Estimation of key in digital music recordings

MSc Computer Science Project Report

This report is substantially the result of my own work, expressed in my own words, except where it is explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service.

Ibrahim Sha'ath
Department of Computer Science and Information Systems
Birkbeck College, University of London

Abstract

The goal of this work is to develop software that can analyse a digital recording of music and estimate the key of the recorded piece, in order to provide, at a glance, some information on its tonal compatibility with other recordings.

Section 1 describes the motivation for automated key classification to support disc jockeys with a computer-enabled workflow. It presents some basic music theory to clarify the concepts and terms underpinning the project. The scope is established with a statement of requirements for the software.

Section 2 describes the project's theoretical approach, comparing it to previous work on the problem. It discusses the algorithms employed, and describes a novel method of translating the output of a Fast Fourier Transform into a musical analysis.

Section 3 describes the implementation from a software engineering perspective. It outlines the principles adhered to during development and describes the design of the user interface and classes. The testing strategy is presented.

Section 4 describes the experiments conducted to test and optimise the accuracy of the software. It compares the performance of the software to existing products.

The conclusions drawn are presented in section 5. The contributions and limitations of the project are considered, and possible future improvements are discussed.

Project supervisor Steve Maybank

Acknowledgements

I would like to thank my supervisor Steve Maybank for his guidance, Roland Heap for his enthusiasm and pride in keying whatever music was thrown his way, and Chris Harte for his encouragement, as well as his ability to explain anything in small words. Thanks also to Qualia for enriching the DJ community with Rapid Evolution.

I would like to acknowledge Mixed In Key for providing a copy of their software to use in my experiments, and the developers behind the Qt framework, LibAV, the Fastest Fourier Transform in the West, TagLib and Secret Rabbit Code, without whose efforts this and doubtless many other projects would not have been feasible.

Contents

List of Figures

List of Tables

1 Introduction

This section explains the motivation for this project, by considering the role of the DJ (disc jockey) and the impact of computing on the DJ's workflow and practice. It explores the principles of western tonal music theory relevant to the concept of key classification, and presents a statement of requirements for the software product.

1.1 The DJ and mixing music

For decades prior to the 1940s, the DJ was exclusively a radio personality, introducing records and talking to the listening audience; dancehalls and nightclubs were venues solely for live bands. In the 1940s, taking some inspiration from the jukebox and some from the "commercial possibilities of a band-less dance" (Brewster and Broughton 2006, p.54), the idea of the club DJ was born, and the practice of the DJ began to evolve beyond the radio presenter style.

In 1965, professional club dancer Terry Noel "first hit on the idea that two records could be somehow sewn together" (Brewster and Broughton 2006, p.78). Since then, many DJs have employed a variety of techniques to *mix* music, using existing records as raw materials to create something new.

The basic equipment for mixing is a pair of turntables and an audio mixer (Poschardt 1998, p.31). The mixer must be able to output either or both of the turntables' signals to the main sound system, and must also allow the DJ to hear either source independently of the main output (typically through headphones). It is usual for the turntables to include some control

over the speed of the motor, allowing the music to be played faster or slower. With this level of control, the DJ is able to match the tempo of one record to another, and by manual manipulation of the turntable platter, to synchronise a record only she can hear to the rhythm of the record currently playing to the audience (Poschardt 1998, p.108). She can then use the controls of the mixer to quickly make a transition or *cut* from the playing record to the other without a break in the rhythm of the music. Alternatively, she can perform a longer *mix*, playing the synchronised records together to create new sounds.

The range of control over a turntable's speed is limited. The industry standard Technics SL-1200 allows a deviation up to ±8% from 33 or 45 rpm, so for example it is not possible to rhythmically mix a record at 90 bpm (beats per minute) with one at 120 bpm, even if they are played at opposite limits of the speed control. Another effect of the speed change that must be considered is that a record played too fast or too slow plays at a noticeably higher or lower pitch. This is not usually a problem for noisy percussive sounds like drums, but tonal sounds like singing may become comical or unpleasant.

For these reasons, and due to the time constraints of performing live (for example, playing a three minute song gives the DJ little time to put away the previous record, choose the next, and prepare it for mixing), many DJs classify their records by tempo in advance of a performance (Poschardt 1998, p.172). They might write on a record's paper sleeve, or otherwise note the tempo where it can be seen at a glance while selecting the next record to play.

With the arrival of the affordable laptop computer, some DJs began to migrate to software equivalents of the turntable and mixer setup[1]. A common reason for this initial migration was that a laptop storing digital recordings is easier to carry around than cases full of vinyl records, but as DJing software evolved, the technical capabilities available to the

[1]Many continued to use hardware devices, including turntables, to maintain tactile control over their computerised performances, but such implementations are not the project's focus.

computer-using DJ improved beyond what was possible with traditional equipment.

The specific innovation that motivates this project is generically called *pitch lock*; DJing software and hardware began to implement algorithms that allowed a song to be played faster or slower without significantly affecting the sound of the recording, in that the pitch of the song did not change even though the speed was changed. This enabled DJs to mix a much wider range of recordings with regard to their tone, as well as their rhythm. For example, if two records are tonally compatible (a concept we will consider in detail in section 1.2.7), the vocal part of one song might be mixed with the instrumental part of the other. Or two compatible records might be mixed together to create a new musical harmony.

This new possibility only increased the DJ's workload, however, and the time pressure during a performance remained. As a result it was necessary to classify recordings with regard to their tonal compatibility in advance, just as was done with tempo. The classification is no longer to be written on a paper sleeve, but stored in the metadata of each audio file, to be displayed by the DJing software during a performance.

The aim of this project is to implement software that can automatically classify a digital recording with respect to its tonal content. This classification must be based on the principles of music theory, which are considered in the next section.

1.2 Music theory

We must begin by considering our scope. This review of music theory is limited to modern, western, tonal music, since this has the most relevance to the problem domain. The principles of this theory do not hold for all historical musical traditions, or for all modern music, especially in other parts of the world.

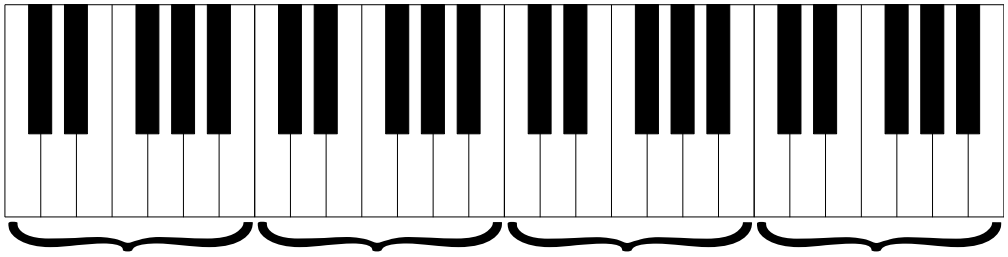

Figure 1.1: Repetitive pattern of octaves on a piano keyboard. Notes increase in pitch from left to right.

1.2.1 Notes

The building block of music is the note. When a note is sounded, we primarily perceive a fundamental frequency called its pitch. In fact, most musical instruments simultaneously sound a series of harmonically related frequencies called overtones, but these are "very soft in comparison with the fundamental" and are perceived as contributing to the quality or timbre of the instrument's sound (Lovelock 1957, pp.122-123).

1.2.2 Octaves

The relative difference between two pitches is called an interval, and the simplest interval is the octave. The octave is the interval between any two notes with a frequency ratio of 2:1. The human ear perceives notes separated by an octave as essentially the same, since "every wave crest in the lower tone is duplicated in every alternate wave crest of the higher tone"; in music theory terms this gives us "a tonal spectrum which is alike at both ends ... so that it can be repeated over and over again at different altitudes of pitch" (Abbott 1942, pp.32-33). This repetition is evident in musical terminology: a set of notes an integral number of octaves apart will share the same name (notes are named with the letters A-G); their octave relationship is more important than their absolute frequencies. The repetition is also obvious in the keyboard of a piano, where the repeating pattern of white and black keys corresponds to the cycle of the octave (as shown in figure 1.1).

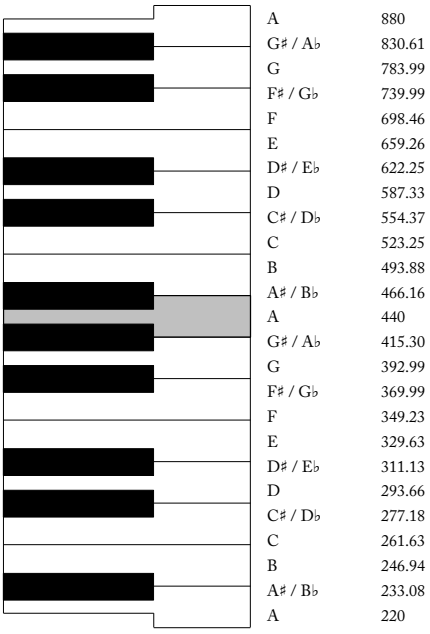

Figure 1.2: The right hand column shows the frequencies (Hz) of two octaves of notes, derived from A440 using 12-TET.

1.2.3 Tuning

Western music is commonly tuned using a system called twelve-tone equal temperament (12-TET) (Sadie 1980, v.6 p.218). An equal temperament system is one in which there is a constant frequency ratio between any two adjacent notes. In 12-TET the octave is divided into twelve notes, so there is a constant frequency ratio of $\sqrt[12]{2}$ between consecutive notes. The interval between adjacent notes is called the semitone, which is the smallest common interval (Károlyi 1973, p.38).

The usual starting point for the western tuning standard is 440 Hz, which is the frequency of a note A near the middle (in musical terms) of the audible spectrum. The frequencies of all other notes in 12-TET can be derived simply from A440 (as shown in figure 1.2).

1.2.4 Scales

A scale is "a series of notes built in progression" from any note (known as the *root* or *tonic*) to the note an octave above it (Károlyi 1973, p.39). The basic scale in western music is the diatonic scale, which employs seven of the twelve notes in an octave (if we also count the note an octave above the root, we have eight notes, which is the origin of the word octave).

Again, the piano keyboard is a simple illustration of the diatonic scale (Sadie 1980, v.16 p.545). If we play all the white keys in an octave starting from C we get a *major* diatonic scale; specifically the scale of C major (C-D-E-F-G-A-B-C). The major scale is characterised by a specific pattern of intervals: from C to D is two semitones, from D to E is two semitones, from E to F is one semitone, and so on. A major scale can start from any note (a major scale starting from E would be called E major), and the pattern of intervals will always be 2-2-1-2-2-2-1.

The other diatonic scale to be aware of is the *minor* scale, which has a different pattern of intervals: 2-1-2-2-1-2-2. The scale of C minor would not, then, be played only on the white keys of a piano; the third, sixth and seventh notes are lowered a semitone from the major scale, and are played on the black keys E♭, A♭ and B♭[2] respectively. The lowered notes are the basis for the term 'minor'; they are a shorter distance from the root, compared to their higher counterparts in the major scale (Abbott 1942, pp.40-41).

These scales have decidedly different characters. The major scale sounds bright, happy or lightweight, whereas the lowered notes of the minor scale lend it a sound considered more emotive, darker or sadder. We also need to be aware of the *chromatic* (meaning colourful or prismatic) scale, so called because it contains all twelve tones of the octave. As such it is not useful in terms of classification, but can be considered the root from

[2]The names of the notes sounded by the black keys on a piano keyboard include a sharp (♯) or flat (♭) symbol, indicating that the natural note referred to by the letter is raised or lowered by a semitone. In an equal temperament system this means that, for example, the names D♯ and E♭ refer to the same note; these are *enharmonic equivalents.*

which the diatonic scales are derived. There are also other, so-called *modal* scales, but these are not considered here as they are not often employed, being for the most part obsolete or exotic variants (Abbott 1942, p.39).

1.2.5 Keys

"When a piece of music is written using as its basis the notes of a certain scale, it is said to be in the key of that scale" (Lovelock 1957, p.41). Since each note in the octave can be used as the starting point for a major and a minor scale, this gives us twenty-four keys which are used in the composition and performance of most modern western music. Abbott (1942, p.45) refers to these as "twenty-four tonal compartments in which to place music", and this project is based on that principle; the aim is to classify musical recordings as being in one of the twenty-four keys.

There is a quandary here: to say that a piece of music is in a single global key is often an oversimplification. Most music moves from one key to another (a process called modulation); usually the initial key is reestablished, but there is also a tendency in some popular music to have a single dramatic key change towards the end of a song to build momentum. There is a converse phenomenon specific to this problem domain: the electronic dance music played by many DJs does not usually feature noticeable key changes; energy and movement is more often derived from the evolving sound of repetitive phrases, and from rhythm, than from modulation.

The primary goal here is to provide a classification that can be noted at a glance, so a single key estimate for a recording must be the project's aim. However, it would be best to provide a supplementary way for the DJ to visualise key changes over the course of a recording. This idea is returned to in section 1.3 when we consider the software requirements in detail.

1.2.6 Consonance

"Acoustically, the sympathetic vibration of sound waves of different frequencies related as the ratios of small whole numbers; psychologically,

a harmonious sounding together of two or more notes, that is with an 'absence of roughness'" (Sadie 1980, v.4 p.668).

We have already seen that two notes an octave apart are perceived as essentially the same; this is called a perfect consonance. The perception of consonance is associated with a simplicity of frequency ratio: the octave is a ratio of 2:1, and other simple ratios approximated in 12-TET give us the other perfect consonances; the perfect fifth (an interval of seven semitones, with a frequency ratio of ∼3:2) and the perfect fourth (the inverse of the fifth: an interval of five semitones, with a frequency ratio of ∼3:4). The fourth and fifth are so called because of their positions in the diatonic scales; note that they appear at the same intervals in both the major and minor scales (hence 'perfect').

