIN is therefore "worse" than SWIPE′ in terms of high error values.

For each dataset, we found that PRAAT AC-P produced worse results than YIN. However, whilst YIN produces more large ("bad") errors than SWIPE′, SWIPE′ produces fewer small ("good") errors. We would like to have a single pitch estimator



(a) Bagshaw (b) Keele

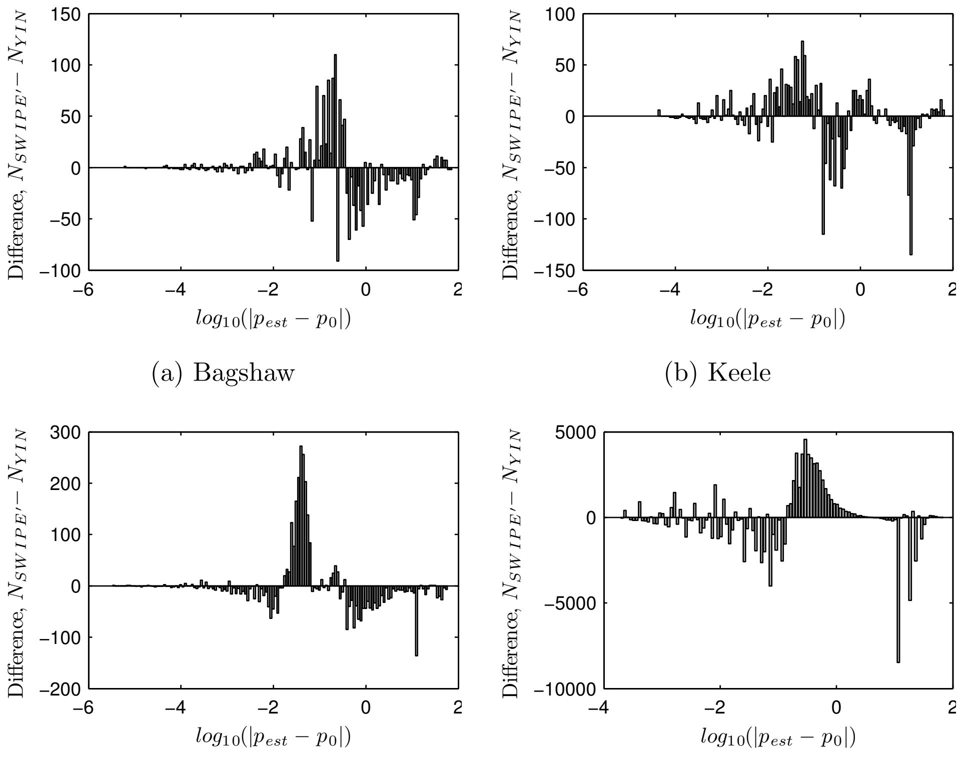
(c) VOQUAL'03 (d) MIREX QBSH

Figure 8.1: Difference between AC-P and YIN error distributions on the Bagshaw, Keele, VOQUAL'03 and MIREX QBSH datasets.

which combines the reduction in gross errors produced by SWIPE′ with the ability of YIN to produce small errors. We therefore looked at producing a pitch estimator combining both the YIN and SWIPE′ algorithms to give a "best of both worlds" result.

8.2 Combining YIN and SWIPE′: The SWIN Estimator

Both YIN and SWIPE′ have strengths – YIN producing high-precision pitch estimates at the cost of increased harmonic errors over SWIPE′. Hence, a combination of the two methods could offer a better estimator combining the strengths of both whilst eliminating their weaknesses. If we simply select the better pitch estimate from those produced by YIN and SWIPE′, we can improve upon the results produced by either estimator alone (Table 8.5). However, this requires us to know in advance which pitch estimator is better. We therefore looked at using a classifier to select either the YIN



(a) Bagshaw (b) Keele

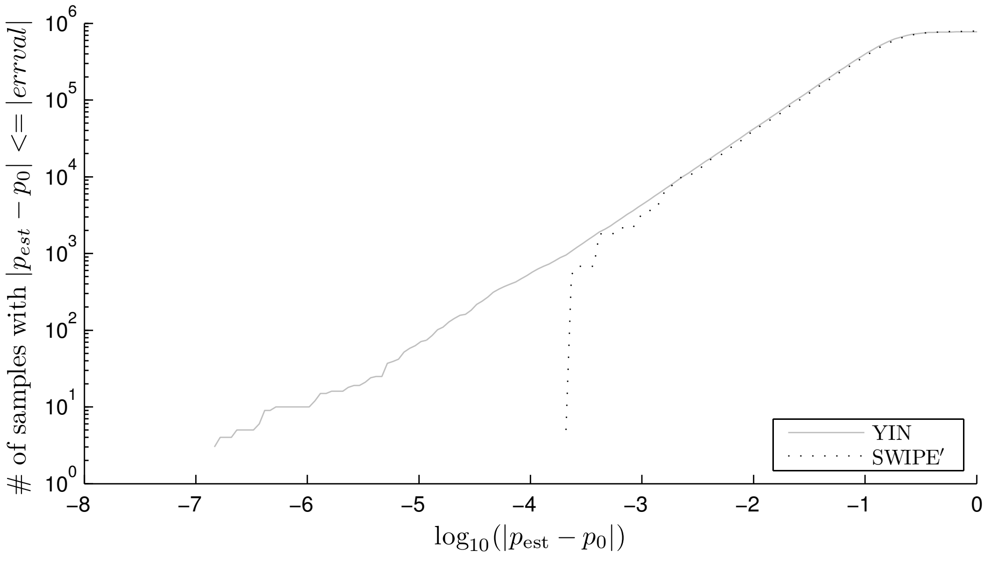
(c) VOQUAL'03 (d) MIREX QBSH

Figure 8.2: Difference between SWIPE$'$ and YIN error distributions on the Bagshaw, Keele, VOQUAL'03 and MIREX QBSH datasets.

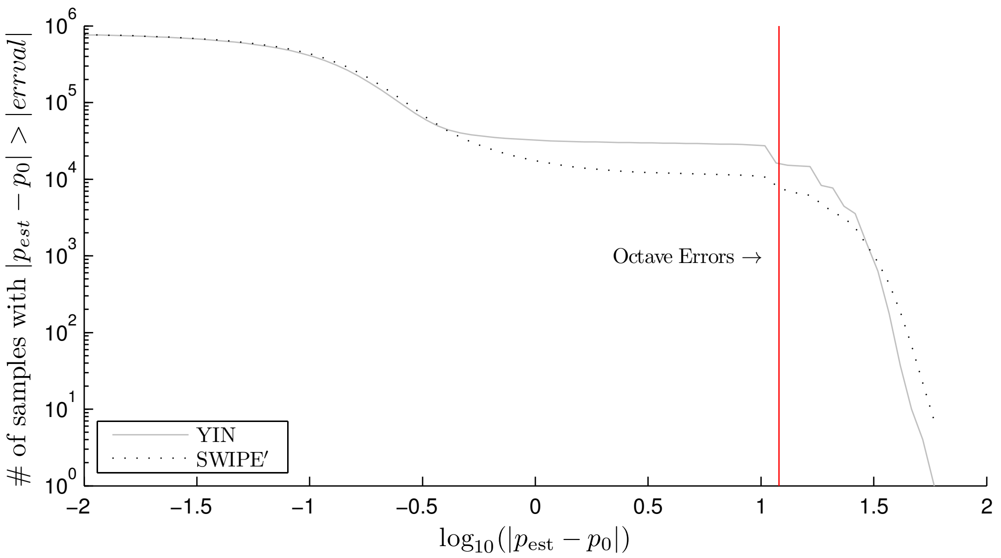
or SWIPE$'$ pitch estimate based on the performance on a set of training data.

We have seen that YIN produces more harmonic errors than SWIPE$'$ (Section 8.1.4). If YIN produces harmonic errors whilst SWIPE$'$ produces correct estimates of the pitch then there should be an integer relationship between their periods (ω_{YIN} and $\omega_{\text{SWIPE}'}$, respectively): if $\omega_{\text{YIN}} > \omega_{\text{SWIPE}'}$ (frequency "too low" errors) the ratio of the period estimates $\frac{\omega_{\text{YIN}}}{\omega'_{\text{SWIPE}}}$ should be an integer; and if $\omega_{\text{YIN}} < \omega_{\text{SWIPE}'}$ (frequency "too high" errors) the ratio of the frequency estimates $\frac{\omega'_{\text{SWIPE}}}{\omega_{\text{YIN}}}$ should be an integer. We therefore created a new pitch estimator ("SWIN") that selected either the YIN or SWIPE$'$ estimate based on the period ratio.

In order to avoid harmonic errors we wish to choose between the YIN and SWIPE$'$ pitch estimates based on some function of $r = \frac{\omega_{\text{YIN}}}{\omega'_{\text{SWIPE}}}$. For each value of r, there is some probability, $\text{P}(e_{\text{YIN}} > e_{\text{SWIPE}'}|r)$, that the YIN error, $e_{\text{YIN}} = |p_{\text{YIN}} - p^*|$, will be greater than the SWIPE$'$ error, $e_{\text{SWIPE}'} = |p_{\text{SWIPE}'} - p^*|$ (where p_{YIN} is the YIN pitch estimate, $p_{\text{SWIPE}'}$ the SWIPE$'$ pitch estimate and p^* the actual pitch). Given this probability we can select the pitch estimate that is most likely to be correct. We do



(a) Log. plot of the number errors *less than* threshold for YIN and SWIPE′ - emphasising behaviour at small error values. Larger numbers of errors to the left-hand side of the plot are *better*, therefore YIN performs better than SWIPE′ at low error thresholds.



(b) Log. plot of the number of errors *greater than* threshold for YIN and SWIPE′ - emphasising behaviour at large error values. Smaller numbers of errors to the right-hand side of the plot are *better*. SWIPE′ is generally better than YIN for thresholds greater than ca. $10^{-0.5} \approx 0.32$, however it also makes a small number of errors greater than the largest YIN errors (ca. $10^{1.5} \approx 32$)

Figure 8.3: Comparison of SWIPE′ and YIN pitch error histograms for the MIREX QBSH dataset at large and small errors.

	YIN	SWIPE′	“Best Of”
$>$ 0.05 semitones	76.40%	77.43%	65.51%
$>$ 0.1 semitones	54.29%	56.04%	42.38%
Period	19.01%	22.14%	11.06%
MIDI	16.74%	16.30%	12.22%
$>$ 0.25 semitones	14.78%	15.93%	8.69%
$>$ .5 semitones	5.24%	4.29%	2.23%
$>$ 2.5 semitones	3.82%	1.59%	1.02%
GEP	3.73%	1.51%	0.96%
$>$ 10 semitones	3.40%	1.36%	0.74%

Table 8.5: Pitch statistics for QBSH – YIN, SWIPE′ and by selecting the better of the YIN and SWIPE′ estimates.

not know this probability distribution, but we can approximate it based on a suitable set of training data and a model of the probability distribution. We do not expect the model to be straightforward – we expect YIN to be better when $r \approx 1$ as it can produce smaller errors and we expect SWIPE′ to be better when $r \approx k (k \in \mathbb{Z}^+)$ or $r \approx \frac{1}{k} (k \in \mathbb{Z}^+)$ as YIN produces more harmonic errors than SWIPE′.