1.2.7 Key compatibility

The consonances between notes lead to strong relationships between groups of keys; some transitions between keys will sound much more concordant than others.

Two songs in the same key will tend to use all the same notes and have the same tonic or root note, so they will almost certainly sound harmonically compatible. Shifting from one key to another with roots separated by a fifth or fourth will likewise be harmonious; they share all but one of the notes in their respective scales, and their tonic notes are related by a perfect consonance. Songs in *parallel* keys, that is to say major and minor keys with the same root (e.g. C major and C minor) share a tonic and all but three notes; they will often be compatible.

There is also the concept of a *relative* key; for every major key there is a minor key whose scale is composed of the same notes (C major and A minor, for example, are both composed of all the white notes on a piano). The transition from a key to its relative major or minor is also likely to sound highly compatible.

These relationships are illustrated by what is known as the circle of fifths

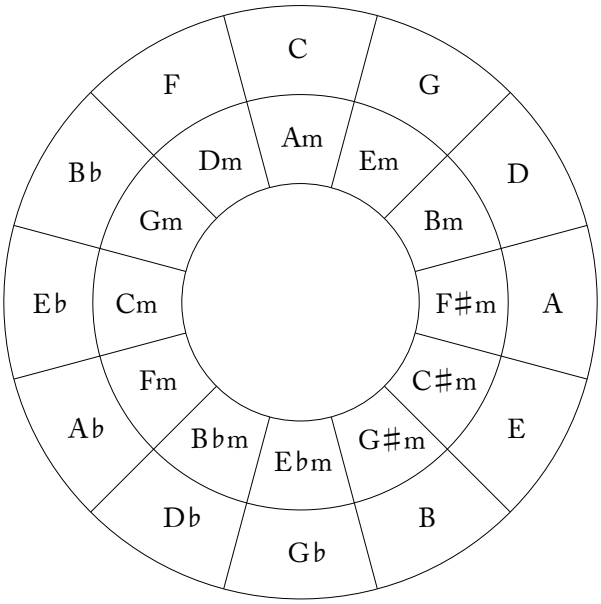

Figure 1.3: The circle of fifths. Major keys are on the outside of the circle, minor keys are on the inside and suffixed 'm'. Each clockwise step is a fifth interval, each anti-clockwise step a fourth. Relative major and minor keys are adjacent. For example, from a passage of music in C, the most harmonious transitions will be to G (a fifth above), F (a fifth below), or Am (the relative minor).

(figure 1.3). It arranges all twenty-four keys with respect to their concordant relationships, with relative keys forming inner and outer rings, and each clockwise step representing a perfect fifth interval.

This demonstrates why key classification is so useful to the DJ. As a rule of thumb, a transition from any starting key to an adjacent one on the circle of fifths will sound concordant, so with pre-classified recordings the DJ can determine whether songs are tonally compatible very quickly[3].

This is a sufficient theoretical backdrop to consider the requirements for the software product.

[3]The circle was further simplified in the 1990s by a DJ named Mark Davis. He employed the metaphor of a clockface and replaced each key's name with a simple alphanumeric code (8A for Am, 8B for C, 9A for Em, etc.), enabling DJs to make quick decisions without needing to remember the circle of fifths (Camelot Sound). This representation of key relationships is commonly seen in DJ culture.

1.3 Requirements

1.3.1 Use Cases

The use cases for the software are straightforward. Firstly, the user must be able to automatically estimate key classifications for a batch of audio recordings, and write these estimates to the audio file metadata for later use (table 1.1). Secondly, she should be able to inspect the key classification of a single recording in detail to consider the effects of key changes over time (table 1.2).

1.3.2 Non-functional requirements

The non-functional requirements for the software are as follows:

- The software must be able to decode a wide variety of audio formats.
- The software must be able to scale robustly to the analysis of very large batches, up to thousands of files.
- The software must be extensible to allow the implementation of new algorithms.
- The software should be Mac OS X compatible but portable to Microsoft Windows.
- The software should be as fast as possible without sacrificing accuracy.
- The release of the software must not infringe on the licenses of any integrated libraries.

Table 1.1: Use case 1

Use Case Name: Batch key estimation	ID: 1	Importance: High
Primary Actor: User	Use Case Type: Detail Essential	
Description: This use case describes the process of estimating the keys of a batch of digital audio recordings.		
Normal Flow: 1. The user selects a set (e.g. a directory) of audio files to analyse. 2. The software computes overall key estimations for each file in turn, and displays them. 3. The user confirms that the key estimations should be written to the audio file metadata. 4. The software writes the key estimations to the audio file metadata.		
Exceptional Flow: 2a. If any of the files cannot be located or decoded, the software informs the user without interrupting the batch process.		

Table 1.2: Use case 2

Use Case Name: Detailed key analysis	ID: 2	Importance: High
Primary Actor: User	Use Case Type: Detail Essential	
Description: This use case describes the process of inspecting the detailed key information of a single digital audio recording.		
Normal Flow: 1. The user selects a single audio file to analyse. 2. The software computes a key estimation for the audio file, providing visualisations of the estimation process to illustrate its operation. The visualisations should include a list of the keys detected over the course of the recording.		
Exceptional Flow: 2a. If the file cannot be located or decoded, or some other exception is encountered, the software informs the user.		

1.4 Summary

This section has outlined the role of the DJ and the motivation for automated key classification. The principles of music theory underlying the project were illustrated, and the requirements for the software were described. The next section discusses the project's theoretical approach, and the algorithms employed in the key estimation process.

2 Solution design

This section discusses the algorithms employed in the extraction of musical features from digital audio recordings, and presents a novel process for a musical spectral analysis. It describes various algorithms which can be applied to the musical features to account for detuning or key changes, and shows the final key classification process.

2.1 Overview

This project's software implementation, referred to henceforth as *KeyFinder*, implements a linear process for estimating the key of a piece of music. First, the file containing the audio is opened and decoded, and the audio stream is extracted. The number of channels in the stream is reduced, mapping it to a single monoaural signal. A low-pass filter is then applied to the signal, in order to enable a reduction in the sample rate.

The stream is divided into overlapping analysis frames in the time domain, each of which is translated into the frequency domain by a Fast Fourier Transform. The output of each FFT is post-processed to obtain Constant Q components, which describe the energy at the frequencies of musical notes. The constant Q components are then mapped to a single-octave chromagram, describing the distribution of energy across the chromatic scale for each temporal frame of the audio.

A harmonic change detection function may be applied, which describes the rate of change in the music's tonal content; peaks in this signal are assumed to be key changes and used to segment the chromagram. Each of these segments is further reduced, the time dimension being discarded

to provide a single chromatic vector. This vector is correlated against a set of tone profiles, which represent the most likely distribution of energy for each of the twenty-four keys. Each segment is labelled with the key corresponding to the tone profile which yields the highest correlation with the chromatic vector.

The value of each segment to the overall classification is dictated by its length and its overall level of energy, so that the keys of longer and louder passages of music are weighted over shorter, quieter passages. The key with the highest final weighting is picked as the overall key for the recording.

2.2 Pre-processing of digital audio

2.2.1 Decoding

Digital audio signals are stored in a variety of file formats (known as containers), and encoded with a variety of algorithms (known as codecs). We will consider three examples.

The Waveform Audio format (commonly referred to as WAV, due to its filename extension), usually stores uncompressed audio in a linear pulse-code modulation (LPCM) format. For example, the LPCM data extracted from an audio CD is an interleaved stereo pair of signals, each sampled at 44100 Hz with 16-bit precision. Uncompressed WAV files tend to be large, roughly 10MB per minute.

Arguably the most popular format for digital audio, at least in the consumer market, is MPEG Audio Layer 3, commonly referred to as MP3. It reduces storage requirements by a process of lossy compression, removing elements of the audio signal at or beyond the limits of human auditory perception. This can be done by a variety of processes, to produce files of varying sizes (as a rule of thumb, a high quality MP3 is about one sixth

the size of the equivalent uncompressed WAV file). Too high a compression ratio can introduce audible artifacts, adversely[1] affecting the quality of the recording.

A middle ground is established by lossless compression formats like the Free Lossless Audio Codec (FLAC). These offer exact reproduction of CD-quality audio, with the storage burden reduced by roughly half, though at an increased computational cost for decoding.

To ensure compatibility with the widest possible range of containers and codecs, KeyFinder employs the LibAV family of open-source libraries (LibAV community 2011). These are the foundation of the FFmpeg transcoding tool and the popular VLC media player, and provide support for a wide variety of media formats, including WAV, MP3, FLAC and many others.

2.2.2 Reducing the data rate

Once decompressed and decoded, the audio files are generally in the LPCM format of an audio CD: an interleaved stereo pair of 44100 Hz, 16-bit sample streams. This gives ∼172 kB to analyse per second of audio, but this data rate can be reduced significantly without losing pertinent data.

First of all, stereophonic sampling is not useful; a signal's position in the stereo image does not affect the tonal content of the music (Noland 2009, p.37). Therefore, a simple average is taken of the two channels to reduce the audio to a single monophonic signal.

Secondly, the 44100 Hz sample rate allows for the accurate capture of frequencies up to the Nyquist frequency of 22050 Hz. This is around the limit of human perception (Olson 1972, p.315), but is an extremely high frequency from a musical point of view, so a low-pass filter can be applied, after which the audio can be *downsampled* to yield a much lower sample

[1] It has been suggested that the ubiquity of MP3 has led to a steady change in the way music is perceived by younger listeners, and that there is actually a growing preference for these artifacts (Berger 2009). For the purposes of this project accurate reproduction of the recorded material is favoured.

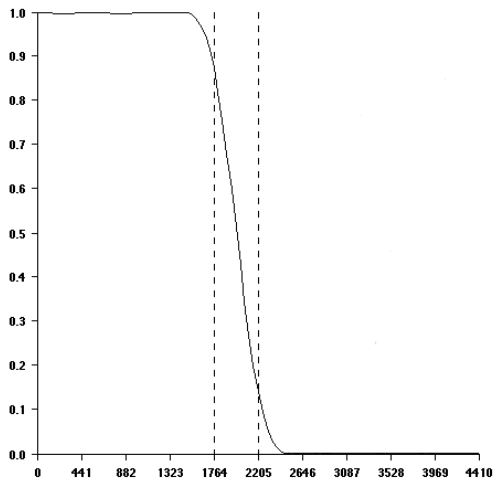

Figure 2.1: Magnitude response of default low-pass filter, plotted using Fisher's online toolkit. Vertical axis denotes magnitude, horizontal axis denotes frequency (Hz). The two dashed bars mark the key frequencies for the roll-off.

rate.

For an analysis of six octaves starting at A0, a very deep sub-bass note at 27.5 Hz, up to A♭6, around the top of the range of soprano singers at ∼1661 Hz, KeyFinder implements a low-pass filter designed with an online toolkit (Fisher 1999). It is a 161-tap root raised cosine filter with a roll-off from around 1760 Hz (A6, a semitone above the top of our range) to 2205 Hz, allowing a downsampling factor of 10 without aliasing (see figure 2.1).

The default downsampling process, incorporating Fisher's filter, is effective and quite fast, but it is based on hard-coded coefficients and is thus limited to these particular parameters. To guard against any non-standard sample rates or changes in the frequency analysis requirements, KeyFinder also integrates a library (de Castro Lopo 2009) which can work with a range of sample rates, albeit with significantly reduced performance.

The reduction to mono and downsampling leave a much more manageable data rate of ∼8.6 kB per second of audio, from which the extraction of musical features can begin.

2.3 Extraction of musical features from digital audio

2.3.1 The Fast Fourier Transform

The most common tool for spectral analysis of digital audio is the Fast Fourier Transform (FFT), an efficient software implementation of the Discrete Fourier Transform (DFT) (Noland 2009, pp.39-40). The DFT, when applied to a sampled audio signal, yields a representation of the energy present at certain frequencies. It is given by

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi nk/N} \quad \text{for} \quad k = 0, 1, \ldots, N-1 \tag{2.1}$$

where x_n is the sampled time-domain input, and X_k is the frequency domain output. The number of input samples, N, determines "the resolution of the frequency-domain results, and the amount of processing time necessary" (Lyons 2010, p.61).

The exact frequencies of the output components depend on both N and the rate at which the original signal was sampled (s). For example, with a signal sampled at 44100 Hz and a 1024-point Fourier transform, the fundamental analysis frequency will be $s/N = 44100/1024 \approx 43.07$ Hz. The other frequency components are "integral multiples of the fundamental frequency" (Lyons 2010, p.62) such that

$$f_k = ks/N \tag{2.2}$$

where k is the index of the frequency-domain component. So, the X_1 term in the output specifies the magnitude of any 43.07 Hz component in the input, the X_2 term specifies the magnitude of any 86.13 Hz component, and so on (Lyons 2010, p.62).

This is why the FFT, while computationally efficient, does not closely suit the problem of musical analysis. As discussed in section 1.2.3, the frequencies of musical notes in 12-TET are exponentially spaced by a constant ratio of $\sqrt[12]{2}$, whereas the frequency samples of an FFT are linearly spaced. Therefore, the FFT "yields components which do not map efficiently to musical frequencies" (Brown 1991, p.425); it has insufficient

resolution for low musical notes and excessive resolution for high notes. The very mature and efficient FFTW library (Frigo and Johnson 2009) is still the foundation of KeyFinder's spectral analyser[2], but the output of the FFT requires some post-processing.

2.3.2 The Constant Q Transform

Brown (1991, pp.426-427) devised a transform algorithm whose components are exponentially spaced and centred on musical frequencies. The eponymous Q is the relationship between a filter's bandwidth and its centre frequency; a filterbank where Q is constant can partition the frequency spectrum "into semitone-size regions" (İzmirli 2005, p.2). The CQT of a sampled time-domain sequence x_n is given by

$$X_k = \frac{1}{N_k} \sum_{n=0}^{N_k-1} w_{k,n} x_n e^{-i2\pi nQ/N_k} \tag{2.3}$$

where N_k is the number of time-domain samples for the kth spectral component and $w_{k,n}$ is a suitable window function[3].

Brown (1991, p.427) describes how the transform returns Fourier components X_k for frequencies Q/N_k; since Q is a constant, the choice of N_k is inversely proportional to the frequency of interest for component k.

Brown and Puckette (1992, pp.2698-2699) went on to publish a method of post-processing the output of an FFT to arrive at a CQT, taking "full advantage of the computational efficiency of the fast Fourier transform" (Brown and Puckette 1992, p.2698). This process pre-calculates a kernel

[2]KeyFinder also includes the Goertzel algorithm (Lyons 2010, pp.738-741), a transform that calculates a single DFT coefficient. It shows some promise for a musical feature extraction due to the ability to choose the analysis frequency, but is ultimately less effective than the FFT-based solution; and without a mature library implementation, is rather less efficient.

[3]A window is applied to the input of a discrete transform function to reduce the effects of *spectral leakage*, which occurs when there are input sinusoids that do not have "an integral number of cycles over N input samples" (Lyons 2010, p.85); that is, they do not oscillate exactly at one of the analysis frequencies. In dealing with real world signals, and certainly with music, this is almost always the case. A window function minimises these end-point discontinuities by smoothing the amplitude at the beginning and end of the sampled interval towards a common value (Lyons 2010, pp.89-90).

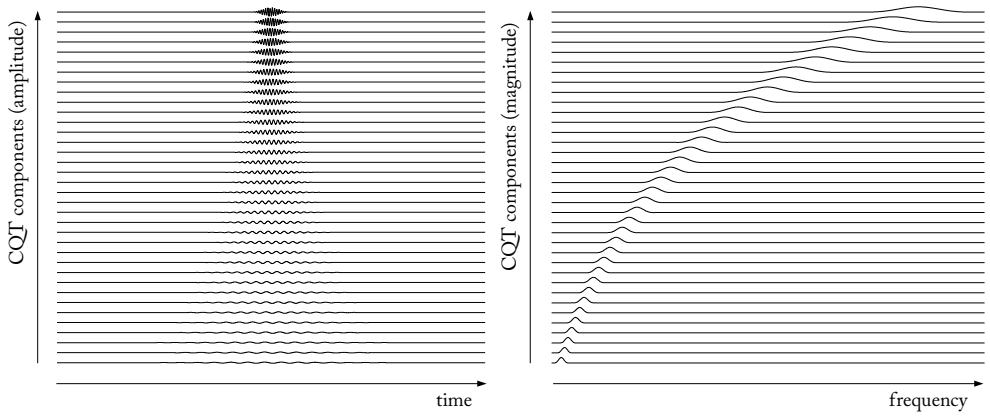

Figure 2.2: Constant Q Transform kernels. Each line represents an analysis component (this depiction is for an analysis of three octaves, from A3 at the bottom to A♭5 at the top). On the left, the real part of the temporal kernel; note the shorter windows and greater amplitude as the frequency increases. The Fourier transform of the temporal kernel is the spectral kernel, whose magnitude is plotted on the right. Note the geometric spacing between the centre frequencies of the components, and the greater bandwidth at higher frequencies.

that can be matrix multiplied with the frequency-domain output of the FFT, and re-used for each transform operation.

Initially they build a time-domain or *temporal kernel* of windowed sinusoids at the required analysis frequencies (left hand side of figure 2.2). They then take the FFT of this temporal kernel to yield a frequency-domain or *spectral kernel*. They determined empirically that the spectral kernel can be made sparse without losing too much information; this is done by discarding any magnitudes below a certain threshold, leaving only one non-zero region per frequency-domain component (right hand side of figure 2.2). Matrix multiplying the output of an FFT by the spectral kernel provides an efficient approximation to the constant Q transform in equation 2.3; the effect of the multiplication is illustrated in figure 2.3.

The CQT included in KeyFinder, based on MatLab implementations by Blankertz (2001) and Harte (2010, p.67), has been quite successful, but the software primarily uses a more efficient variation on the algorithm.

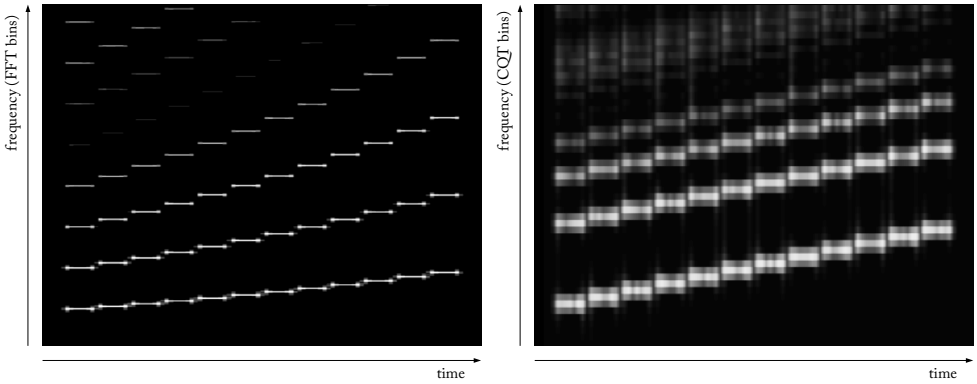

Figure 2.3: Comparison between FFT and CQT. Each image depicts a chromatic scale played on a square wave synthesizer. Lighter greys denote higher energy. On the left is a multi-frame FFT, on the right is the same result after post-processing with a CQT. In each image the harmonic overtones of the notes are obvious. Note the linear rise of the scale in the CQT, in contrast to the exponential pattern of the FFT.

2.3.3 A simpler approach to the CQT

This section presents a novel method of translating an FFT's output into constant-Q spectral components. This method is more efficient than Brown and Puckette's CQT. The principle is the same, but the spectral kernel is built directly, rather than by taking the FFT of a temporal kernel. This speeds up the generation of the kernel (which, admittedly, is not a concern in the long run, since it is reused for each transform) and reduces the number of arithmetic operations in each transform.

To approximate the CQT's spectral kernel, a cosine window in the frequency domain is used for each component, centred on the frequency of interest and with the bandwidth scaled by a constant Q. The left and right bounds of this window are defined as points along the continuum of FFT bins[4], and are given by

$$l_k = (1 - \frac{Q}{2})(\frac{f_k N}{s}) \tag{2.4}$$

$$r_k = (1 + \frac{Q}{2})(\frac{f_k N}{s}) \tag{2.5}$$

[4]As such, equations 2.4 and 2.5 are derived from equation 2.2.

where f_k is the frequency of interest for component k, N is the resolution of the FFT, and s is the sample rate. Q is given by

$$Q = p(\sqrt[12]{2} - 1) \tag{2.6}$$

where p is a user-defined parameter. FFT bins whose indices fall within the bounds l_k and r_k are assigned a coefficient, as given by

$$c_{b,k} = \frac{w_{b,k} f_k}{\sum_{i=\lceil l_k \rceil}^{\lfloor r_k \rfloor} w_{i,k}} \quad \text{for} \quad l_k \leq b \leq r_k \tag{2.7}$$

where b is the index of the FFT bin and $w_{x,k}$ is the spectral window function

$$w_{x,k} = 1 - \cos(2\pi \frac{x - l_k}{r_k - l_k}) \tag{2.8}$$

The normalisation by f_k and the summation of $w_{i,k}$ in equation 2.7 were arrived at empirically. Once normalised in this way, the coefficients produce a spectral kernel very similar to Brown and Puckette's, which can likewise be matrix multiplied with the output of an FFT to yield constant Q frequency components.

The parameter p in equation 2.6 allows for broader or narrower windows with more or less frequency overlap (see figure 2.4). This has been found to significantly increase KeyFinder's accuracy. Each window can be limited to spectral information from only one semitone, and the effects of noisy, wideband signals is significantly reduced. The results of this parameterisation are illustrated further in section 4 below.

2.3.4 Framing

Whichever version of the CQT is used, the output is a frequency-domain chromatic vector, measuring the energy found for each of the requested musical notes. Since the output of the FFT (and thus the CQT) has no time resolution, multiple transforms must be performed across the duration of the recording. As shown in section 2.3.1, the FFT transforms N samples at a time, so the audio stream is divided into frames of N samples, and a Blackman window function is applied to each frame to reduce

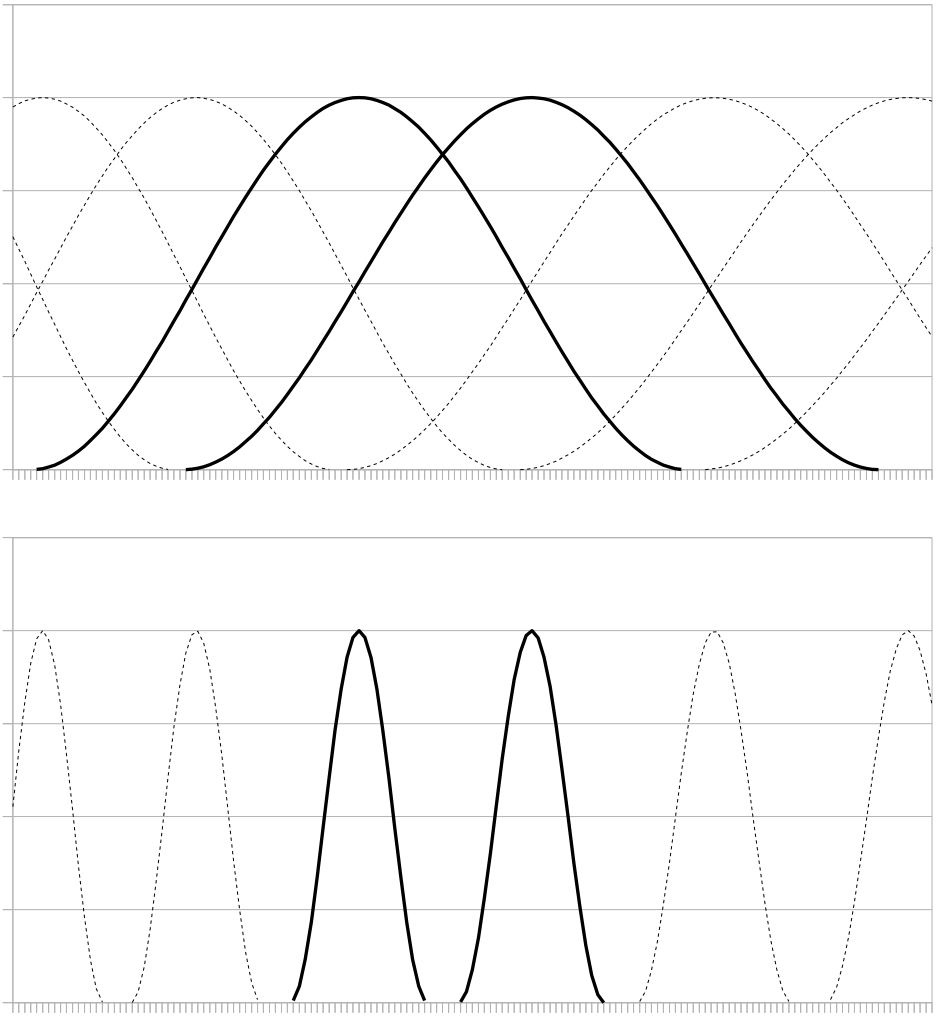

Figure 2.4: Parameterisation of direct spectral kernel bandwidth. The vertical axis denotes the magnitude of the window function, the horizontal axis denotes frequency, delineated by the analysis frequencies of FFT bins. The first graph shows neighbouring spectral windows with $p = 3.8$, which is where the direct spectral kernel most closely models Brown and Puckette's CQT. Note the considerable overlap between the frequencies analysed by adjacent windows. The second graph shows the same windows for $p = 0.8$, which is where KeyFinder's overall accuracy is highest. There is no overlap in analysis frequency bands, and indeed some frequencies that are not of musical interest are discarded.

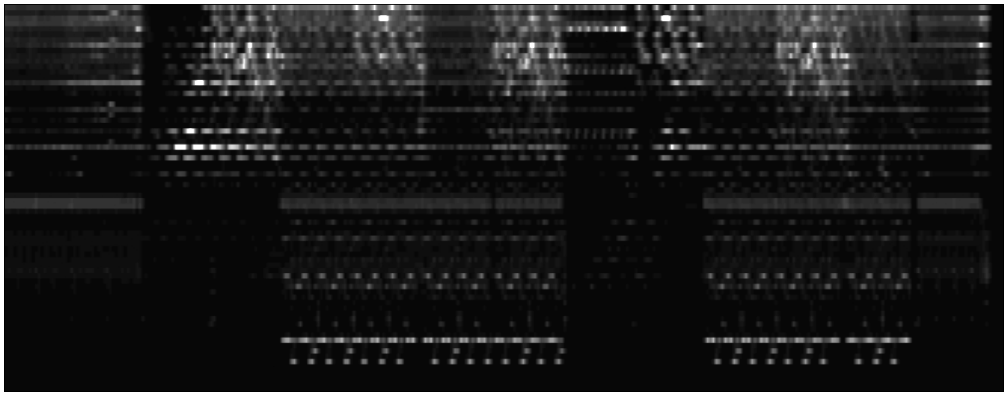

Figure 2.5: 72-bin chromagram of *Brighter Day* by Cyantific and Natalie Williams. The vertical axis denotes the musical note frequencies of the 72 bins. The horizontal axis denotes time. Lighter greys denote higher energy. This illustrates a 6-octave analysis of about 5 minutes of music.