We adopted a piecewise constant model in which, for small ranges of values of r, the probability $\mathrm{P}(e_{\mathrm{YIN}} > e_{\mathrm{SWIPE}'}|r)$ was taken to be fixed:

$$\mathrm{P}(e_{\mathrm{YIN}} > e_{\mathrm{SWIPE}'}|r) = \kappa_j \qquad \text{for } a_j \le r < a_{j+1} \ . \tag{8.19}$$

Theorem 3 *Given a training set of ground-truth pitch values, f_0, and the corresponding YIN and SWIPE′ period estimates (ω_{YIN} and $\omega_{\mathrm{SWIPE}'}$) the optimal estimates of the uniform probabilities κ_j across the series of j ranges of r are:*

$$\kappa_j = \frac{n_j^Y}{N_j} \tag{8.20}$$

where: n_j^Y is the number of pitch estimates such that $a_j \le r < a_{j+1}$ and the error using YIN is greater than the error using SWIPE′; and N_j is the total number of pitch estimates such that $a_j \le r < a_{j+1}$.

Proof 3 *In order to estimate optimal values for κ_j, we consider a set of training data, $\mathcal{T}$. Given $\mathcal{T}$ we can calculate the probability of that dataset being generated from our model*

$$\mathrm{P}(\mathcal{T}) = \prod_{i=1}^{n} \mathrm{P}(t_i) \tag{8.21}$$

assuming that the individual training data samples are independent and identically distributed (iid). We adopted a naïve Bayes approach – in which we assume the data is iid although this assumption does not strictly hold. For SWIPE′ and YIN, this is reasonable as the pitch estimates are calculated based on single frames of audio. However, we note that as frames overlap they are not truly iid.

We can separate out the training samples for each range of values

$$\mathrm{P}(\mathcal{T}) = \prod_{j=1}^{m} \prod_{i \in \mathcal{T}_j} \mathrm{P}(t_i) \tag{8.22}$$

where $\mathcal{T}_j \subset \mathcal{T}$ is the set of training points in slice j.

The marginal probability, $\mathrm{P}(t_i)$, can be calculated by summing across the possible classes for the training sample $\mathrm{P}(t_i) = \sum_{c \in \mathcal{C}} \mathrm{P}(t_i, c_i = c) = \sum_{c \in \mathcal{C}} \mathrm{P}(t_i | c_i = c)\mathrm{P}(c)$ from the product (chain) rule. Adopting a uniform prior, $\mathrm{P}(c) = p_{\mathrm{j}}$ on c within slice j,

$$\mathrm{P}(\mathcal{T}) = \prod_{j=1}^{m} p_{\mathrm{j}} \prod_{i \in \mathcal{T}_j} \sum_{c \in \mathcal{C}} \mathrm{P}(t_i | c_i = c) \tag{8.23}$$

where $\mathcal{T}_j \subset \mathcal{T}$ is the set of training points in slice j.

However, by Bayes theorem,

$$\mathrm{P}(t_i | c_i = c) = \frac{\mathrm{P}(c_i = c | t_i)\mathrm{P}(t_i)}{\mathrm{P}(c_i = c)} \; . \tag{8.24}$$

Therefore

$$\begin{aligned} \mathrm{P}(\mathcal{T}) &= \prod_{j=1}^{m} p_{\mathrm{j}} \prod_{i \in \mathcal{T}_j} \mathrm{P}(t_i) \sum_{c \in \mathcal{C}} \frac{\mathrm{P}(c_i = c | t_i)}{\mathrm{P}(c_i = c)} \\ &= \prod_{j=1}^{m} p_{\mathrm{j}} \prod_{i \in \mathcal{T}_j} \mathrm{P}(t_i) \left(\frac{\mathrm{P}(c_i = 1 | t_i)}{\mathrm{P}(c_i = 1)} + \frac{\mathrm{P}(c_i = 0 | t_i)}{\mathrm{P}(c_i = 0)} \right) \end{aligned} \tag{8.25}$$

where $c = 0$ indicates that the SWIPE′ pitch estimate is better, $c = 1$ indicates that the YIN pitch estimate is better.

Defining $\mathcal{T}_j^0 \subset \mathcal{T}_j$ as the subset of training samples in $\mathcal{T}_j$ for which the SWIPE′ pitch estimate is better and $\mathcal{T}_j^1 \subset \mathcal{T}_j$ as the subset of training samples in $\mathcal{T}_j$ for which the YIN pitch estimate is better, $\mathrm{P}(c_i = 0 | t_i) = 1$ and $\mathrm{P}(c_i = 1 | t_i) = 0$ for $t_i \in \mathcal{T}_j^0$ and $\mathrm{P}(c_i = 1 | t_i) = 1$ and $\mathrm{P}(c_i = 0 | t_i) = 0$ for $t_i \in \mathcal{T}_j^1$. Hence,

$$\mathrm{P}(\mathcal{T}) = \prod_{j=1}^{m} \prod_{i \in \mathcal{T}_j} p_{\mathrm{j}} \mathrm{P}(t_i) \prod_{i \in \mathcal{T}_j^0} \frac{1}{\mathrm{P}(c_i = 0)} \prod_{i \in \mathcal{T}_j^1} \frac{1}{\mathrm{P}(c_i = 1)} \; . \tag{8.26}$$

From our model, $\mathrm{P}(c_i = 0) = \kappa_j$ *for* $i \in \mathcal{T}_j$*. Noting that all points must be in either class* $c = 0$ *or class* $c = 1$*,* $\mathrm{P}(c_i = 1) = 1 - \kappa_j$ *for* $i \in \mathcal{T}_j$*. Hence*

$$\begin{aligned}\mathrm{P}(\mathcal{T}) &= \prod_{j=1}^{m} p_{\mathrm{j}} \prod_{i \in \mathcal{T}_j} \mathrm{P}(t_i) \prod_{i \in \mathcal{T}_j^0} \frac{1}{\kappa_j} \prod_{i \in \mathcal{T}_j^1} \frac{1}{1-\kappa_j} \\ &= \prod_{j=1}^{m} p_{\mathrm{j}} \kappa_j^{-|\mathcal{T}_j^0|_0} (1-\kappa_j)^{-|\mathcal{T}_j^1|_0} \prod_{i \in \mathcal{T}} \mathrm{P}(t_i) \ .\end{aligned} \tag{8.27}$$

In order to select the model that best explains the training data, we select the values of κ_j *that maximise this probability:*

$$\begin{aligned}\frac{\partial \mathrm{P}(\mathcal{T})}{\partial \kappa_j} &= \left(-|\mathcal{T}_j^0|_0 \kappa_j^{-1} + |\mathcal{T}_j^1|_0 (1-\kappa_j)^{-1}\right) \mathrm{P}(\mathcal{T}) \\ &= 0 \textit{ to maximise } \mathrm{P}(\mathcal{T}) \ .\end{aligned} \tag{8.28}$$

Hence

$$|\mathcal{T}_j^1|_0 \kappa_j = |\mathcal{T}_j^0|_0 (1-\kappa_j) \tag{8.29}$$

and

$$\begin{aligned}\kappa_j &= \frac{|\mathcal{T}_j^0|_0}{(|\mathcal{T}_j^1|_0 + |\mathcal{T}_j^0|_0)} \\ &= \frac{|\mathcal{T}_j^0|_0}{|\mathcal{T}_j|_0}\end{aligned} \tag{8.30}$$

i.e. κ_j *is the proportion of training points within that segment for which the SWIPE′ pitch estimate is better than the YIN pitch estimate.* ▮

Given this model, and YIN and SWIPE′ pitch estimates, we can calculate $r = \frac{\omega_{\mathrm{YIN}}}{\omega'_{\mathrm{SWIPE}}}$ and look up the relevant $\mathrm{P}(e_{\mathrm{YIN}} > e_{\mathrm{SWIPE}'} | r) = \kappa_j : a_j \leq r < a_{j+1}$. If this probability is greater than 0.5, then it is likely that the YIN pitch estimate is worse than the SWIPE′ estimate and hence the SWIPE′ estimate should be chosen. If the probability is less than 0.5 then the YIN pitch estimate should be used. We refer to this combined estimator as “SWIN”.

8.2.1 The SWIN Estimator

Applying the SWIN estimator requires an initial training phase, in which the classifier “learns” how to select between the YIN and SWIPE′ pitch estimates. After learning the model for the data, SWIN can then be applied to new data samples.

Training SWIN

A training dataset was ordered by the period ratio $\frac{\omega_{\text{YIN}}}{\omega'_{\text{SWIPE}}}$ and divided into bins of 100 pitch estimates, the number of pitch estimates per bin being the only parameter specific to SWIN. The proportion of pitch estimates in each bin for which YIN provided a greater absolute error than SWIPE′ was then calculated, giving an estimate of the probability that the YIN error is larger than the SWIPE′ error for frequency ratios within that bin.

Applying SWIN

Given a test pitch sample, with YIN and SWIPE′ estimates, the appropriate period ratio bin was found and the YIN estimate selected when probability assigned to the bin was less than 0.5, the SWIPE′ estimate being selected otherwise.

8.2.2 Testing SWIN

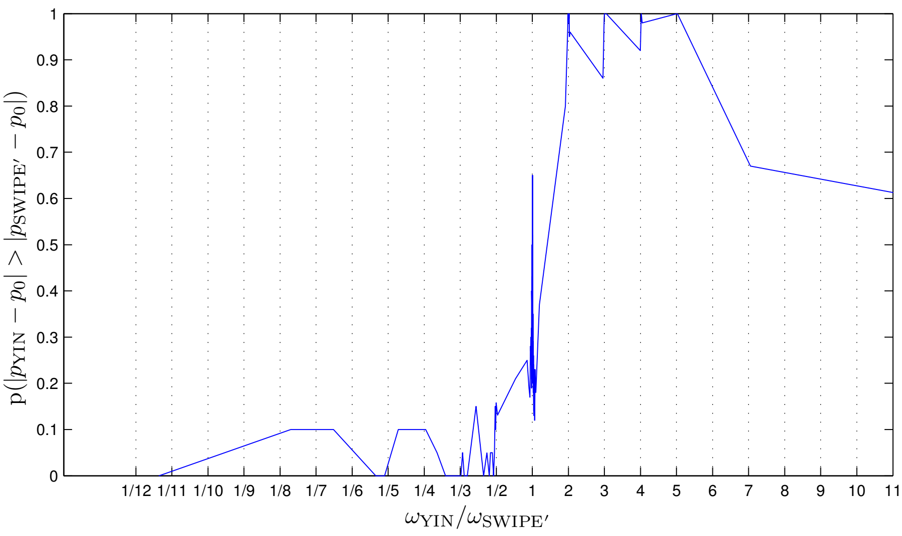
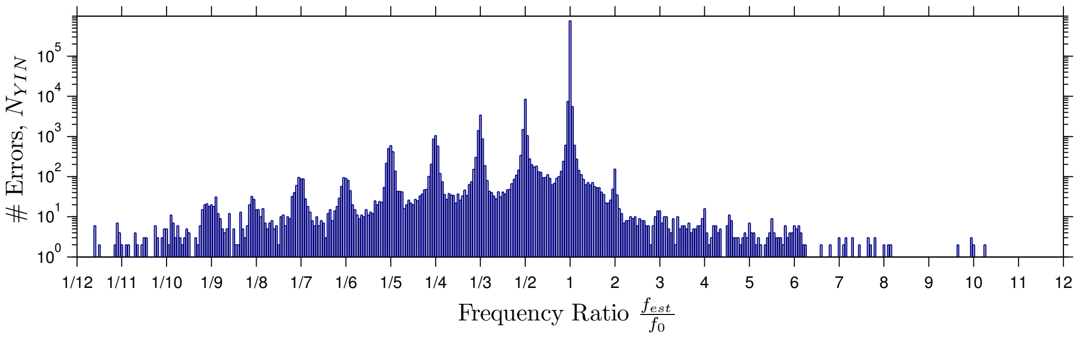
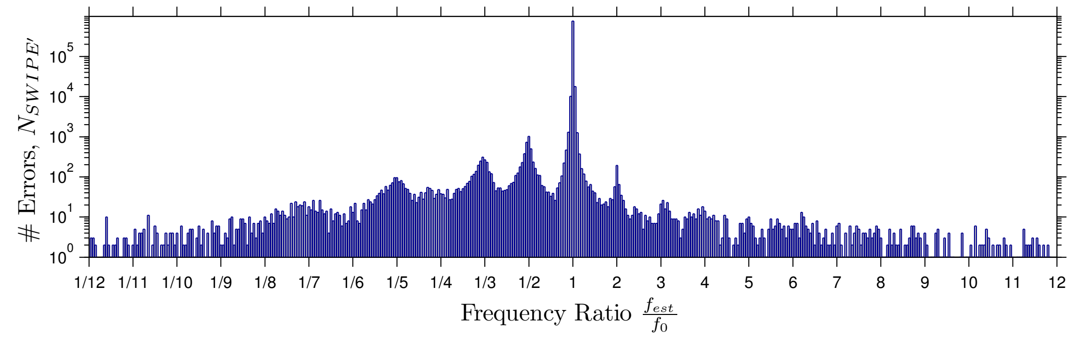