Figure 2.6: A 12-bin chromagram obtained by summing the data in figure 2.5 at octave boundaries.

spectral leakage before running the FFT. To improve the time resolution of the output, and to avoid missing any musical events near the edge of the window function, the frames are overlapped as described by Noland (2009, p.38), starting each new analysis after $N/4$ samples. This sequence of frames defines the time axis in figures 2.5 and 2.6.

2.3.5 The chromagram

Having completed analysis frames over the duration of a recording, the result is a *chromagram* showing the energy found for each musical note over time, as illustrated in figure 2.5. This is a useful visualisation of the musical features of a recording, and is included in KeyFinder's user interface. However, the data can be simplified for further analysis by taking advantage of the cyclical properties of the musical spectrum discussed in section 1.2.2. The absolute octave to which each musical note belongs is discarded, preserving only their relative positions within a single octave, thus reducing 72 bins to a 12-bin vector for each analysis frame, as shown

in figure 2.6. From this point on, all analysis is based on this single octave chromagram.

2.3.6 Tuning

KeyFinder includes the option to start the spectral analysis with more than one frequency component per semitone, allowing for the capture of musical features slightly detuned from the A440 standard introduced in section 1.2.3. For example, if 3 bins per semitone are analysed, the central bin captures events at A440 and the bins on either side capture events detuned by ±33 cents (a cent is 1/100 of a semitone).

The result of this extra data capture is that at this stage in the process the chromagram has more than twelve bins per octave, and must be simplified before further analysis. KeyFinder implements two different algorithms for this simplification.

The first, described by Harte (2010, pp.71-73), is aimed at recordings where the piece of music is internally consistent but uses a tuning standard that differs from A440. This can be seen in some songs in The Beatles' back catalogue (for more detail on the music used in experiments, see section 4 below); for example, *Lovely Rita* is about a quarter of a semitone flat. The algorithm uses quadratic interpolation to determine the distance from A440 at which the majority of musical events occur; this essentially yields an offset from A440 by which the chromagram is adjusted.

The second algorithm, developed specifically for the analysis of electronic dance music, is aimed at recordings where some parts of the music are detuned from others. This happens occasionally in music made using samples of other recordings, like *People Everyday* by Arrested Development. For each semitone, the algorithm determines which of its bins has the highest energy, and uses this as the basis for the simplified single semitone bin. Rather than discarding the other bins, it adds some proportion of their value (one fifth, by default) to the simplified bin. This process is applied to each semitone, and the result is a 12-bin chromagram.

We do not discuss these algorithms in detail here because, as noted in section 4 below, they do not improve KeyFinder's results.

2.4 Segmenting music over time

KeyFinder achieves some good results by discarding the time dimension entirely at this point and making a key classification based on the features of the recording as a whole. However, as discussed in section 1.2.5, it is often an oversimplification to classify a piece of music as being in a single key; a piece may modulate between keys and the software should attempt to capture this.

KeyFinder therefore includes the option to segment a recording, primarily using key changes. These algorithms have enriched the user interface somewhat, as they allow the inspection of the different keys which influence a recording. Unfortunately, as with the processes described above that take account of detuning, these algorithms have not significantly improved KeyFinder's overall results, so they are not discussed in detail.

Firstly, KeyFinder includes a harmonic change detection function devised by Harte (2010, pp.77-82) for use in the detection of chord changes, using a chromagram as input. The basis for inclusion here is that, with different parameters, this function can detect the larger harmonic movements that occur at key changes.

Secondly, there is an arbitrary segmentation algorithm, which simply divides the chromagram into n segments of equal length along the time axis. While the divisions are obviously not based on music theory, they allow the global key of a recording to assert itself by limiting the effect of short passages that might otherwise have affected the overall key classification. As such this algorithm has demonstrated some success.

2.5 Key classification

With the chromagram divided as necessary, the time dimension is discarded, reducing each segment (or the whole recording, if no segmentation was applied) to a single 12-element chromatic vector. This represents the distribution of notes across the octave for the entire segment. There are a range of existing approaches to key classification which might be applied here, including geometric models based on the harmonic relationships within a scale (Chew 2002), machine learning techniques (Peeters 2006), and tone profiles based on psychoacoustic perception studies (Krumhansl 1990). The final approach, classification against tone profiles, has received a lot of attention as well as some noted revision (Temperley 1999; Gómez Gutiérrez 2006), and it has been successfully applied in many previous experiments (Gómez and Herrera 2004; Pauws 2004; İzmirli 2005; Peeters 2006). For these reasons it is implemented as the basis of KeyFinder's classification algorithm.

2.5.1 Tone profiles

A series of psychoacoustic experiments by Krumhansl (1990, p.21) involved playing incomplete scales (C major and C minor) to musically trained listeners, followed by a final tone taken at random from the chromatic scale. The listeners were asked to rate how well the final tone "fit with the context in a musical sense" (Krumhansl 1990, p.27). The results show that the listeners rated the tones within the scale higher than those without, and that the tonic, fifth and third were rated particularly highly (see figure 2.7).

Krumhansl (1990, p.77-81) goes on to describe a key-finding algorithm based on these results. Twenty-four tone profiles are generated, one for each key, by transposing these major and minor profiles to each of the twelve positions of the chromatic scale. Each profile is then correlated in turn against a representation of a passage of music similar to an analysis vector from the chromagram (Krumhansl actually derives her musical data

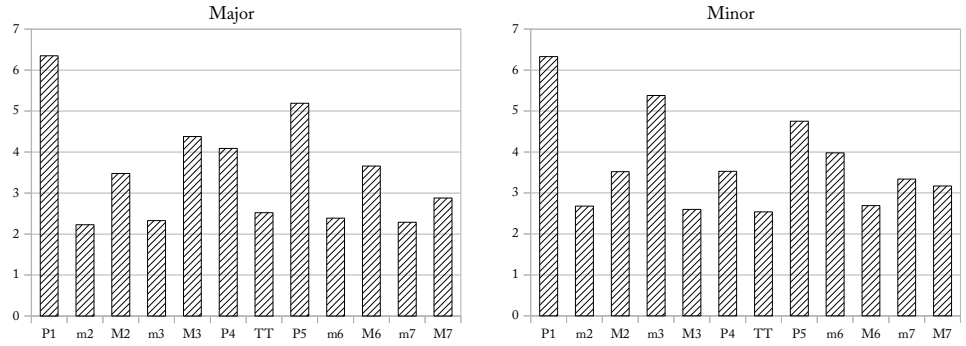

Figure 2.7: Krumhansl's (1990, p.30) tone profiles. Note the variation between major and minor, particularly that the weight given to the notes included in each scale (as discussed in section 1.2.4) is much higher than to those excluded.

from a symbolic representation – sheet music – rather than a recorded performance, but the principle is the same). The correlation score c_{xy} between an input vector x and a tone profile y is given by

$$c_{xy} = \frac{\sum_{n=0}^{11}(x_n - \bar{x})(y_n - \bar{y})}{\sqrt{\sum_{n=0}^{11}(x_n - \bar{x})^2 \sum_{n=0}^{11}(y_n - \bar{y})^2}} \tag{2.9}$$

where $\bar{x}$ is the mean of the input vector values and $\bar{y}$ is the mean of the tone profile values. The profile with the highest correlation score yields the estimated key.

KeyFinder's implementation of tone profile classification includes the correlation score method, as well as an alternative, slightly more efficient cosine similarity method given by

$$s_{xy} = \frac{x \cdot y}{\| x \| \| y \|} \tag{2.10}$$

This yields the cosine of the angle between the two vectors x and y; close vectors will score near 1 while orthogonal vectors will score 0.

The choice of tone profiles has a dramatic effect on the results of the key estimation algorithm. KeyFinder includes Krumhansl's profiles, as well as variations by Temperley (1999, p.74), Gómez Gutiérrez (2006, p.107), and a new set devised during the project.

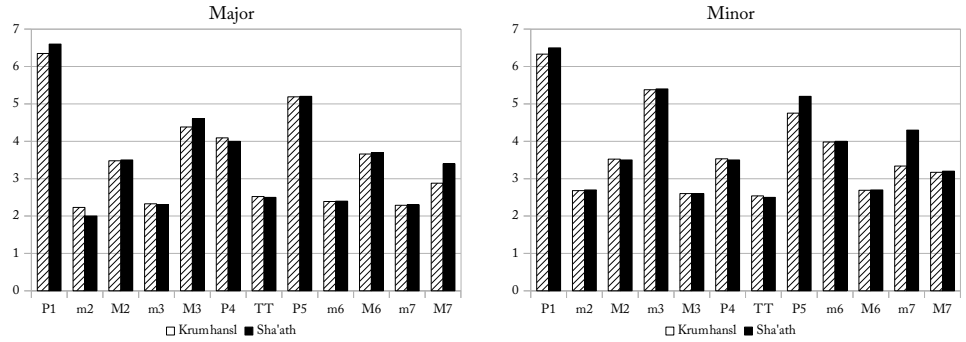

Figure 2.8: Variations applied to Krumhansl's tone profiles to increase accuracy. The most significant changes are to the tonic (P1), and to the final diatonic element in each vector (M7 and m7 respectively).

The new profiles (figure 2.8) were arrived at empirically. They are a slight adjustment of Krumhansl's profiles and have yielded greater accuracy (see section 4 below).

Given the importance of the tone profile to the musical analysis, the KeyFinder interface also includes the option for the user to specify custom profiles for key classification.

2.5.2 A final key estimate

If no segmentation has been applied, the classification against tone profiles provides a key estimate for the whole recording. In a segmented recording, a score is assigned to each of the keys estimated, based on the duration of their segment(s) divided by the energy captured in the chromagram over the duration of the segment(s). This weights the keys of longer and louder passages of music over shorter, quieter passages. The key with the highest score is picked as the final key estimate for the recording.

2.6 Summary

This section has outlined the theoretical background for the KeyFinder implementation and the algorithms it employs. The next section describes

the software engineering processes applied during the project. Thereafter, section 4 presents experimental results demonstrating the effects of these algorithms, and justifying the choices of the default values for their key parameters.

3 Software implementation

This section describes the software engineering process and principles for the KeyFinder implementation. The choice of libraries is discussed, and the design of interfaces and classes is outlined. The testing strategy is described.

3.1 Implementation principles

3.1.1 An object-oriented approach

KeyFinder is implemented using object-oriented programming (OOP) techniques, for a number of reasons. The procedure for estimating a recording's key can be easily decomposed into shorter, specialised actions, each of which is concerned with a different family of algorithms or operations, and often with a different set of data. This is reflected in the structure of section 2 of this report and reviewed in figure 3.1. Such a structure lends itself to encapsulation of the data storage into objects and the abstraction of interchangeable families of algorithms into class hierarchies.

3.1.2 Design patterns

Since most operations are implemented in a number of ways, the Strategy design pattern is used extensively. This encapsulates a family of algorithms and makes them interchangeable, so the appropriate algorithm can be chosen at runtime (Freeman et al. 2004, p.24). For example, as noted in section 2.3.1, the results of the Fast Fourier Transform need to be post-processed to be useful in a musical analysis, but KeyFinder has two ways

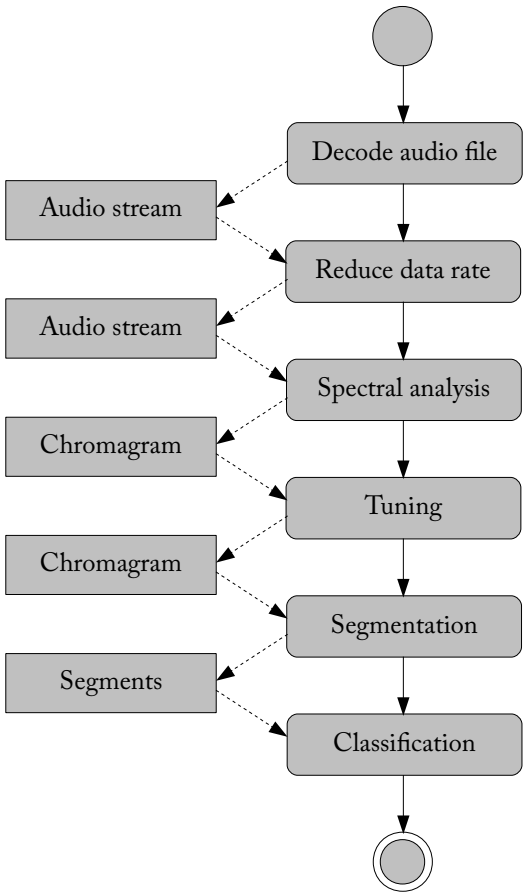

Figure 3.1: Activity diagram for key estimation process

of accomplishing this; with Brown and Puckette's Constant Q Transform, or with the more efficient directly-computed spectral kernel. These two algorithms are implemented with a common interface, inherited from an abstract parent class, so the choice of which implementation to use can be made at runtime. The use of this design pattern makes KeyFinder's computational model extensible, since it is straightforward to implement new algorithms conforming to the same interface.

The actual instantiation of these objects is handled by a *factory*, an OOP concept that forms the foundation of several design patterns as discussed by Freeman et al. (2004, p.117). In the case of KeyFinder, the factories are static methods that decouple the instantiation of concrete implementations from the classes that use them. Rather than instantiating an object itself, the executing code simply calls the factory method and is passed an object of the appropriate class. This reduces coupling, since the executing code is not required to know any more than the interface which the concrete class implements.

The initialisation of some objects is relatively expensive, as in the creation of KeyFinder's spectral analysis kernels. Here the factory employs the Singleton pattern, which ensures that only one instance of a class can exist (Freeman et al. 2004, p.177). In this case, the Singleton is a globally accessible repository of spectral analysers, each of which is instantiated just once and persists for reuse.

The user interface is based on the Model-View-Controller architecture, isolating the application logic in a worker model which asynchronously passes data to the views as it becomes ready.

3.2 Choice of language and libraries

KeyFinder is written in C++, due to the author's familiarity with the language, the requirement for high performance and OOP principles, and the need to maintain cross-platform portability. Furthermore, most of the mature and widely supported libraries in the problem domain are written

in C and C++ .

To decode audio files, KeyFinder initially used *LibSndfile* version 1.0.23 (de Castro Lopo 2010), and later migrated to *LibAV* version 0.7 (LibAV community 2011), both written in C. One of the sample rate conversion algorithms employs *Secret Rabbit Code* version 0.1.7 (de Castro Lopo 2009), and the primary spectral analyser integrates *FFTW* version 3.2.2 (Frigo and Johnson 2009), both in C. *TagLib* version 1.7 (Wheeler 2011), written in C++, is integrated for the reading and writing of audio file metadata.

After an initial command line implementation for Linux, KeyFinder was ported to the Apple Mac and the Qt[1] framework, version 4.7.4 (Nokia et al. 2011). This framework was initially chosen because it enables the development of user interfaces native to Mac OS X's Cocoa API, without the need to port the original C++ code to Objective-C. Several other advantages presented themselves over the course of development, most notably the simplicity of implementing multithreading.

3.3 User interface

KeyFinder has two primary interfaces, each developed to implement one of the core use cases (see section 1.3.1).

The first thing the user sees upon launching the software is the batch interface (figure 3.2). This is a straightforward tabular presentation, which is populated with the paths and metadata of audio files dragged onto the window. There is a single, obvious button to begin a batch estimation process on the files, which activates a progress bar indicating the proportion of files completed (figure 3.3). The interface is intended to match the user's expectations for batch processing, and as such has common features like copying text data to the clipboard, the option to revert to a previous state and re-run a batch, and alerting the user with a sound when a batch job is complete.

[1] Pronounced "cute".

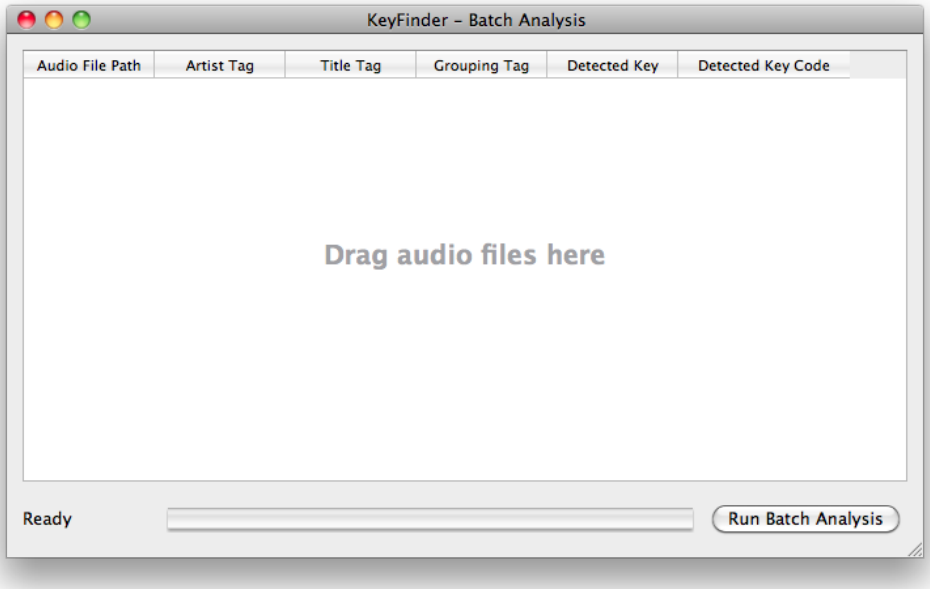

Figure 3.2: Screenshot of the batch processing interface at launch

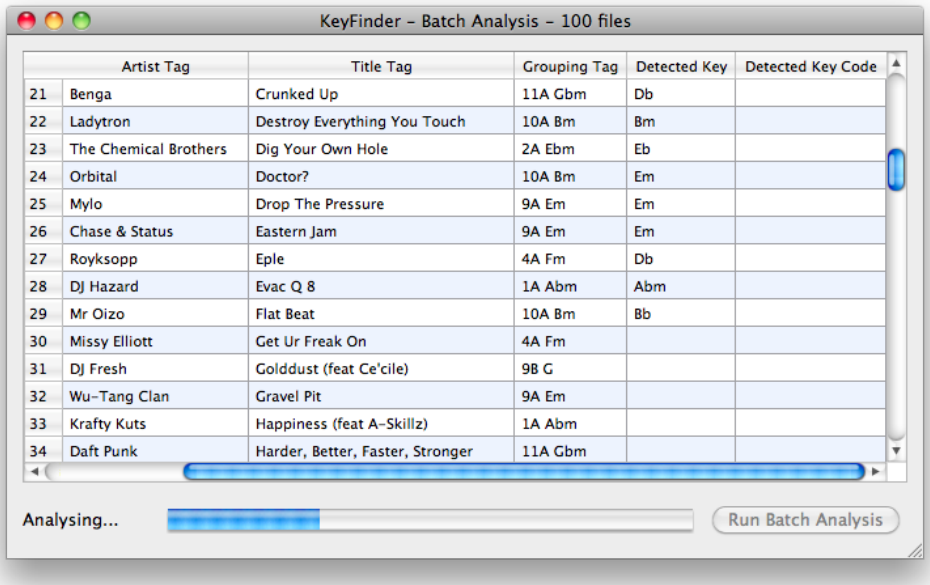

Figure 3.3: Screenshot of the batch processing interface in operation

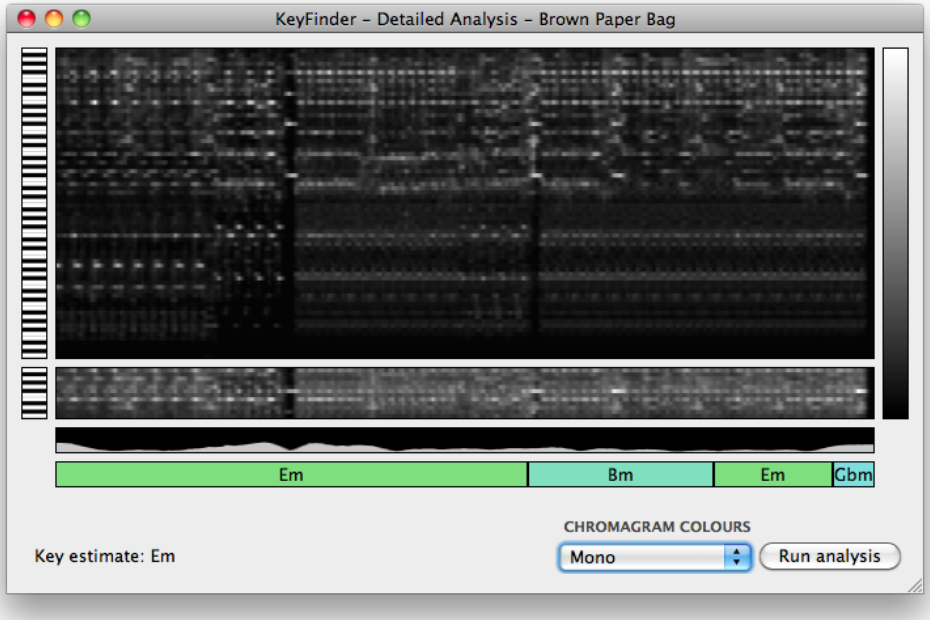

Figure 3.4: Screenshot of the detailed analysis interface. Recording illustrated is *Brown Paper Bag* by Roni Size Reprazent.

The other main feature the batch interface offers, accessible via shortcut key or context menu, is to launch the detailed analysis interface for a selected audio file.

The detailed analysis window (figure 3.4) aims to illustrate the workings of the KeyFinder algorithms. Working from the top of the screen down, it highlights the key stages of the analysis. First and most prominently, a full chromagram (72 bins by default) is shown, which offers the user a quick visual association with the recording; it is normally possible to immediately distinguish the major sections of a song (verse, chorus, and so forth). The time axis of this chromagram is maintained throughout the other elements vertically aligned on the screen.

The next element down the screen is the reduced 12-bin chromagram, which is the basis for further analysis. Both the chromagrams are aligned to a representation of a piano keyboard, indicating the mapping of the data to musical notes. This can often enable a musically skilled user to determine the tonal content of the piece by sight, and it was found during

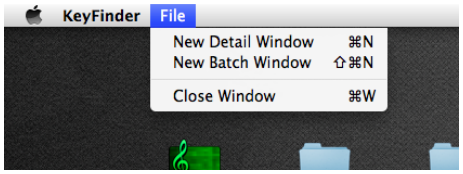

Figure 3.5: Screenshot of the KeyFinder menu bar.

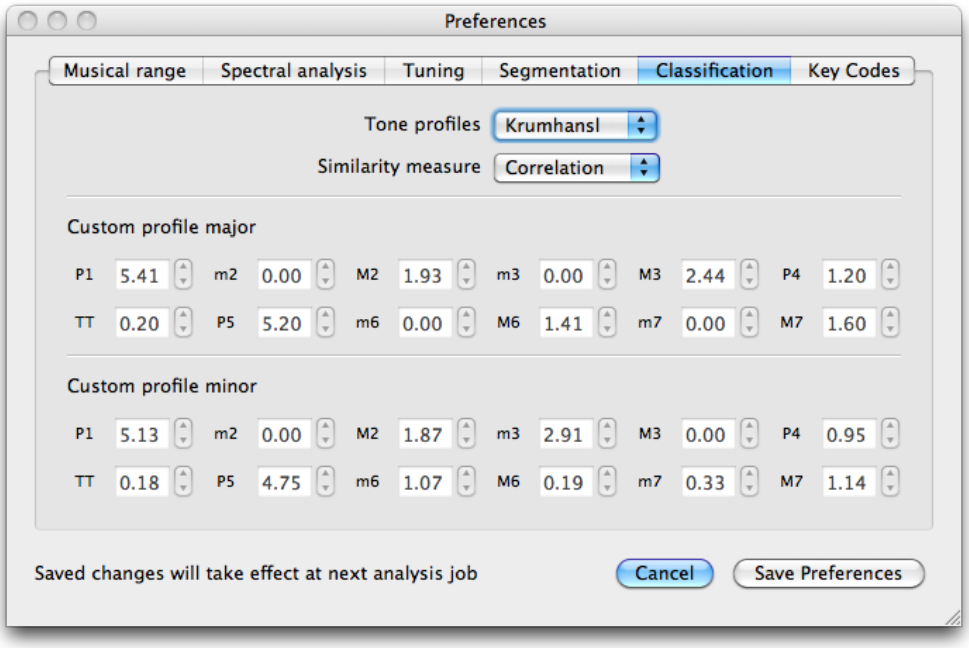

Figure 3.6: Screenshot of a section of the Preferences pane.

experiments to be a very useful feature when KeyFinder's estimations were incorrect.

The last two elements of the interface are useful if segmentation is enabled (see section 2.4). The first is a representation of the rate of harmonic change in the musical signal; the peaks used for segmentation are usually obvious. The final element displays the segmentation yielded by the rate of change curve, and the key classifications for each segment. Key classifications are colour coded, so that an incorrect estimation for some segment of the recording will usually appear immediately incongruous. The final key estimate is rendered at the bottom of the screen.

These interfaces are supported by a standard menu bar (figure 3.5), which

includes the ability to open multiple copies of each window, as well as launching the Preferences pane (figure 3.6), which allows the user to configure the many parameters of the KeyFinder algorithms.

3.4 Class overview

KeyFinder's computational model is primarily managed by a single class, the `KeyFinderWorkerThread`. This is derived from the Qt framework's `QThread` class, allowing all analysis work to be carried out in a distinct thread without affecting the operation of the user interface.

The same model is employed by both primary GUI windows. The worker thread sends a signal to its parent window when some data element is ready for display (e.g. a chromagram, the rate of change curve, or a key estimate). The batch window is uninterested in most of these, dealing only with the final key estimate, but the detail window receives and displays each element in turn.