Figure 8.4: Model of MIREX QBSH 2003 data

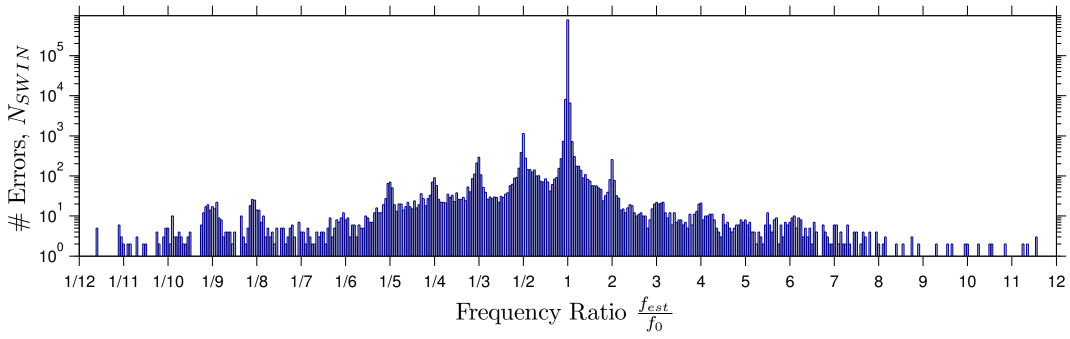
SWIN was trained on the 2003 data from the MIREX QBSH dataset, and then tested against data from the subsequent years. The distribution produced on the training data (Figure 8.4) shows that when the YIN period is an integer multiple of the SWIPE′ period ($\frac{\omega_{\text{YIN}}}{\omega_{\text{SWIPE}'}} \in \mathbb{Z}^+$), SWIPE′ always has a smaller error ($p = 1$) and for ratios close to these integer values SWIPE′ *usually* produces a smaller error ($0.5 < p < 1$). Conversely, sometimes when the SWIPE′ period is an integer multiple of the



(a) YIN

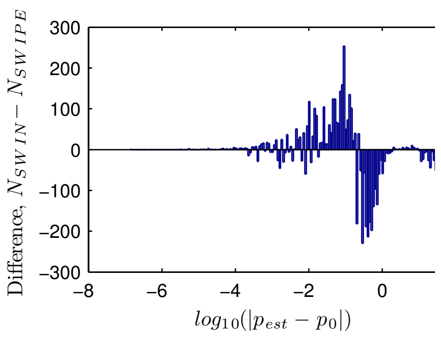


(b) SWIPE′

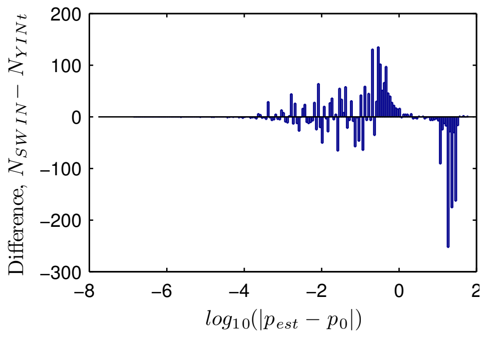


(c) SWIN

Figure 8.5: YIN, SWIPE′ and SWIN error distributions on the MIREX QBSH datasets.



(a) SWIPE′ and SWIN



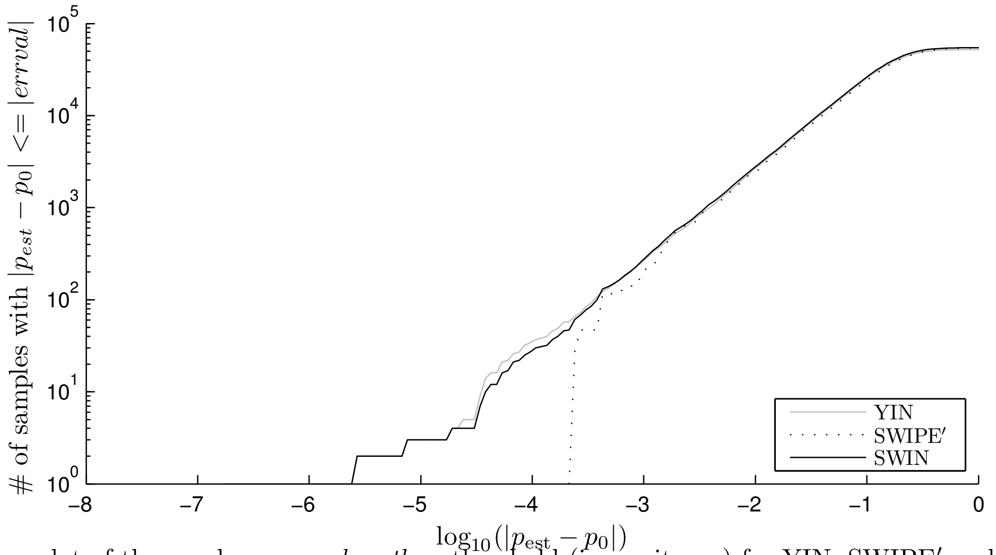
(b) YIN and SWIN

Figure 8.6: Difference between error histograms on the MIREX QBSH dataset from 2004 onwards (SWIN trained on MIREX QBSH 2003 data).

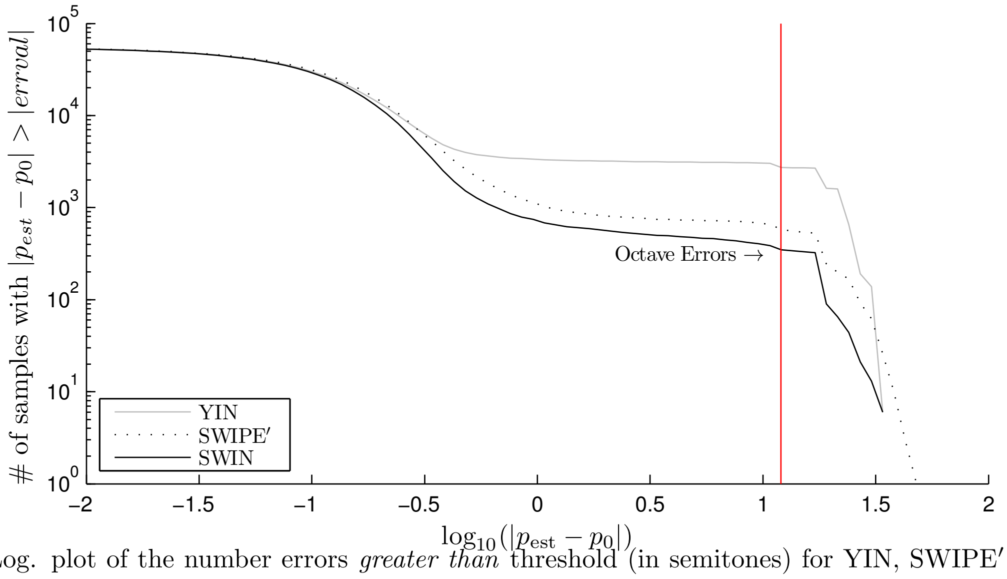
YIN period, YIN has the smaller error (dips to $p = 0$ ca. $\frac{\omega_{\text{YIN}}}{\omega_{\text{SWIPE}'}} \approx \frac{1}{2}, \frac{1}{3}, \frac{1}{5}$). Apart from a region of period ratios close to 1 (i.e. when both estimates are approximately equal), the classifier will select the algorithm that produces the lower pitch estimate. This is explained by the nature of the errors by period ratio. For the MIREX QBSH dataset (Figure 8.5), most YIN and SWIPE′ errors are frequency "too low" errors – the period estimates being greater than the actual period. Therefore, the estimator which makes the higher frequency estimate is, on average, likely to be closer to the correct frequency.

Testing SWIN on the MIREX QBSH data from 2004 onwards, SWIN reduces the number of errors greater than 0.1 semitones relative to the SWIPE′ results (Figure 8.6a); and removes many of the higher harmonic errors that occur with YIN (Figure 8.6b). SWIN has similar performance to YIN at low error values (Figure 8.7a); and outperforms both YIN and SWIPE′ regarding high error values (Figure 8.7b). Whilst YIN is slightly better than SWIN at very low error thresholds (< 0.1 semitones), SWIN is generally better than YIN. Although the SWIN pitch estimates are better than using either YIN or SWIPE′, harmonic errors are still apparent (Figure 8.5c) – these may occur when *both* YIN and SWIPE′ produce harmonic errors.

SWIN matched YIN performance in terms of matching the period to the nearest sample (PMP), again being better than SWIPE′ (Table 8.6). At higher thresholds, SWIN performance was better than YIN. For all four metrics, SWIN performance was better than SWIPE′. Simply selecting the higher pitched estimate from YIN and SWIPE′ gave a slightly lower GEP than YIN or SWIPE′ (0.93%), but did not match the performance of YIN at low error thresholds. The region of the model where YIN and SWIPE′ produced similar estimates ($\frac{\omega_{\text{YIN}}}{\omega_{\text{SWIPE}'}} \approx 1$) therefore successfully selected



(a) Log. plot of the number errors *less than* threshold (in semitones) for YIN, SWIPE′ and SWIN - emphasising behaviour at small error values. More errors to the left-hand side of the plot are *better*. SWIPE′ is worse than YIN or SWIN, fewer SWIPE′ errors occurring for thresholds less than 10^{-1} and the smallest SWIPE′ errors being ca. $10^{-3.7} \approx 2 \times 10^{-4}$. YIN is slightly better than SWIN clearly having more errors in the region around 10^{-4}.