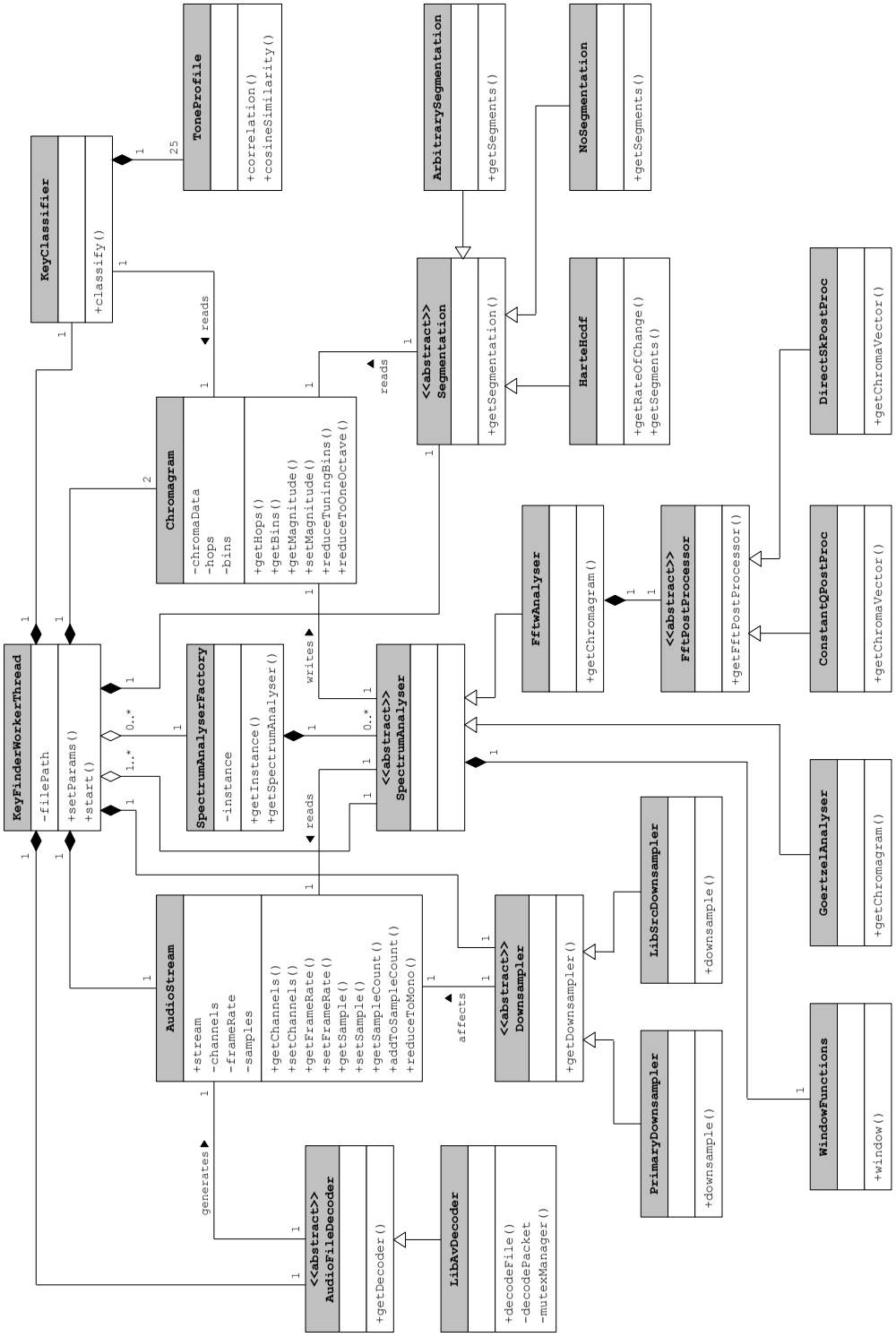

Figure 3.7: Class diagram for the KeyFinder estimation model

The class diagram in figure 3.7 illustrates the interaction between the worker thread and the main algorithms. For the sake of legibility, only the most important properties and methods are shown. Also omitted from this diagram is the `Preferences` class, which contains the parameters chosen in the preferences dialog described above. The parameters are initially set for a thread of analysis with the `setParams()` method of `KeyFinderWorkerThread`, and are passed around between the operational classes during the thread's execution.

Note the application of the Singleton pattern for the `SpectrumAnalyserFactory`, the Strategy pattern for the `AudioFileDecoder`, `Downsampler`, `SpectrumAnalyser`, `FftPostProcessor` and `Segmentation`, and the implementation of static factory methods for each of these abstract classes.

The interaction between the user interface and the core computational model is illustrated by figure 3.8. Note that each window instance has its own worker thread, and that the `Preferences` class is composed into most other objects (this is also the case, though not illustrated, with the core computational objects).

3.5 Testing

Functional testing was carried out by means of a series of regression tests and by unit testing of the core algorithms. The Qt framework's support for automated unit testing is sadly lacking, so the unit tests are compiled as a separate binary when certain build options are set for the project.

The unit tests include, but are not limited to:

- audio format decoding: the same known waveform was transcoded into multiple formats and the accuracy of the decoding is tested in each case;
- downsampling: the effect of each downsampling algorithm on a known waveform is tested;

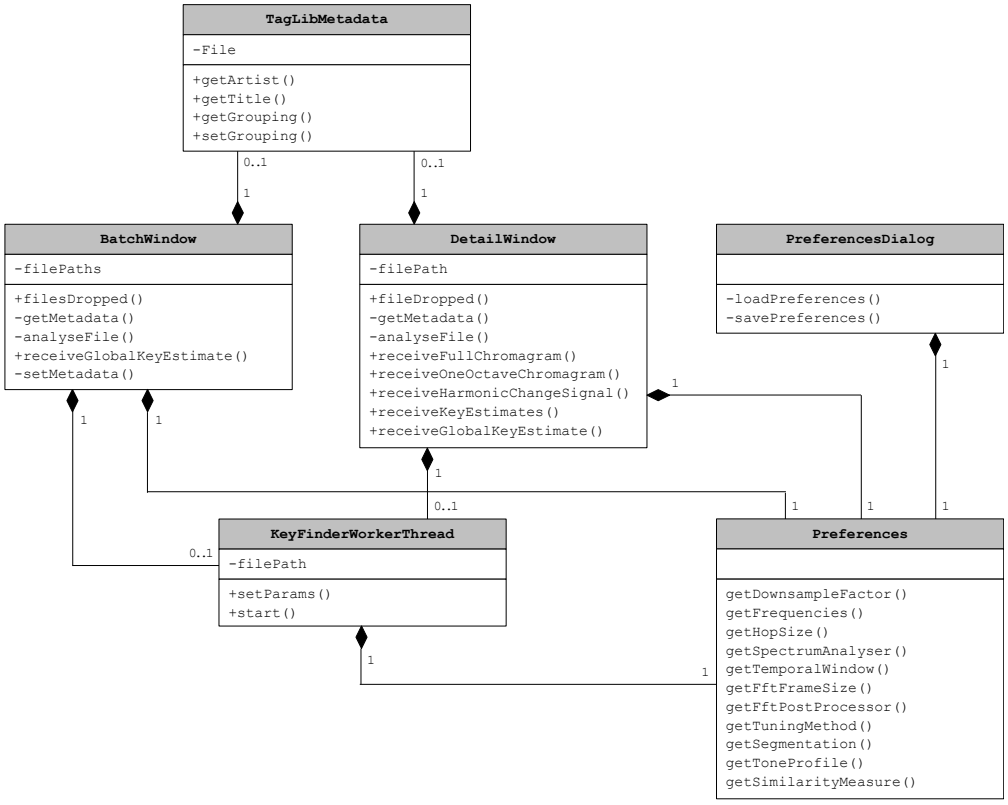

Figure 3.8: Class diagram for the main user interface

- spectral analysis: the output of the spectrum analysers for a known tonal input is tested;
- tuning: the effect of each tuning algorithm on a known chromagram is tested.
- functions of the main data storage classes: all mutators of the `AudioStream` and `Chromagram` are tested for the normal flow and exceptional cases.

The regression testing involved ensuring that the KeyFinder application continued to return the same results for a given data set of a few hundred songs, given the same parameters; this was completed for each major build.

Non-functional testing was mainly limited to informal tests of the parallelisation, resource use and speed of the application. In particular, it was ensured that the software dealt gracefully with concurrent access by a number of threads to a single resource (e.g. an external audio file or internal spectral analysis kernel), that successive builds made much the same use of system memory for a given task, and that the time taken to complete a known task remained similar.

3.6 Summary

This section outlined the software engineering principles underlying KeyFinder, and described the design and testing of the system and the integration of library code. Section 4 below presents the results of the experiments conducted to test KeyFinder's accuracy, and shows the reasoning behind the default values for the software's parameters.

4 Experiments

This section describes the experiments carried out to test and optimise KeyFinder's performance. It introduces the data sets used for these experiments and the scoring method, and presents the results of varying the key parameters of the algorithms. KeyFinder's performance is compared to two popular existing applications.

4.1 Data sets

The primary data set is a collection of 100 dance music recordings. These were randomly selected from the author's own DJ playlists; the key of each recording was manually classified and verified by another musician. Where there was disagreement, a key classification was arrived at with the help of another musically-trained expert.

The vast majority of the recordings are in minor keys, which is a significant characteristic of dance music generally. They mostly have very few, if any, key changes, and are almost all tuned to the A440 standard (though several songs are composed of samples of other recordings, which are sometimes tuned inconsistently with other elements). There are several genres represented, and the music is split fairly evenly across the last two decades (with one exception that dates from 1976).

The secondary data set is drawn from The Beatles' back catalogue; 179 songs from 12 albums. This collection is not particularly relevant to the DJ, but is in common use by the music information retrieval community. As such there were existing key classifications (Mauch et al. 2009) which serve as ground truth for these experiments.

Table 4.1: MIREX score weightings

Key relation	Score
Exact match (tonic)	1.0
Perfect fifth (dominant)	0.5
Perfect fourth (subdominant)	0.5
Relative major/minor	0.3
Parallel major/minor	0.2

The data sets, which are composed of uncompressed WAV files, are listed in full in appendices A and B.

4.2 Measuring success

As stated in section 1, the goal of this project is to achieve the best possible accuracy in key estimation for DJs. The experiments below all measure the accuracy of KeyFinder's results as compared to the ground truth classifications.

Due to its relevance to the problem domain, accuracy in the primary data set is favoured over the Beatles collection. The secondary data set serves mainly to ensure that KeyFinder's configuration is not overly biased towards dance music, to the exclusion of its possible application to other musical genres.

The method of scoring used below was developed for the Music Information Retrieval Evaluation eXchange (MIREX), an annual evaluation campaign for music information retrieval algorithms (ISMIR 2005). This method gives full credit to key classifications which match the ground truth exactly, and partial credit to estimations closely related to the correct answer (see table 4.1; these relationships are introduced in section 1.2.7). As noted by Noland (2009, p.53), the MIREX measure is more valuable than a simple count of exact matches, being "more closely related to both music theory and perception than one that gives a zero score to both the dominant key and the key furthest away on the circle of fifths".

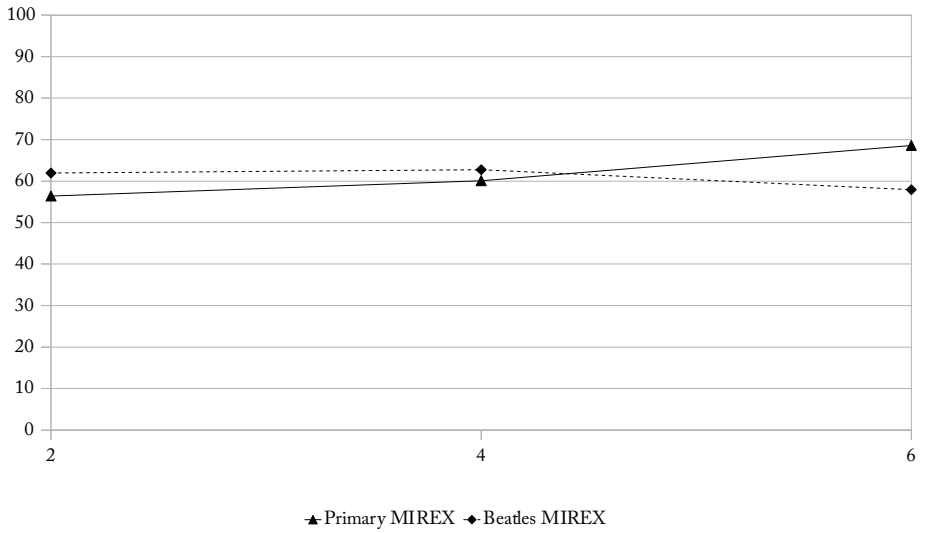

Figure 4.1: Frequency analysis range. The horizontal axis denotes the number of octaves analysed.

In each experiment, MIREX scores are given for both collections. Since the primary data set contains 100 songs, the scores for the Beatles collection are normalised (divided by 1.79) for ease of comparison.

4.3 Parameter testing

Several parameters have an effect on the overall accuracy of KeyFinder. These are considered in the order they are applied (see section 2.1). The full parameters for each experiment are listed in appendix C.

4.3.1 Frequency analysis range

The first parameter to consider is the breadth of frequencies to analyse. Figure 4.1 illustrates the effects of broadening the analysis range, starting with two octaves from C3 to B4, and adding an octave on either side for each step. The results for the primary data set improve as the more extreme bass and treble are included, while the Beatles collection is not

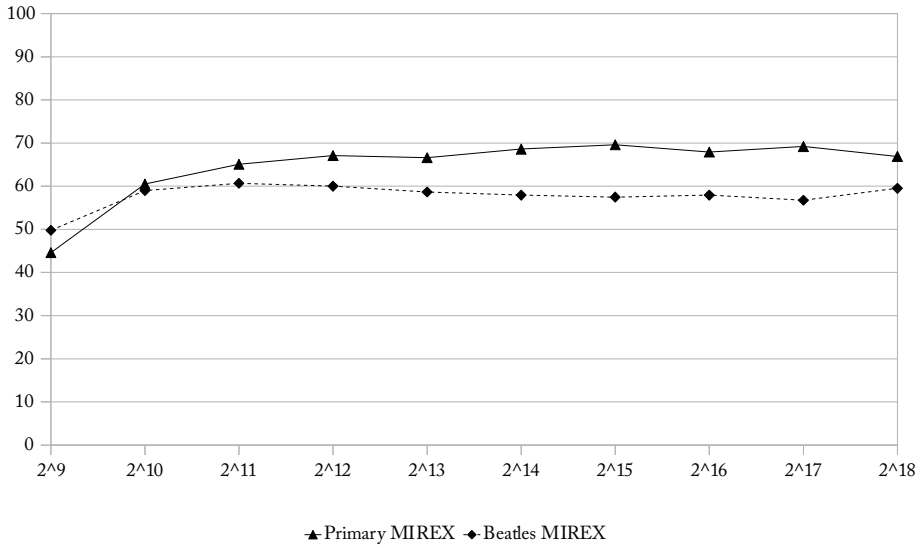

Figure 4.2: Spectral transform resolution. The horizontal axis denotes the number of samples in the Fast Fourier Transform, expressed in powers of 2. Hop sizes are always 1/4 of the analysis frame size.

significantly affected. Continuing to add further octaves beyond 6 was found to be detrimental to the results from both collections, so this parameter is set at 6 octaves (from C1 to B6, or ∼32.7 Hz to ∼1975 Hz) henceforth.