(b) Log. plot of the number errors *greater than* threshold (in semitones) for YIN, SWIPE′ and SWIN– emphasising behaviour at large error values. Fewer errors to the right-hand side of the plot are *better*. SWIN is better than both YIN and SWIPE′ for thresholds greater than ca. 10^{-1}. For all three algorithms, there is a sharp dropoff in errors at ca. $10^{1.2} \approx 17$

Figure 8.7: Comparison of details of YIN, SWIPE′ and SWIN pitch errors for the MIREX QBSH dataset (SWIN trained on MIREX QBSH 2003, tested against MIREX QBSH 2004 onwards).

	YIN	SWIPE$'$	SWIN	Higher of YIN and SWIPE$'$ estimates	Better of YIN and SWIPE$'$ estimates
PMP	20.11%	23.64%	20.11%	21.77%	12.30%
10c	56.35%	58.62%	55.00%	57.55%	45.34%
MMP	19.49%	17.50%	15.64%	16.71%	13.39%
GEP	5.68%	1.37%	0.91%	0.93%	0.76%

Table 8.6: Summary of performance for YIN, SWIPE$'$ and SWIN on MIREX QBSH data 2004 onwards (SWIN trained on MIREX QBSH 2003). Additionally, statistics are given based on selecting the higher of the YIN and SWIPE$'$ estimates and selecting the better of the YIN and SWIPE$'$ estimates.

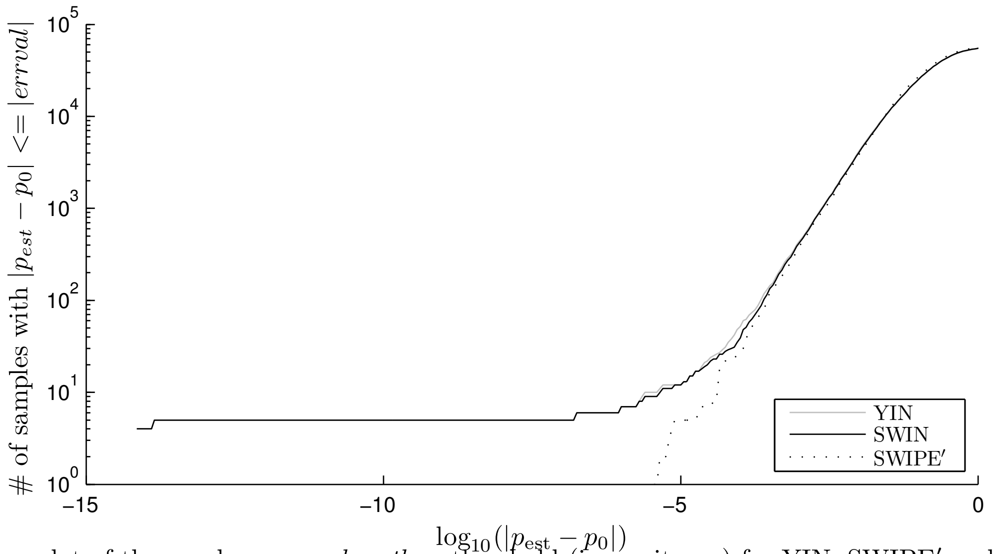
	YIN	SWIPE$'$	SWIN
PMP	65.61%	63.00%	65.61%
10c	59.59%	56.54%	59.34%
MMP	27.12%	24.23%	26.74%
GEP	3.57%	2.12%	2.86%

Table 8.7: Summary of proportion of errors greater than various thresholds (in pitch cents) on combined data with SWIN using a model trained on MIREX QBSH 2003.

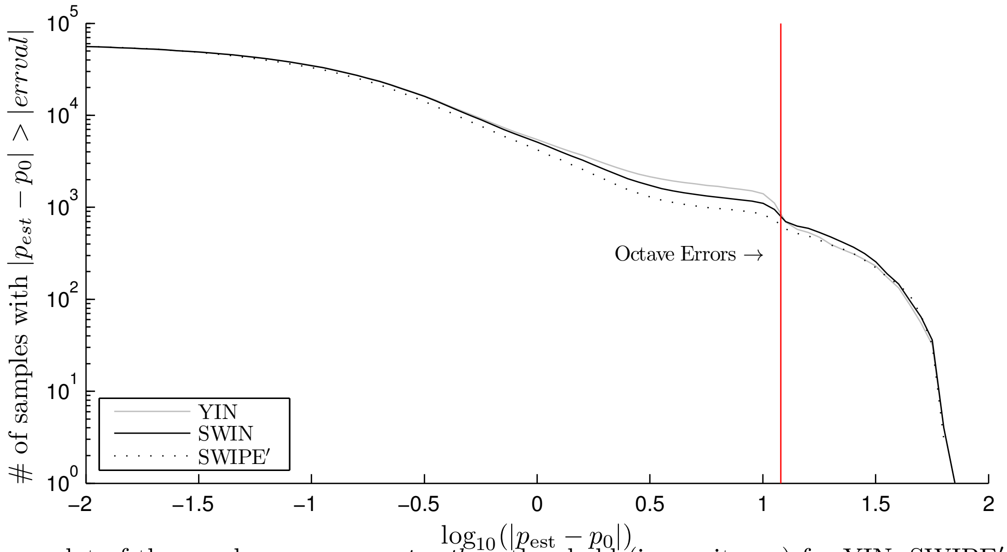
the better of the YIN and SWIPE$'$ estimates in some cases – for segments where $\kappa_j \neq 0$ and $\kappa_j \neq 1$ there will be some values for which the worse pitch estimate will be selected. The best possible performance by combining YIN and SWIPE$'$ output can be found by selecting the better of the two estimates. We see that although output could be improved further, simply using the best estimates still results in 45% of errors being greater than 10 pitch cents.

In order to assess whether the estimator based on MIREX QBSH data is applicable across other datasets, a combined dataset was created combining all pitch estimates from the Bagshaw, Keele and VOQUAL'03 datasets. Applying the model learnt from the 2003 MIREX QBSH data to the combined dataset produced results between those of SWIPE$'$ and YIN (Table 8.7, Figure 8.8). As SWIPE$'$ performed better on average for this dataset apart from for small thresholds ($< 10^{-1}$, Figure 8.8a) it is not unexpected that choosing between the YIN and SWIPE$'$ pitch estimates made performance worse for thresholds of 10 pitch cents and above.

We created a model for SWIN from 10% of the combined dataset (Figure 8.9), and tested it on the remaining 90% of the data. This version of SWIN matched the performance of YIN at small error thresholds and performance was close to that of SWIPE$'$ at high thresholds (Table 8.8). However, SWIPE$'$ performance was still



(a) Log. plot of the number errors *less than* threshold (in semitones) for YIN, SWIPE′ and SWIN - emphasising behaviour at small error values. More errors to the left-hand side of the plot are *better*.



(b) Log. plot of the number errors *greater than* threshold (in semitones) for YIN, SWIPE′ and SWIN– emphasising behaviour at large error values. Fewer errors to the right-hand side of the plot are *better*.

Figure 8.8: Comparison of details of YIN, SWIPE′ and SWIN pitch errors for the combined dataset (SWIN trained on MIREX QBSH 2003, tested against MIREX QBSH 2004 onwards). In both cases, SWIN performance is between that of YIN and SWIPE′.

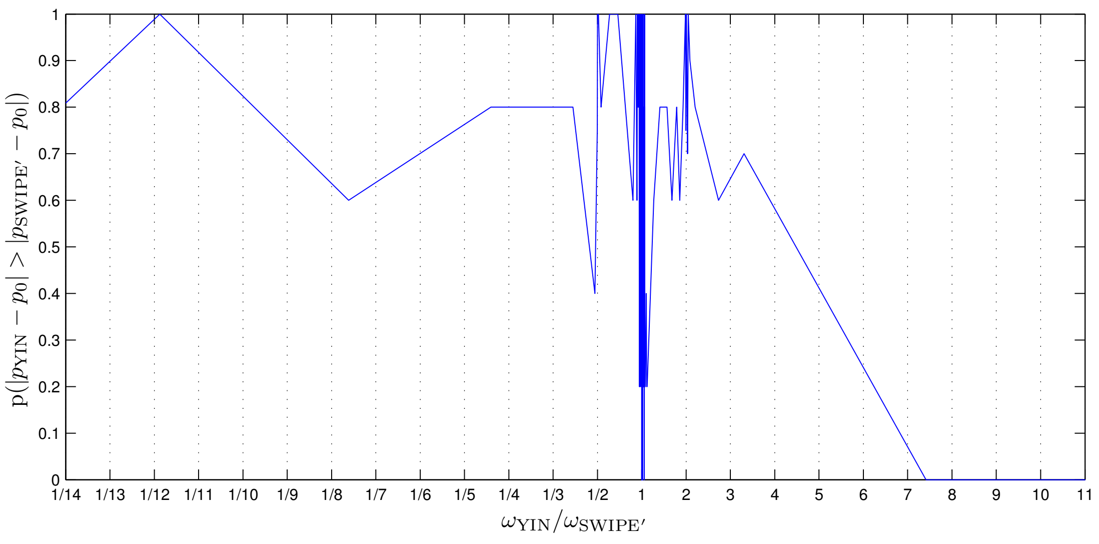


Figure 8.9: SWIN model of combined Bagshaw, Keele and VOQUAL'03 data

better than SWIN. We hypothesise that this is caused by the three datasets combined for this having fewer harmonic errors than the QBSH dataset (Figures 7.3 and 8.2) and therefore not matching our harmonic model.

	YIN	SWIPE′	SWIN
PMP	65.53%	62.99%	65.53%
10c	59.56%	56.59%	57.06%
MMP	27.16%	24.28%	24.68%
GEP	3.53%	2.12%	2.28%

Table 8.8: Summary of proportion of errors greater than various thresholds (in pitch cents) on 90% of the combined data with SWIN using a model trained on the remaining 10% .

8.3 Conclusions

We compared the performance of the YIN pitch estimator with PRAAT AC-P and SWIPE′. In general, SWIPE′ performed better than YIN, which performed better than PRAAT AC-P. We observed that YIN produced more harmonic errors than SWIPE′ on the QBSH dataset. However, when YIN produced good pitch estimates these were generally closer to the ground-truth pitch than those produced by SWIPE′.

Having examined additional pitch estimators, we concluded that none of them provided accurate, high-resolution pitch estimates. The three pitch estimators considered (PRAAT AC-P, YIN and SWIPE′) achieved at best 82.26% matches of the period in samples (Table 8.4) and then only on low sample-rate data from the MIREX QBSH dataset).

Given that both YIN and SWIPE$'$ showed benefits for pitch estimation and that YIN produced harmonic errors on the QBSH dataset, we examined whether it was possible to combine these benefits by using a classifier to select either the YIN or SWIPE estimate based on the ratios of their frequency outputs.

We created a new pitch estimator ("SWIN") which treated YIN and SWIPE as "expert" pitch estimators, and selected the appropriate expert pitch estimate based on a classifier, the classifier being trained on a set of training data. Training SWIN on data from one year of the QBSH dataset and testing it on the remaining QBSH data, SWIN performed better than both YIN and SWIPE for large errors thresholds and produced results close to YIN at small thresholds. The Gross Error Percentage (GEP) for SWIN on the QBSH dataset was 0.91%, whereas the GEP for YIN was 5.68% and the GEP for SWIPE$'$ was 1.37%. However, on the other datasets fewer harmonic errors were observed, and SWIN did not improve results over those from SWIPE$'$. Including alternative "expert" pitch estimators and examining alternative models of the relationship between estimator results may allow additional improvements to the art of pitch estimation.