4.3.2 FFT resolution

As discussed in section 2.3.1, the number of input samples N to the Fast Fourier Transform determines the resolution of its frequency domain output. In a musical analysis, the most obvious advantage of a high resolution is greater accuracy at the lower, bass end of the spectrum.

KeyFinder's accuracy is charted for a range of values of N in figure 4.2. For our default starting frequency, the practical minimum for N is 2^{11}, since any lower power of two provides insufficient spectral resolution to distinguish between adjacent notes of the lowest octaves. Beyond this, there is little difference in the results. The default is set at 2^{14}, which provides a reasonable analysis length for capturing musical features; analysis

Figure 4.3: Comparison between results of Constant Q Transform and Direct Spectral Kernel Transform. The DSK parameter p (see equation 2.6). is set to 3.8.

frames of ∼4 seconds, at intervals of ∼1 second.

4.3.3 Spectral kernel bandwidth

As noted in section 2.3.3, the directly-computed spectral kernel (DSK) developed during this project can very closely model the Constant Q Transform; figure 4.3 illustrates the similarity between the results of the CQT and the new algorithm when properly parameterised.

Figures 4.4 and 4.5 show the most important aspect of the DSK: the parameter p that varies the bandwidth of the spectral windows enables a significant increase in accuracy. Note in particular the effect illustrated by the chromagrams in figure 4.5; when p is reduced there is a much tighter focus on the frequencies of interest, eliminating the vertical smearing between bins caused by the large overlaps between the wider spectral windows (as illustrated in figure 2.4 above). The default value for this parameter is set at 0.8.

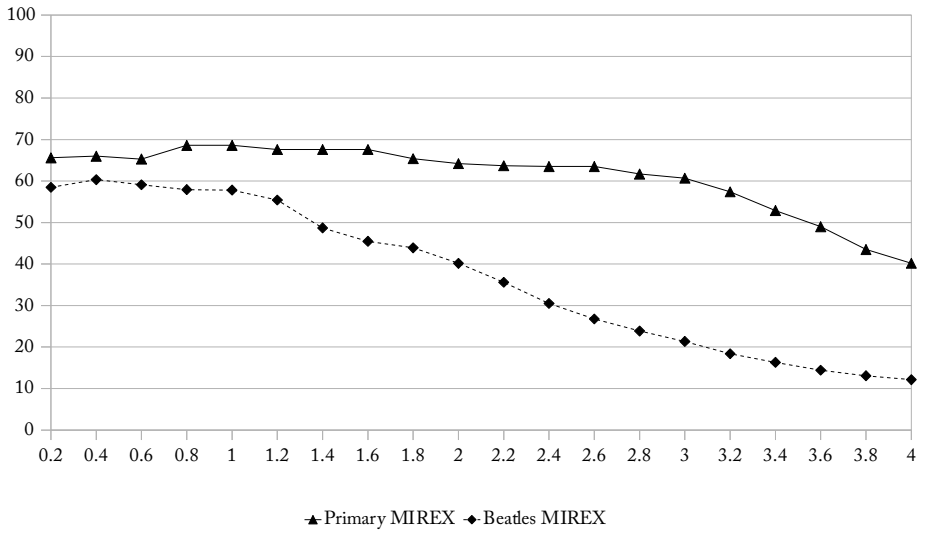

Figure 4.4: Results of narrowing spectral kernel bandwidth. The horizontal axis denotes the parameter p (see equation 2.6).

4.3.4 Tuning algorithms

As briefly noted in section 2.3.6, neither of the tuning algorithms improved KeyFinder's results. In fact, as illustrated in figure 4.6, Harte's algorithm reduced accuracy on the primary data set significantly, though it did improve the results for the Beatles by nearly as much. This is perhaps not surprising, as the Beatles collection was the primary basis for Harte's own experiments (Harte 2010, p.223). The bin-adaptive algorithm developed during this project has little effect on the results for either collection (figure 4.7). The default for KeyFinder is therefore to generate an initial chromagram with only one bin per semitone, so that no accounting for tuning is required.

4.3.5 Segmentation

As illustrated in figure 4.8, segmentation of the chromagram has no positive effect on KeyFinder's accuracy. The default behaviour is therefore not to segment the recording in the time dimension.

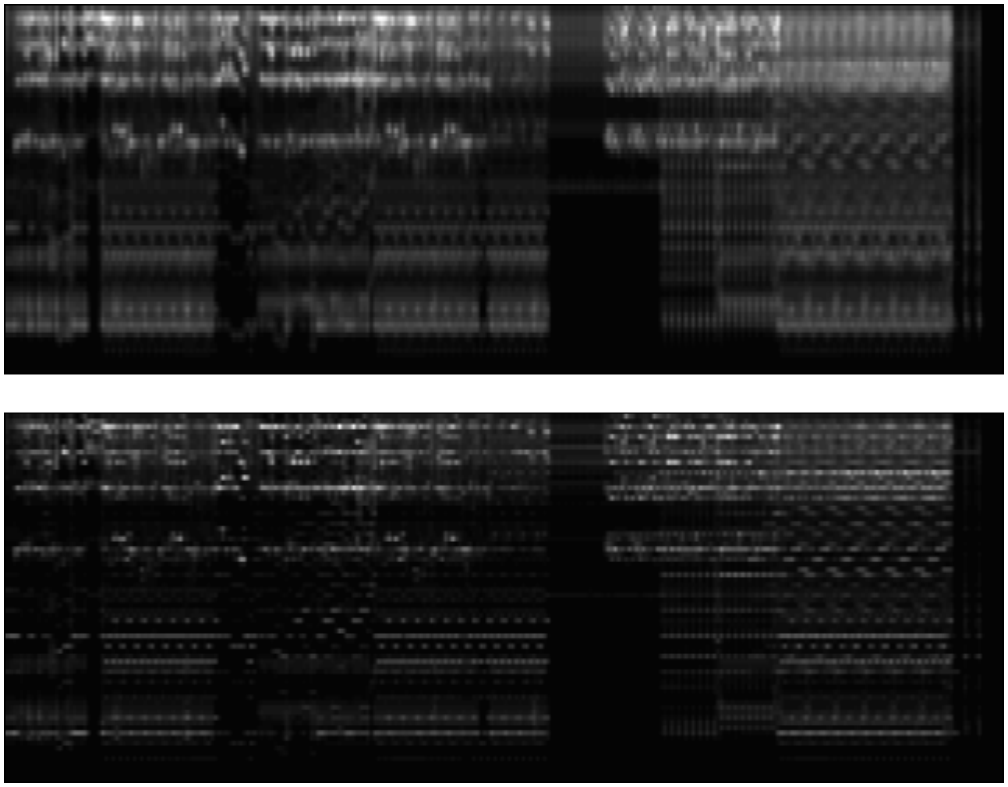

Figure 4.5: Chromagrams demonstrating spectral kernel bandwidth parameterisation. The top image shows the output of the DSK with the parameter $p = 3.8$, which closely models the Constant Q Transform. The bottom image has $p = 0.8$, which is the default for KeyFinder. Recording illustrated is *Je Veux Te Voir* by Yelle.

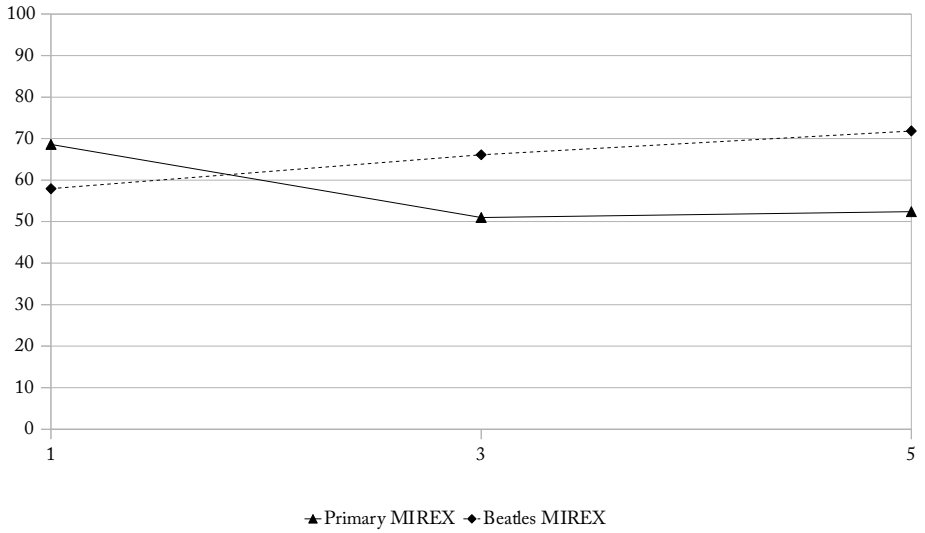

Figure 4.6: Harte's tuning algorithm. The horizontal axis denotes the number of bins per semitone; when this is 1, no tuning is applied.

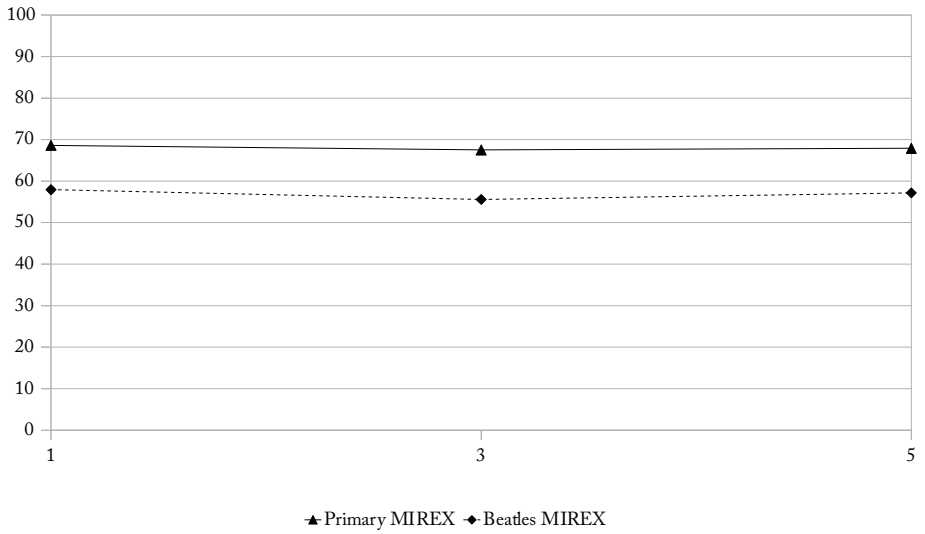

Figure 4.7: Bin-adaptive tuning algorithm. The horizontal axis denotes the number of bins per semitone; when this is 1, no tuning is applied.

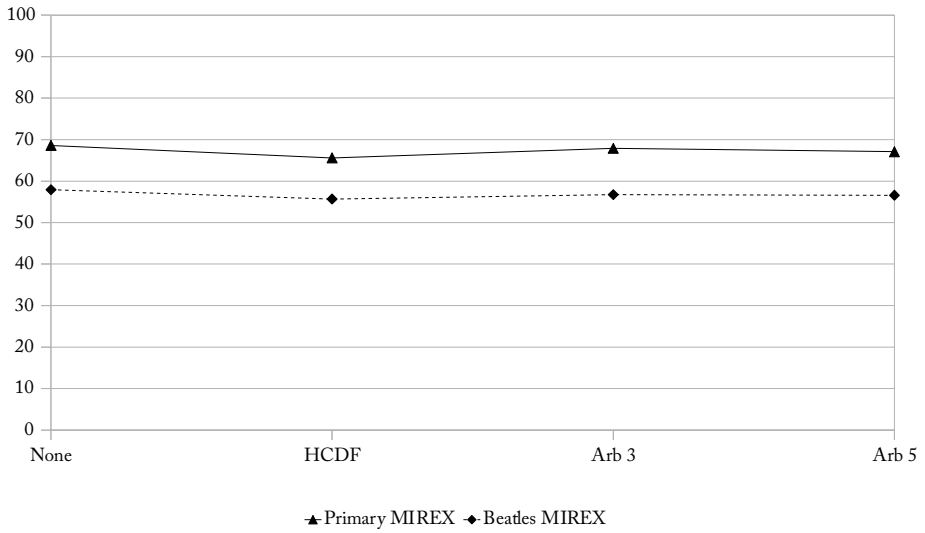

Figure 4.8: Effects of chromagram segmentation. The horizontal axis denotes the type of segmentation applied; from left to right: no segmentation, Harte's harmonic change detection function, arbitrary division into 3 equal segments, and arbitrary division into 5 equal segments.

4.3.6 Tone profiles and similarity measure

The method chosen for the key classification has a greater effect on KeyFinder's results than any other parameter, as illustrated in figure 4.9.

It is worth noting that the profiles from the literature tend to provide much greater accuracy for the Beatles than for the dance music collection. This validates the motivation of this project in designing a system primarily for the analysis of dance music, but it also justifies enabling the user to select from a range of profiles (and to define their own) to best match their own music collection. It is clear, even from the limited contrast afforded by these data sets, that there are different tonal characteristics to be found in different musical genres.

KeyFinder defaults to using the author's profiles since they provide the best accuracy for the primary data set. Also, the cosine similarity measure is shown to be consistently more effective than correlation for the primary data set, and is used as the default.