Chapter 9

Conclusions

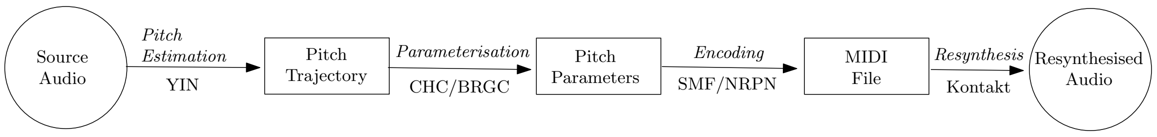


Figure 9.1: System Model indicating the tools used at each stage

In this work, we have produced a novel Expressive MIDI system (Figure 9.1) for resynthesising audio based on the pitch trajectory of source audio material.

In Chapter 2 we considered a selection of audio coding techniques and identified MPEG-4 Structured Audio (SA) Wavetable Synthesis as a possible format with which to build an object coding system for musical audio. MPEG-4 SA Wavetable Synthesis combines a Structured Audio Sample Bank Format (SASBF) (aka MMA DLS) synthesiser definition with a MIDI score to produce synthetic audio. We defined an Expressive MIDI model for object coding based on MPEG-4 SA Wavetable Synthesis in Chapter 3. This encoded individual note pitch trajectories for use with the Downloadable Sounds (DLS) pitch model – combining a base pitch with an EG and an LFO.

We proceeded to build an Expressive MIDI system based on existing technologies. As the first stage in the system, we extracted pitch trajectories from audio using the YIN [de Cheveigné and Kawahara, 2002] pitch estimator and segmented this data into individual notes. We then examined possible optimisation algorithms for inferring Expressive MIDI parameters from the note pitch trajectories (Chapter 4), specifically Steepest Descent, a Stochastic Hill-Climber (SHC), the Simple Genetic Algorithm and Eshelman's CHC. We considered several bit-wise encodings for use in optimisation and their properties, introducing a new visualisation that gave insights into the effects of the encodings on optimisation problems.

Extending the work of Mathias and Whitley [1994] to include Balanced and Monotone Gray codes, we examined the performance of algorithm/encoding combinations across a set of standard test problems and found that the choice of a suitable encoding allowed simple techniques to solve otherwise difficult problems (e.g. in general, bit-wise Steepest Descent worked most effectively using the Balanced Gray code). We hypothesised that as each bit in the Balanced Gray code has a similar number of transitions, it gave a more uniform distribution of neighbours across the search space and is therefore particularly effective at exploring the search space.

CHC was by far the most effective optimisation technique for the test problems – finding solutions for most problems from a single start. CHC performance using the Balanced Gray Code almost matched that with the Binary Reflected Gray Code (BRGC). Given that the BRGC is simple to generate and use, we adopted the CHC algorithm [Eshelman, 1991] with a BRGC encoding as the optimisation algorithm to infer Expressive MIDI parameters from note pitch trajectories.

In Chapter 5 we estimated Expressive MIDI parameters from audio files from the RWC Musical Instrument Sounds database using the CHC/BRGC optimisation. We found that the Expressive MIDI parametrisation closely matched the pitch trajectories for many notes, more closely approximating the original audio than simply using a constant pitch. The Expressive MIDI pitch parameters provided a smoothed approximation of the pitch trajectory and often bore little resemblance to that expected from a “standard” EG+LFO trajectory – the LFO often being used for the gross trajectory shape rather than representing vibrato effects. With vibrato examples, both frequency and phase were often matched. However, segmentation of the audio was imperfect and harmonic errors were observed in the pitch output, both causing outliers in the pitch trajectories and producing a more difficult optimisation problem.

We completed our Expressive MIDI system in Chapter 6, selecting to resynthesise audio using Native Instruments Kontakt, a software sampler, and storing the Expressive MIDI data in a Standard MIDI File (SMF) using NRPNs. This gave a low bit-rate representation of less than 100 bytes per note. We created MIDI encodings of the Expressive MIDI parameters estimated from the RWC data and found that resynthesised trajectories were often within 10 pitch cents of the Expressive MIDI model and closely resembled the original pitch trajectories. This in spite of differences between the Expressive MIDI trajectory model and that used by Kontakt. The close match between the output pitch trajectories produced using Kontakt and the expected trajectories based on the EG+LFO parameters gave confidence that, by estimating parameters that match the Kontakt model, we could produce audio which closely matched the parameterised pitch trajectory.

Small pitch differences observed between the resynthesised audio and the expected values could have been caused by either differences between the pitch of the Kontakt output and the expected pitch or by artifacts produced in the YIN pitch estimation. Whichever was the actual cause of the pitch difference, Kontakt or YIN, we needed to assess the quality of pitch estimation.

In Chapter 7 we examined the quality of the YIN pitch estimates in detail. After introducing several pitch datasets and pitch metrics, we presented a novel analysis of the performance of the YIN pitch estimator using both traditional metrics and by examining the pitch error distributions. The error distributions revealed that most YIN errors occur at approximately the 10 pitch cent level, the vast majority of errors being in the range 1 cent to 100 cents (1 semitone). Previously, in Chapter 5, comparing the median YIN pitch for each note with the YIN pitch trajectory had revealed the presence of harmonic errors in the YIN output. We therefore presented a theoretical analysis of the effect of removing such errors from the YIN output, noting that even if the harmonic errors could be removed, residual pitch errors larger than required for Expressive MIDI would still occur.

In order to produce a better Expressive MIDI representation of audio, we therefore looked for a better pitch estimator than YIN. In Chapter 8 we compared the performance of YIN with the PRAAT autocorrelation based pitch estimator (AC-P) [Boersma, 1993] and the prime harmonic variant SWIPE′ of the Sawtooth Wave Inspired Pitch Estimator (SWIPE) [Camacho, 2007]. Both SWIPE′ and YIN produced better results than AC-P, on average producing smaller errors.

Noting that SWIPE′ produced fewer *harmonic* errors than YIN, whilst YIN produced a larger number of *very small* errors than SWIPE′, we created a new “best of both worlds” pitch estimator, SWIN, which automatically selected the YIN or SWIPE′ pitch estimate after training on a sample of labelled audio data. We showed that on datasets where YIN was particularly prone to harmonic errors SWIN produced better results – with a 20% threshold, the Gross Error Percentage (GEP) for SWIN on the QBSH dataset was 0.91%, whereas the GEP for YIN was 5.68% and for SWIPE′ it was 1.37%. However, on other datasets fewer harmonic errors were observed, and SWIN did not improve results over those from SWIPE′. Including alternative “expert” pitch estimators and examining alternative models of the relationship between estimator results may allow additional improvements to the art of pitch estimation.

9.1 Contributions

We developed a new system for Expressive MIDI coding of audio and, in the process:

- produced an extended analysis of optimisation methods and binary codings emphasising that appropriate codings can make problems solvable using simple optimisation techniques such as Steepest Descent or a Stochastic Hill Climber (Chapter 4);
- developed a process for estimating parameters for the Expressive MIDI pitch model using CHC with BRGC encoding (Chapter 5);
- showed that it is possible to resynthesise audio from a MIDI representation of the estimated pitch parameters to within 10 pitch cents of the estimate (Chapter 6);

In a new examination of the performance of pitch estimators (Chapters 7 and 8) we:

- found that the Gross Error Percentage metric for judging pitch estimators was insufficient at a 20% threshold for distinguishing the suitability of pitch estimators for high resolution pitch estimation – hence we introduced a new view of pitch estimator performance based on the *distribution* of pitch errors;
- showed that even if problems with harmonic errors could be solved, residual inharmonic errors prevent the current generation of pitch estimators from providing high resolution pitch estimates;
- created a novel pitch estimator, SWIN, which,after training on a subset of the data, combined the benefits of the YIN and SWIPE$'$ pitch estimators on data prone to harmonic errors.

9.2 Future Work

There is scope for additional work in improving the output of this Expressive MIDI system.

- With complex pitch variation, the Expressive MIDI parameters sometimes used the LFO to represent trajectory coarse features rather than representing any vibrato. In order to capture the vibrato effect, we propose a two stage process: first using CHC to estimate gross features of the pitch trajectory with the EG; and then separately finding the LFO parameters to best represent the residual pitch trajectory unaccounted for by the EG (e.g. using autocorrelation on the residual pitch trajectory after the EG estimate is used to estimate the LFO frequency and subsequently estimating the LFO delay using cross-correlation of the residual trajectory and a "test" LFO based on a zero-padded sine wave).

- Segmentation errors resulted in sudden pitch changes which were poorly represented using our model. Attempting estimation of these features reduced performance of the pitch trajectory parameter estimation. Additional work is therefore required to improve the segmentation of the audio. For the RWC Musical Instrument Sounds database, each individual note is separated by a period of silence. Better segmentation can therefore be achieved by: splitting the audio into segments at the silences and then refining those segments using power and/or aperiodicity thresholds to refine the note boundaries. However, for arbitrary audio segmentation is much more complex.

- Similarly, when outliers and harmonic errors occurred in the YIN pitch estimates, the square error cost function produced pressure on the system to match these variations in the pitch trajectories. Improved segmentation will help, but using an absolute difference (L_1 norm, $|\cdot|$) cost function would reduce the pressure to match these variations and may produce a more accurate estimate of the smaller variations in pitch across the note.

- Although Kontakt was the closest synthesiser to the Expressive MIDI model, Kontakt's synthesis model was not the same as the one selected for Expressive MIDI i.e. the resynthesis model differed from that used in the CHC analysis stage. Matching the analysis and synthesis models should produce a closer match between the resynthesised audio the analysis results. The analysis stage should therefore be updated to agree with the resynthesis stage, optimising Kontakt parameter values directly. Additionally, this would lead to the representation of the parameters in CHC being reduced from 32-bits to agree with the 14-bit NRPN values. This reduction in dimensionality would reduce the size of the search space and should improve CHC performance and the output pitch parameters.

Additionally, the system could be extended to incorporate further audio features e.g. the EG + LFO model is not just available for pitch, but also for gain control. It may also be possible to estimate suitable samples for use in the resynthesis from the source audio material.

Pitch estimation did not provide the accuracy we hoped for and prevented us producing precise judgements of the quality of the output. There is still much room for improvement in monophonic pitch estimation.

- Existing pitch databases largely include the ground-truth pitch based on estimates of the period to the nearest sample. This limits the level at which pitch estimators can be tested.
- The SWIN algorithm improved pitch estimation results for a specific class of data. Examining the content of the signal, it may also be possible to classify pitch data and select an appropriate model for pitch estimation other than the harmonic model assumed by SWIN.
- Examining the various techniques used by pitch estimation algorithms for interpolating their results, weighting pitch detection function output and peak/best path picking may provide further insights into combining current pitch estimators – possibly suggesting additional "experts" to add to the SWIN model.

9.3 Closing Remarks

As initially intended, we have produced an Expressive MIDI system which can create a pitch based MIDI sketch of source audio material. Output pitch trajectories closely resembled those of the source material, but led us to consider just what the current state-of-the-art is for pitch estimation.

We found that most of the literature on pitch estimation used the Gross Error Percentage (GEP) at a 20% difference in frequency. This does not appear to be suitable for measuring the performance of state-of-the-art algorithms, and is, additionally, inappropriate for estimating pitch in musical applications as pitch errors of 3 semitones are classified as "correct". In order to advance pitch estimation for music, it is necessary to move away from the GEP and consider alternative performance measures.

Appendix A

Scripting Kontakt

A.1 Validating Kontakt Parameter Formulae

The “KSP Math Library Technical Manual” [Villwock, 2009] gives reverse-engineered conversion functions to convert Kontakt script parameter values into appropriate units. We set the internal Kontakt parameters to various values and recorded the values displayed in the Kontakt interface (in appropriate units) (Table A.1). These parameter and display values were then compared (below). For the LFO frequency and sustain level, we derived formulae for the relationships between the parameter values and the displayed settings.

Kontakt Parameter Value	Frequency (Hz)	LFO Fade (ms)	LFO Depth (st)	EG Attack (s)	EG Decay (s)
0	0.01	0	-12	0	0
100000	0.03	2.7	-6.1	0.0029	0.0031
200000	0.08	9	2.6	0.0099	0.0112
300000	0.2	23.7	-0.77	0.0271	0.0319
400000	0.55	58.4	-0.1	0.0689	0.0851
500000	1.5	139.4	0	0.1711	0.2216
600000	4	329.5	0.1	0.4208	0.5724
700000	10.8	775	0.77	1	1.5
800000	29.3	1.8k	2.6	2.5	3.8
900000	78.6	4.3k	6.1	6.1	9.7
1000000	213.1	10.0k	12	15	25

Table A.1: Kontakt parameter settings and the related values as displayed in Kontakt

A.1.1 Kontakt LFO Frequency

We noted a logarithmic relationship between the Kontakt LFO frequency parameter value and the frequency in Hz. The best-fit linear approximation of the relationship between the logarithm of the parameter value, $\log_{10}(v)$ and the displayed LFO frequencies gave the following relationship:

$$f_{\text{LFO}} = 0.2318 \times \log_{10}\left(\frac{v}{1000000}\right) + 0.4598 \tag{A.1}$$

For low parameter values (< 500000) this gave errors of up to 5% in the frequency estimates, for higher values the errors were less than 1%.

A.1.2 Kontakt Depths

From Villwock [2009], The depth values, d, are given (in semitones) by

$$d = 12\left(\frac{v - 500000}{500000}\right)^3 . \tag{A.2}$$

This gave errors of less than 0.1% when comparing calculated depths with displayed depths.

A.1.3 Kontakt Timings

Based on the formulae in Villwock [2009], we found a single formula which relates the Kontakt timing parameters with their values in milliseconds:

$$t = 2\left(1 + \frac{t_{\max}}{2}\right)^{\frac{v}{1000000}} - 2 \tag{A.3}$$

where $t_{\max}$ is the maximum time value for the parameter in question (10 seconds for LFO fade, 15 seconds for EG attack and hold, 25 seconds for EG decay and release). This gave low errors ($< 1\%$ of timings) for the tested parameter values except for ca. 700000. However at 700000 Kontakt switches from displaying times in milliseconds with 1 decimal place to displaying times in seconds with one decimal place. The precision of the displayed timings of 1 and 1.5 is therefore approximately $\pm 5\%$ and the calculated errors are less than this value.

Kontakt Value	Kontakt dB	Calculated dB	Ratio
v	d_{K}	$d_{\mathrm{C}} = 20\log_{10}\left(\frac{v}{1000000}\right)$	$\frac{d_{\mathrm{K}}}{d_{\mathrm{C}}}$
100000	-59.8	-20.000000	2.990000
200000	-41.8	-13.979400	2.990114
225001	-38.7	-12.956311	2.986961
330001	-28.8	-9.629695	2.990749
435000	-21.6	-7.230215	2.987463
500000	-18	-6.020600	2.989735
535000	-16.2	-5.432924	2.981820
591605	-13.6	-4.559363	2.982873
665000	-10.6	-3.543567	2.991336
755000	-7.3	-2.441061	2.990503
795000	-6	-1.992657	3.011054
835000	-4.7	-1.566270	3.000759
924999	-2	-0.677175	2.953447
938636	-1.6	-0.550056	2.908795

Table A.2: Kontakt Sustain Parameters Expressed as Decibels

A.1.4 Kontakt Sustain Level

Within Kontakt, sustain levels are displayed in terms of decibels (dB). The formula for expressing an amplitude ratio r as a decibel value, d, is $d = 20\log_{10}(r)$. We examined a range of Kontakt parameter values and the decibel values produced and compared them with the output of this formula (Table A.2). In each case, the ratio of the Kontakt dB values and the calculated values is approximately 3. The relationship which we used to relate the Kontakt parameter value, v, to the sustain level, s , was:

$$s = \left(\frac{v}{1000000}\right)^3 \tag{A.4}$$

. The differences between the sustain levels calculated using this formula and the displayed values in Kontakt were less than 0.1dB apart from at sustain levels less than -18dB (i.e. sustain depths less than $\approx \frac{1}{8}$). As sustain levels are displayed to a precision of 0.1dB, we believe this formula to accurately map Kontakt parameter values to sustain levels.

Listing A.1: Sample Kontakt code for controlling LFO delay parameter

```
1  on init
2      declare $newval
3  end on
4
5  on nrpn
6      $newval := ($RPN_VALUE * 61)
7      select ($RPN_ADDRESS)
8          case 1001
9              _set_engine_par($ENGINE_PAR_LFO_DELAY, $newval, 0, 0, 0)
10     end select
11 end on
```

A.2 Kontakt Instrument Script

Listing A.1 shows a sample of Kontakt script (KSP) which links NRPN 1001 to the LFO delay parameter. Lines 1-3 declare numeric variable $newval for handling NRPNs. Lines 5-11 allow the LFO delay to be controlled using NRPN 1001. 14-bit NRPN values (i.e. values 0-16383) are scaled by a factor of 61 (line 6) to approximate the Kontakt parameter range of 0 to 1000000 (actually 0 to 999363). The appropriate engine parameter is then set to that value (line 9 for NRPN 1001).

Parameter values are set using:

_set_engine_par(<p>, <v>, <g>, <m>, <t>)

where: <p> is the engine parameter; <v> the new value for the parameter; <g> the modulator group; <m> the modulator within that group; and <t> which target for that modulator is required (e.g. <t> allows separate levels if a single EG controls both pitch and amplitude modulation). For this bank of sounds, there is one group (0), the LFO being modulator 0 within that group, and the EG modulator 1. Both the EG and LFO only control pitch, so the target is 0.

The KSP script we used within Kontakt to convert NRPN values to internal Kontakt parameter settings (Section 6) follows in three parts: listing A.2 initialises the script and sets up user interface controls to display and set parameter values; listing A.3 links those controls to the internal Kontakt parameters; and listing A.4 sets the controls based on NRPNs received.

Listing A.2: Interface Setup and Initialisation

```
{Use onscreen controls to modify parameters}
on init
    declare $lfomod := 0
    declare $egmod := 1
    {LFO controls}
    declare ui_knob $lfodelay (0, 1000000, 1000000)
    declare ui_knob $lfofreq (0, 1000000, 1000000)
    declare ui_knob $lfodepth (0, 500000, 500000)
    {EG controls - dAhDSR, but no d available}
    declare ui_knob $egatk (0, 1000000, 1000000)
    declare ui_knob $eghld (0, 1000000, 1000000)
    declare ui_knob $egdec (0, 1000000, 1000000)
    declare ui_knob $egsus (0, 1000000, 1000000)
    declare ui_knob $egrel (0, 1000000, 1000000)
    declare ui_knob $egdepth (0, 500000, 500000)
    {Informatioon panel}

    declare $newval
    $lfodelay := _get_engine_par($ENGINE_PAR_LFO_DELAY,0,$lfomod,0)
    $lfofreq := _get_engine_par($ENGINE_PAR_INTMOD_FREQUENCY,0,$lfomod,0)
    $lfodepth := _get_engine_par($ENGINE_PAR_INTMOD_INTENSITY,0,$lfomod,0)
    $egdepth := _get_engine_par($ENGINE_PAR_INTMOD_INTENSITY,0,$egmod,0)
    $egatk := _get_engine_par($ENGINE_PAR_ATTACK,0,$egmod,0)
    $eghld := _get_engine_par($ENGINE_PAR_HOLD,0,$egmod,0)
    $egdec := _get_engine_par($ENGINE_PAR_DECAY,0,$egmod,0)
    $egsus := _get_engine_par($ENGINE_PAR_SUSTAIN,0,$egmod,0)
    $egrel := _get_engine_par($ENGINE_PAR_RELEASE,0,$egmod,0)
    make_perfview
end on
```

Listing A.3: Linking onscreen controls to parameters

```
on ui_control ($lfodelay)
    _set_engine_par($ENGINE_PAR_LFO_DELAY, $lfodelay,0,$lfomod,0)
end on
on ui_control ($lfofreq)
    _set_engine_par($ENGINE_PAR_INTMOD_FREQUENCY, $lfofreq,0,$lfomod,0)
end on
on ui_control ($lfodepth)
    _set_engine_par($ENGINE_PAR_INTMOD_INTENSITY, $lfodepth+500000,0,$lfomod,0)
end on
on ui_control ($egatk)
    _set_engine_par($ENGINE_PAR_ATTACK, $egatk,0,$egmod,0)
end on
on ui_control ($eghld)
    _set_engine_par($ENGINE_PAR_HOLD, $eghld,0,$egmod,0)
end on
on ui_control ($egdec)
    _set_engine_par($ENGINE_PAR_DECAY, $egdec,0,$egmod,0)
end on
on ui_control ($egsus)
    _set_engine_par($ENGINE_PAR_SUSTAIN, $egsus,0,$egmod,0)
end on
on ui_control ($egrel)
    _set_engine_par($ENGINE_PAR_RELEASE, $egrel,0,$egmod,0)
end on
on ui_control ($egdepth)
    _set_engine_par($ENGINE_PAR_INTMOD_INTENSITY, $egdepth+500000,0,$egmod,0)
end on
```

Listing A.4: Responding to NRPNs

```
on nrpn
    {Convert NRPN values received to changes for onscreen controls}
    {Force 0 => 0, 8192 => 500000, 16383 => 1000000}
    if ($RPN_VALUE < 8192)
        $newval := ($RPN_VALUE * 61)
    else
        if ($RPN_VALUE > 8192)
            $newval := 1000000 - ((16383 - $RPN_VALUE) * 61)
        else
            $newval := 500000
        end if
        $newval := ($RPN_VALUE - 8192) * 61 + 500000
    end if
    select ($RPN_ADDRESS)
        case 1001
            $lfodelay := $newval
            _set_engine_par($ENGINE_PAR_LFO_DELAY, $lfodelay,0,$lfomod,0)
        case 1002
            $lfofreq := $newval
            _set_engine_par($ENGINE_PAR_INTMOD_FREQUENCY, $lfofreq,0,$lfomod,0)
        case 1003
            $lfodepth := $newval / 2
            _set_engine_par($ENGINE_PAR_INTMOD_INTENSITY, $lfodepth+500000,0,$lfomod,0)
        case 1004
            $egatk := $newval
            _set_engine_par($ENGINE_PAR_ATTACK, $egatk,0,$egmod,0)
        case 1005
            $eghld := $newval
            _set_engine_par($ENGINE_PAR_HOLD, $eghld,0,$egmod,0)
        case 1006
            $egdec := $newval
            _set_engine_par($ENGINE_PAR_DECAY, $egdec,0,$egmod,0)
        case 1007
            $egsus := $newval
            _set_engine_par($ENGINE_PAR_SUSTAIN, $egsus,0,$egmod,0)
        case 1008
            $egrel := $newval
            _set_engine_par($ENGINE_PAR_RELEASE, $egrel,0,$egmod,0)
        case 1009
            $egdepth := $newval / 2
            _set_engine_par($ENGINE_PAR_INTMOD_INTENSITY, $egdepth+500000,0,$egmod,0)
    end select
end on
```

Bibliography

Innovative Music Systems, Inc. intelliScore 8.0 ensemble Wav to MIDI converter. [Computer program] Link valid 23rd September 2010. Available from http://www.intelliscore.net.

Akoff Sound Labs. Akoff music composer version 2.0. [Computer program] Link valid 23rd September 2010. Available from http://www.akoff.com.

Apple. *Audio Interchange Format: "AIFF"*. Apple Computer Inc., January 1989.

P. C. Bagshaw, S. M. Hiller, and M. A. Jack. Enhanced pitch tracking and the processing of f0 contours for computer aided intonation teaching. In *Proceedings of the 3rd European Conference on Speech Communication and Technology (EUROSPEECH '93)*, pages 1003–1006, Berlin, Germany, September 1993.

J. Beament. *How We Hear Music: The Relationship between Music and the Hearing Mechanism*. Boydell Press, Woodbridge, UK, 2001.

J. W. Beauchamp. Synthesis by amplitude and "Brightness" matching of analyzed musical instrument tones. In *Proceedings of the 69th Convention of the Audio Engineering Society*, Los Angeles, California, USA, May 1981. AES.

F. Beritelli, S. Casale, and M. Russo. A pattern classification proposal for object-oriented audio coding in MPEG-4. *Telecommunication Systems*, 9(3):375–391, 1998.

N. Bertin, R. Badeau, and G. Richard. Blind signal decompositions for automatic transcription of polyphonic music: NMF and K-SVD on the benchmark. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2007)*, volume 1, pages I–65–I–68, Honolulu, Hawai'i, USA, April 2007.

G. S. Bhat and C. D. Savage. Balanced Gray codes. *The Electronic Journal of Combinatorics*, 3(R25):2, 1996.

P. Boersma. Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. In *Proceedings of the Institute of Phonetic Sciences*, volume 17, pages 97–110, Amsterdam, 1993.

P. Boersma and D. Weenink. *Praat: doing phonetics by computer (version 5.1.23)*, 2010. [Computer Program] Retrieved 2nd January 2010 from http://www.praat.org.

R. Boulanger. *The CSound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing and Programming.* The MIT Press, Cambridge, Massachusetts, USA, 2000.

K. Brandenburg. MP3 and AAC explained. In *Proceedings of 17th International Conference of the Audio Engineering Society*, Florence, Italy, September 1999. AES.

K. Brandenburg, O. Kunz, and A. Sugiyama. MPEG-4 natural audio coding. *Signal Processing: Image Communication*, 15(4-5):423–444, 2000.

R.P. Brent. *Algorithms for Minimization without Derivatives.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1973.

R.L. Burden and J.D. Faires. *Numerical Analysis.* Brooks/Cole Publishing Company, Pacific Grove, CA, USA, sixth edition, 1997.

A. Camacho. *SWIPE: A Sawtooth Waveform Inspired Pitch Estimator for Speech and Music.* PhD thesis, University of Florida, FL, USA, December 2007.

A. Camacho and J. G. Harris. A sawtooth waveform inspired pitch estimator for speech and music. *The Journal of the Acoustical Society of America*, 124(3):1638–1652, 2008.

R. Caruana and J. D. Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proceedings of the 5th International Workshop on Machine Learning*, pages 153–161, Ann Arbor, Michigan, USA, June 1988.

Celemony. Melodyne editor 3.2. [Computer program] Link valid 23rd September 2010. Available from http://www.celemony.com.

U. K. Chakraborty and C. Z. Janikow. An analysis of Gray versus binary encoding in genetic search. *Information Sciences*, 156(3-4):253–269, 2003.

Wei-Chen Chang and A. W. Y. Su. A multi-channel recurrent network for synthesizing struck coupled-string musical instruments. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, page 677–686, Martigny, Switzerland, September 2002.

S.S. Chen. *Basis Pursuit.* PhD thesis, Stanford University, CA, USA, November 1995.

J. M. Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Soc*, 21(7):526–534, 1973.

Josh Coalson. *FLAC: Free Lossless Audio Codec*, 2000. [Online] Valid 23 September 2010. Available at http://flac.sourceforge.net.

J. Copp. Audio for a mobile world. *Communications Engineer*, 1(2):26–29, 2003.

A. de Cheveigné and A. Kawahara. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930, 2002.

K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

S. Dixon. Extraction of musical performance parameters from audio data. In *Proceedings of the 1st IEEE Pacific Rim Conference on Multimedia (PCM 2000)*, page 42–45, Sydney, Australia, December 2000.

B. Edler, H. Purnhagen, and C. Ferekidis. ASAC—Analysis/synthesis audio codec for very low bit rates. In *Proceedings of the 100th Convention of the Audio Engineering Society*, Copenhagen,Denmark, May 1996. AES.

A. E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998.

L. J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, USA, 1991.

J. ffitch. CSound5: the design, the outcome, the experience. *CSound Journal*, 1(2), 2006.

H. Fletcher and W. A. Munson. Loudness of a complex tone, its definition, measurement and calculation. *The Journal of the Acoustical Society of America*, V(2):82–108, October 1933.

S. Forrest and M. Mitchell. Relative Building-Block Fitness and the Building-Block Hypothesis. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126. Morgan Kaufmann, San Mateo, CA, USA, 1993.

R. Geiger, Rongshan Yu, J. Herre, S. Rahardja, Sang-Wook Kim, Xiao Lin, and M. Schmidt. ISO/IEC MPEG-4 High-Definition scalable advanced audio coding. *Journal of the Audio Engineering Society*, 55(1/2):27–43, February 2007.

A. Glass and K. Fukudome. Warped linear prediction of physical model excitations with applications in audio compression and instrument synthesis. *EURASIP Journal on Applied Signal Processing*, 2004(7):1036–1044, June 2004. ISSN 1110-8657.

D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, Massachusetts, USA, 1989.

M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka. RWC music database: Music genre database and musical instrument sound database. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, pages 229–230, Washington, D.C., USA and Baltimore, Maryland, USA, October 2003.

A. Grajdeanu and K. De Jong. Improving the locality properties of binary representations. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, page 1186–1196, Seattle, Washington, USA, 2004.

F. Gray. *Pulse Code Communication.* March 1953. US Patent 2,632,058. Bell Telephone Laboratories, NY, NY. Filing date 13 Nov 1947.

M. Hans and R. W. Schafer. Lossless compression of digital audio. *Signal Processing Magazine, IEEE*, 18(4):21–32, 2001.

J.E. Hawkins, Jr. and S. S. Stevens. The masking of pure tones and of speech by white noise. *The Journal of the Acoustical Society of America*, 22(1):6–13, January 1950.

N. Henrich, B. Doval, and M. Castellengo. On the use of the derivative of electroglottographic signals for characterization of nonpathological phonation. *The Journal of the Acoustical Society of America*, 115:1321, 2004.

R. Hinterding, H. Gielewski, and T.C. Peachey. The nature of mutation in genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 65–72, Pittsburgh, PA, USA, July 1995. Morgan Kaufmann.

J. H. Holland. Genetic algorithms. *Scientific American*, 267(1):66–72, 1992.

A. Horner, J. Beauchamp, and L. Haken. Methods for multiple wavetable synthesis of musical instrument tones. *Journal of the Audio Engineering Society*, 41(5):336 – 356, May 1993.

IBM and Microsoft. *Multimedia Programming Interface and Data Specifications 1.0*, chapter 3, pages pp. 22–31. IBM Corporation and Microsoft Corporation, August 1991.

IEEE. *IEEE Standard for a High Performance Serial Bus.* IEEE Std 1394-1995. The Institute of Electrical And Electronics Engineers, Inc., New York, NY, USA, 1996.

ISO/IEC. *ISO/IEC 11172-3:1993 Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s, Part 3: Audio.* MPEG-1 Audio. International Organization for Standardization, 1993.

ISO/IEC. *13818-3:1998 : Information technology - generic coding of moving pictures and associated audio, part 3: Audio.* MPEG-2. International Organization for Standardization, 1998.

ISO/IEC. *226:2003 : Acoustics - Normal equal-loudness-level contours.* International Organization for Standardization, 2003.

ISO/IEC. *14496-3:2005 Coding of audio-visual objects, part 3: Audio.* MPEG-4 Audio. International Organization for Standardization, 2005.

E. Jacobsen and P. Kootsookos. Fast, accurate frequency estimators [DSP tips & tricks]. *IEEE Signal Processing Magazine*, 24(3):123–125, 2007.

J. S. R. Jang. QBSH: a corpus for designing QBSH (Query by Singing/Humming) systems, 2010. [Online] Link valid 23rd September 2010. Available at http://mirlab.org.

D. Karaboğa and S. Őkdem. A Simple and Global Optimization Algorithm for Engineering Problems: Differential Evolution Algorithm. *Turk J Elec Engin*, 12(1), 2004.

T. Kientzle. *A Programmer's Guide To Sound.* Addison Wesley, Reading, Massachusetts, USA, 1998.

R. Koenen. Profiles and levels in MPEG-4: approach and overview. *Signal Processing: Image Communication*, 15(4-5):463–478, 2000.

Creative Labs. *Soundfont 2.01 Specification*, 1998. [Online] Available 23 September 2010. From http://connect.creativelabs.com/developer/SoundFont.

J. Lazzaro and J. Wawrzynek. Compiling MPEG 4 structured audio into c. In *Workshop and Exhibition on MPEG-4 (WEMP 2001)*, pages 5–8, San Jose, California, USA, June 2001.

J. Lazzaro and J. Wawrzynek. An RTP payload format for MIDI. In *Proceedings of the 117th Convention of the Audio Engineering Society*, San Francisco, California, USA, October 2004. AES.

T. Liebchen. An introduction to MPEG-4 audio lossless coding. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. (ICASSP '04)*, volume III, pages 1012–1015, may. 2004.

G. Loy. *Musimathics, Volume 1.* The MIT Press, Cambridge, MA, USA, June 2006.

G. Loy. *Musimathics, Volume 2.* The MIT Press, Cambridge, MA, USA, June 2007.

E. Lyon. Dartmouth symposium on the future of computer music software: A panel discussion. *Computer Music Journal*, 26(4):13–30, 2002.

D. MacKay. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, Cambridge, UK, 2003.

N. Marinic, J. Natterer, and W. Schneider. *Kontakt Script Language Manual.* Native Instruments Software Synthesis GmbH., berlin, Germany, 1994.

K. E. Mathias and L. D. Whitley. Transforming the search space with Gray coding. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 513—518, Orlando, Florida, USA, June 1994.

R. J. McAulay and Th. F. Quartieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-34: 744–754, 1986.

K. Melih and R. Gonzalez. Audio object coding for distributed audio data management applications. In *Proceedings of the 8th International Conference on Communication Systems (ICCS 2002)*, volume 2, Singapore, November 2002.

M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, Massachusetts, 1998.

T. J. Mitchell and D. P. Creasey. Evolutionary sound matching: A test methodology and comparative study. In *Proceedings of the 6th International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 229–234, Cincinnati, Ohio, USA, December 2007.

MMA. *General MIDI.* MIDI Manufacturers Association, Los Angeles, CA, USA, 1991a.

MMA. *General MIDI 2.* MIDI Manufacturers Association, Los Angeles, CA, USA, 1991b.

MMA. *Standard MIDI Files 1.0.* MIDI Manufacturers Association, Los Angeles, CA, USA, 1996.

MMA. *DLS Level 1 Specification.* MIDI Manufacturers Association, Los Angeles, CA, USA, 1997.

MMA. Sound synthesis standards harmonized, October 1998. [Online Press Release] Link valid 23rd September 2010. Available at http://www.midi.org/newsviews/sasbf.shtml.

MMA. *MIDI Media Adaptation Layer for IEEE-1394.* MIDI Manufacturers Association, Los Angeles, CA, USA, 2000a.

MMA. *Complete MIDI 1.0 Detailed Specification.* MIDI Manufacturers Association, Los Angeles, CA, USA, 2000b.

MMA. *XMF Meta File Format 1.0 Specification.* MIDI Manufacturers Association, Los Angeles, CA, USA, 2001.

MMA. *Downloadable Sounds Level 2.1.* MIDI Manufacturers Association, Los Angeles, CA, USA, April 2006.

T. Modegi and S. -I. Iisaku. Proposals of MIDI coding and its application for audio authoring. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS 1998)*, page 305–314, Austin, Texas, USA, July 1998.

Brian R.; Baer Thomas Moore, Brian C. J.; Glasberg. A model for the prediction of thresholds, loudness, and partial loudness. *Journal of the Audio Engineering Society*, 45 (4):224–240, 1997.

Jerry Morrison. *"EA IFF 85" Standard for Interchange Format Files.* Electronic Arts, January 1985.

P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech Concurrent Computation Program, C3P Report*, 826: 1989, 1989.

M. Nelson and J.-L. Gailly. *The Data Compression Book second edition.* M & T Books, New York, NY, USA, 1997.

L. Peltola, C. Erkut, P. R Cook, and V. Valimaki. Synthesis of hand clapping sounds. *IEEE Transactions on Audio, Speech and Language Processing*, 15(3):1021–1029, 2007.

J. R. Pierce. *An introduction to information theory: symbols, signals and noise.* Dover Publications, New York, NY, USA, 1980.

F. Plante, G. F. Meyer, and W. A. Ainsworth. A pitch extraction reference database. In *Proceedings of the 4th European Conference on Speech Communication and Technology (EUROSPEECH '95)*, pages 837–840, Madrid, Spain, September 1995. ISCA.

A. Puri and A. Eleftheriadis. MPEG-4: an object-based multimedia coding standard supporting mobile applications. *Mobile Networks and Applications*, 3(1):5–32, 1998.

H. Purnhagen and N. Meine. HILN - the MPEG-4 parametric audio coding tools. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, volume 3, Geneva, Switzerland, May 2000.

H. Purnhagen, B. Edler, and C. Ferekidis. Object-based analysis/synthesis audio coder for very low-bit rates. In *Proceedings of 104th Convention of the Audio Engineering Society*, Amsterdam, The Netherlands, May 1998. AES.

D. W. Robinson and R. S. Dadson. Threshold of hearing and Equal-Loudness relations for pure tones, and the loudness function. *The Journal of the Acoustical Society of America*, 29(12):1284–1288, December 1957.

J. P. Robinson and M. Cohn. Counting sequences. *IEEE Transactions on Computers*, 30 (1):17–23, 1981.

T. Robinson. SHORTEN: simple lossless and near-lossless waveform compression. Technical report, Cambridge University Engineering Department, Cambridge, UK, 1994.

D. Rossum. *The SoundFont 2.0 File Format: A White Paper.* Joint E-mu/Creative Technology Center, 1997. [Online] Link valid 23 Spetember 2010. Available from http://connect.creativelabs.com/developer/SoundFont.

F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms.* Springer, Heidelberg, Germany, 2nd edition, 2006.

Franz Rothlauf. *Locality, Distance Distortion, and Binary Representations of Integers.* Fakultät für Betriebswirtschaftslehre, Universität Mannheim, Mannheim, Germany, 2004.

J. Rowe, D. Whitley, L. Barbulescu, and Jean-Paul Watson. Properties of Gray and binary representations. *Evolutionary Computation*, 12(1):47–76, 2004.

C. Savage. A survey of combinatorial Gray codes. *SIAM Review*, 39(4):605–629, 1997.

C. D. Savage and P. Winkler. Monotone Gray codes and the middle levels problem. *Journal of Combinatorial Theory, Series A*, 70(2):230–248, 1995.

E. D. Scheirer. The MPEG-4 structured audio standard. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 6, page 3801–3804 vol.6, Seattle, Washington, USA, 1998. ISBN 1520-6149.

E. D. Scheirer. Structured audio and effects processing in the MPEG-4 multimedia standard. *Multimedia Systems*, 7:11–22, 1999.

E. D. Scheirer. Structured audio, Kolmogorov complexity, and generalized audio coding. *IEEE Transactions on Speech and Audio Processing*, 9(8):914–931, November 2001.

E. D. Scheirer and L. Ray. Algorithmic and wavetable synthesis in the MPEG-4 multimedia standard. In *Proceedings of the 105th Convention of the Audio Engineering Society*, San Francisco, California, USA, September 1998. AES.

Eric D. Scheirer, Youngjik Lee, and Jae-Woo Yang. Synthetic and SNHC audio in MPEG-4. *Signal Processing: Image Communication*, 15(4-5):445–461, January 2000.

H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RFC1889 - RTP: A Transport Protocol for Real-Time Applications.* Internet Engineering Task Force (IETF), September 1996.

X. Serra. *A system for sound analysis / transformation based on a deterministic plus stochastic decomposition.* PhD thesis, Stanford University, USA, 1989.

X. Serra. Musical sound modelling with sinusoids plus noise. In C. Roads, S. Pope, A. Picialli, and G. De Poli, editors, *Musical Signal Processing*, page 91–122. Swets and Zeitlinger, 1997.

W.A. Sethares. *Tuning, Timbre, Spectrum, Scale.* Springer-Verlag London Ltd, Heidelberg, Germany, 1997.

Y. S. Siao, A. W. Y. Su, J. L. Yah, and J. L. Wu. A structured audio system based on JavaOL. In *Proceedings of the International Workshop on Computer Music and Audio Technology (WOCMAT '05)*, Taipei, Taiwan, 2005.

N. J. Sieger and A. H. Tewfik. Audio coding for conversion to MIDI. In *Proceedings of the 1st IEEE Workshop on Multimedia Signal Processing*, page 101–106, Princeton, New Jersey, USA, June 1997.

F. Signol, C. Barras, and J.-S. Liénard. Evaluation of the pitch estimation algorithms in the monopitch and multipitch cases. In *Proceedings of Acoustics '08*, Paris, France, July 2008.

J. O. Smith and X. Serra. PARSHL: an analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation. In *Proceedings of the 1987 International Computer Music Conference (ICMC 1987)*, Champaign-Urbana, Illinois, USA, 1987.

M. Srinivas and L. M Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, June 1994.

A. W. Y. Su, Yi-Song Xiao, Jia-Lin Yeh, and Jian-Lung Wu. Real-Time internet MPEG-4 SA player and the streaming engine. In *Proceedings of the 116th Convention of the Audio Engineering Society*, Berlin, Germany, May 2004. AES.

Tallstick Sound Project. Ts-audiotomidi 2.01. [Computer program] Link valid 23rd September 2010. Available from http://www.tallstick.com.

T. Tolonen. Object-Based sound source modeling for musical signals. In *Proceedings of the 109th Convention of the Audio Engineering Society*, Los Angeles, USA, September 2000. AES.

S. van de Par, A. Kohlrausch, G. Charestan, and R. Heusdens. A new psychoacoustical masking model for audio coding applications. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '02)*, volume 2, Orlando, Florida, USA, May 2002.

N. Vasiloglou, R. W. Schafer, and M. C Hans. Lossless audio coding with MPEG-4 structured audio. In *Proceedings of the 2nd International Conference on Web Delivering of Music*, page 184–191, Darmstadt, Germany, December 2002. IEEE Computer Society.

B. Vercoe, MIT Media Lab, et al. *The Canonical Csound Reference Manual*, version 5.12 edition, August 2010. Downloaded 30th August 2010.

B. L. Vercoe, W. G. Gardner, and E. D. Scheirer. Structured audio: creation, transmission, and rendering of parametric sound representations. *Proceedings of the IEEE*, 86(5): 922–940, 1998.

R. D. Villwock. KSP math library technical guide v2.15, April 2009. [Online] Link valid 23 September 2010. Available from http://www.andrewkmusic.com/filearea/SIPS/MathLibraryV215.zip.

E. Vincent and M. D. Plumbley. A prototype system for object coding of musical audio. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA 2005)*, page 239–242, New Paltz, New York, USA, 2005.

E. Vincent and M. D. Plumbley. Low Bit-Rate object coding of musical audio using bayesian harmonic models. *IEEE Transactions on Audio, Speech and Language Processing*, 15(4): 1273–1282, 2007.

H. Viste and G. Evangelista. Sound source separation: Preprocessing for hearing aids and structured audio coding. In *Proceedings of the 4th COST G-6 Conference on Digital Audio Effects (DAFx-01)*, Limerick, Ireland, December 2001.

VOQUAL'03. Voice quality: Functions, analysis and synthesis (VOQUAL'03). Geneva, Switzerland, August 2003.

Tien-Ming Wang, Yi-Song Siao, and A. W. Y. Su. JavaOL - a structured audio orchestra language: Tools, player and streaming engine. In *Proceedings of the 120th Convention of the Audio Engineering Society*, Paris, France, May 2006. AES.

S. J. Welburn and M. D. Plumbley. Estimating parameters from audio for an EG+LFO model of pitch envelopes. In *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx-09)*, Como, Italy, September 2009a.

S. J. Welburn and M. D. Plumbley. Properties of Gray and binary codes for optimization. Technical Report C4DM-TR-09-02, Centre for Digital Music, Queen Mary, University of London, London, UK, 2009b.

S. J. Welburn and M. D. Plumbley. Rendering audio using expressive MIDI. In *Proceedings of the 127th Convention of the Audio Engineering Society*, New York NY, USA, October 2009c. AES.

S. J. Welburn and M. D. Plumbley. Improving the performance of pitch estimators. In *Proceedings of the 128th Convention of the Audio Engineering Society*, London, UK, May 2010. AES.

D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.

D. Whitley. A free lunch proof for Gray versus binary encodings. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, volume 1, page 726–733, Orlando, FL, USA, July 1999.

D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Building better test functions. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, page 239–246, Pittsburgh, PA, USA, July 1995. Morgan Kaufmann.

D. Whitley, Soraya B. Rana, J. Dzubera, and K. E Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1–2):245–276, 1996.

D. H. Wolpert and W. G. Macready. No Free Lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

S. Wun and A. Horner. Evaluation of iterative methods for wavetable matching. *Journal of the Audio Engineering Society*, 53(9):826–835, September 2005.

Yamaha. *TX816 FM Tone Generator System Performance Notes*. Nippon Gakki Co. Ltd., Hamamatsu, Japan, 1985.

Yamaha. *MU80 Tone Generator Owners Manual*. Yamaha Corporation, Japan, 1994.

G. Zoia and C. Alberti. A virtual DSP architecture for audio applications from a complexity analysis of MPEG-4 structured audio. *IEEE Transactions on Multimedia*, 5(3):317–328, 2003.