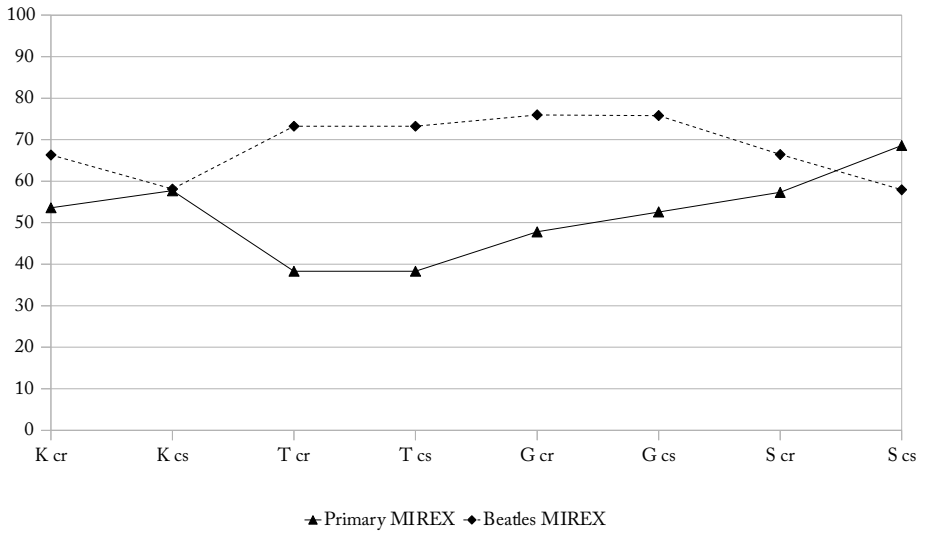

Figure 4.9: Tone profiles and similarity measure. On the horizontal axis, K denotes profiles by Krumhansl, T Temperley, G Gómez Gutiérrez and S the author. The suffix cr denotes the correlation similarity measure, and cs denotes the cosine similarity measure.

4.4 Comparison to other software

There are existing free and commercial key estimation software products aimed at DJs, but those in common use are proprietary and their algorithms have not been published. In order to place KeyFinder's results in context, figure 4.10 presents comparative analyses of the data sets using recent versions of the two most popular packages, Rapid Evolution (free, version 2.13.13 and version 3 beta 52) and Mixed In Key (commercial, version 4.1). More details of the results are listed in tables 4.2 and 4.3.

KeyFinder's default parameters provide significantly greater accuracy for the primary data set than the other packages. Its results for the Beatles data collection are the lowest of the four (though comparable to the nearest competitor, Mixed In Key), but KeyFinder's flexibility is its strength here. As shown in figures 4.6 and 4.9, KeyFinder can be configured to almost match the accuracy of Rapid Evolution on the Beatles recordings; and it scores several points higher on the primary collection even with those

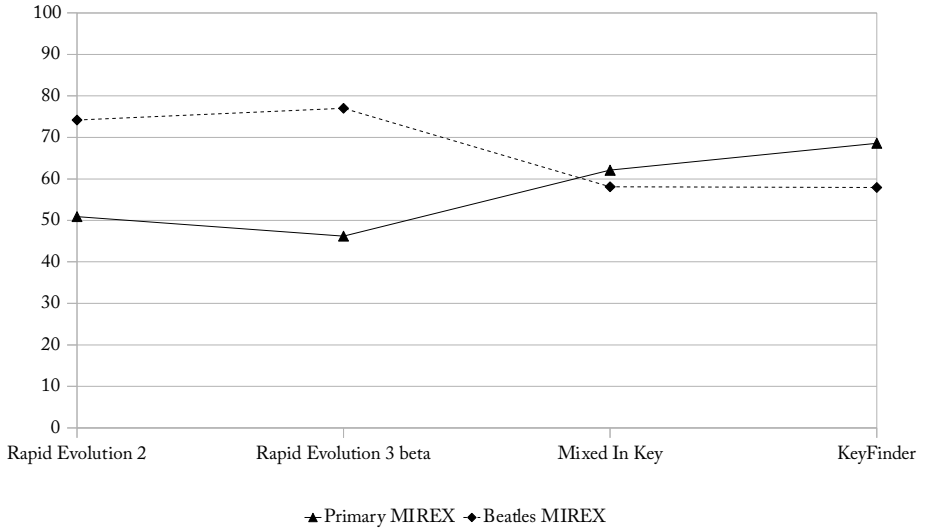

Figure 4.10: Comparison of KeyFinder to existing tools.

Table 4.2: Detailed results from software comparison, primary data set

	RE 2.13.13	RE 3b52	MIK 4.1	KeyFinder
Exact matches	39	36	51	59
Related by perfect fifth	6	4	7	2
Related by perfect fourth	7	6	8	11
Relative key	6	6	6	5
Parallel key	18	17	9	8
Incorrect	24	31	19	15
MIREX score	50.9	46.2	62.1	68.6

Table 4.3: Detailed results from software comparison, Beatles data set

	RE 2.13.13	RE 3b52	MIK 4.1	KeyFinder
Exact matches	118	124	79	83
Related by perfect fifth	3	3	4	1
Related by perfect fourth	19	18	18	27
Relative key	4	4	26	17
Parallel key	13	11	31	8
Incorrect	22	19	21	43
MIREX score	132.8	137.9	104.0	103.7

parameters.

KeyFinder also works much faster than the other packages. To analyse the author's entire music collection of 5604 audio files, KeyFinder took 4 hours 45 minutes, while Rapid Evolution 2 took 11h 56m and the Rapid Evolution 3 beta release took 12h 47m. It is not possible to make an entirely fair comparison with Mixed In Key since it also analyses recordings for tempo during a batch job, but it took 13h 14m to complete the task. These experiments were run on a 2007 MacBook Pro with a 2.4GHz Intel Core 2 Duo processor and 4GB of RAM.

4.5 Summary

This section presented the results of the project's key experiments. It described the ground truth data and scoring method used for measuring success, and illustrated how the default parameters for the KeyFinder software were chosen to maximise the accuracy of the results. A comparison with the results of the most popular existing applications was presented, in which KeyFinder scored higher on the primary data set than the other packages.

The last section concludes the project report by considering the contributions and limitations of the project, as well as possible further work.

5 Conclusions and future work

This report describes the design and implementation of KeyFinder, a software package which can estimate the key of digital music recordings, with a particular focus on modern dance music and the workflow of the DJ. This final section brings together the conclusions of the project, and considers opportunities for further work on the problem.

5.1 Contributions

The most obvious result of the project is that KeyFinder demonstrates some success in improving key estimation accuracy for dance music, performing significantly better than the most popular implementations in the field for a small data set.

The software makes other novel contributions to the DJ community. The many parameters controlling KeyFinder's algorithms are customisable by the user, unlike in the other software tested; this may enable tailoring to particular musical genres, as we have seen to some extent with the secondary data set.

The detailed analysis interface not only illustrates the working of the key estimation algorithms, but provides a general purpose musical visualisation tool apparently unique in the DJ software world.

In terms of contributions to the wider music information retrieval field, the project presents an efficient, flexible alternative to a spectral transform algorithm used in many other audio analysis tools.

5.2 Limitations

The least satisfactory aspect of the project is the size of the data set. 100 recordings is too small a sample size to be confident in KeyFinder's results, as it may have led to a bias in the development and parameterisation of the algorithms that will limit their usefulness. Unfortunately the time scale of this project did not allow the manual classification of a larger selection of recordings, and there are apparently no pre-existing annotated data sets of dance music that have an appropriate or consistent level of accuracy. Even the Beatles collection, extensive and varied though it is, seems an obviously limited sample.

The tone profile correlation method of key estimation employed by KeyFinder is less favoured in some recent work such as Noland's (2009, p.57). It has clear limitations as it does not capture the order, progression or other context of musical events, which certainly play a part in a human's estimation of key. While the project has had some success in applying the tone profile model, it would have been ideal to consider other methods as well.

With regards to the software implementation, circumstances did not allow the testing of many audio file formats[1], though in theory the integration of the LibAV libraries should enable the analysis of a very broad range of containers and codecs.

One of the stated requirements for the software was portability to Microsoft Windows. This should not be particularly complicated given the use of the Qt framework, but it has not been attempted.

A minor bug in the current implementation is that files whose names contain non-ASCII characters cannot be analysed. Again, this should be possible to fix, but was not a priority during the project.

[1] WAV, AIFF, MP3, MP4 with ALAC encoding, and FLAC were tested during the project.

5.3 Future work

The project has highlighted some areas where further work would be beneficial. Most obviously, the development of a large, trustworthy data set of expertly annotated dance music would simplify the work of development and testing in future[2].

The similarity measures employed by KeyFinder in the classification against tone profiles assume that chroma vectors exist in a uniform space, but the literature shows other ways of modelling spaces, reflecting the principles of music theory and aspects of human acoustic perception. For example, Harte's (2010, p.77-79) harmonic change detection function places chroma vectors in a space whose dimensions map close harmonic relationships like perfect fifths and major/minor thirds to small Euclidean distances, and distant harmonic relationships to larger distances. Since this is already implemented in KeyFinder, there may be value in investigating the application of this spatial model, or something like it, to tone profile classification.

Perhaps the most practical direction for KeyFinder in its current form is in developing parameter sets that are known to yield good accuracy for particular musical genres. The divergent results for the primary and Beatles collections in some of the experiments show that there are methods that work well for certain styles but not for others. Another observation made during experimentation is that hip-hop music (which did not make up a large proportion of the data set) is difficult to analyse, as rap vocal parts are usually loud and atonal. Informal experimentation suggested that parameterising the analysis to discard the vocal frequency range for those recordings yielded better accuracy. There may be other such optimisations for a variety of styles, which could be saved as user-friendly patches and shared around the community.

[2]The author of the Rapid Evolution software hosts an online database of audio file metadata for DJs at `http://www.mixshare.com`, which is an excellent start, but entries are not verified and the data is thus unsuitable for scientific work.

A Primary data set

100 songs of various dance music genres, chosen at random from the author's collection. The key classifications were initially made by the author and verified by two musical experts.

Table A.1: Primary data set and ground truth key classifications

Artist	**Title**	**Key**
Adam F	Circles (Album Edit)	Dm
Air	Sexy Boy	Dm
Arrested Development	People Everyday	Em
B-Complex	Beautiful Lies (VIP Mix)	Dm
Beenie Man	Who Am I? (Playground Mix)	Bm
Benga	Crunked Up	G♭m
Beyoncé	Crazy In Love	Dm
Bodyrox	Yeah Yeah (feat Luciana) (D Ramirez Vocal Club Mix)	C
Boys Noize	Yeah	Dm
Breakbeat Era	Animal Machine	B♭m
Britney Spears	Toxic (Armand Van Helden Remix)	Cm
Brookes Brothers	Tear You Down	E♭m
Calvin Harris	Acceptable In The 80s	E
Chase & Status	Can't Get Enough	D♭m
Chase & Status	Eastern Jam	Em
Chase & Status	Streetlife (feat Takura)	Dm
The Chemical Brothers	Dig Your Own Hole	E♭m
Commix	Be True	A♭m
Cyantific	Brighter Day (feat Natalie Williams)	G♭m
Daft Punk	Harder, Better, Faster, Stronger	G♭m
Dan Le Sac	Thou Shalt Always Kill (feat Scroobius Pip)	B♭m
De La Soul	All Good? (feat Chaka Khan)	Am
Dead Prez	Hip-Hop	Am
DJ DeeKline	Steam (feat Wizard and DJ Fresh)	B♭m
DJ Fresh	Golddust (feat Ce'cile)	G
DJ Hazard	Evac Q 8	A♭m
DJ Rap	Bad Girl (Roller Remix)	B♭m
DJ SS	The Lighter	Cm
DJ Zinc	Super Sharp Shooter	Gm
Dom Almond	Shake It (Philipe Boyar Remix)	Fm
Dom And Roland	Thunder (Remix)	G♭m

Artist	**Title**	**Key**
Double 99	Ripgroove (feat Top Cat) (Vocal Club Mix)	Fm
Duck Sauce	Barbra Streisand	E♭
edIT	Straight Heat	Fm
Faithless	Insomnia (Monster Mix)	Bm
Fugees	Ready Or Not (DJ Zinc Remix)	Am
Gabby Young and Other Animals	We're All In This Together (Ibrahim's D'n'B Mix)	D♭m
Ges-E	Streets Of Basra	Bm
Goose	Black Gloves	Em
Gorillaz	Clint Eastwood (Ed Case & Sweetie Irie Refix)	E♭m
High Contrast	Kiss Kiss Bang Bang	Gm
Josh Wink	Higher State Of Consciousness (Tweekin' Acid Funk)	G
Jurassic 5	Concrete Schoolyard	C
Justice vs Simian	Never Be Alone	Am
Katy B	Katy On A Mission	Cm
Kelis	Milkshake (Freq Nasty's Hip Hall Mix)	D♭m
Kloe & Imprintz	It's Time	Em
Krafty Kuts	Happiness (feat A-Skillz)	A♭
Ladytron	Destroy Everything You Touch	Em
Laurent Garnier	The Sound Of The Big Babou	B♭
LCD Soundsystem	Tribulations	Bm
Leftfield	Phat Planet	A♭m
Leftfield	Song Of Life	Cm
Leftfield	Space Shanty	G
London Elektricity	Just One Second (Apex Remix)	G♭m
Lowkey	I'm Back	Gm
Luniz	I Got 5 On It (Aphrodite's Original Dubplate)	B♭m
M-Beat	Sweet Love (feat Nazlyn)	B♭
Max Romeo & The Upsetters	Chase The Devil	Am
Mickey Finn & Aphrodite	Bad Ass	Gm
Missy Elliott	Get Ur Freak On	Fm
Moving Fusion	Turbulence	A♭m
Mr Oizo	Flat Beat	Bm
Mylo	Drop The Pressure	Em
Nero	Requiem	Gm
Netsky	Secret Agent	B♭m
Noisia	Shellshock (feat Foreign Beggars)	Fm
Noisia	Stigma	G
Northern Lite	Treat Me Better	A♭m
Omni Trio	Secret Life	Fm
Orbital	Doctor?	Em
Orbital	Nothing Left (feat Alison Goldfrapp) (Short Version)	A
Outkast	Hey Ya!	G
Peaches	I Feel Cream	E♭m
Pendulum	Tarantula (feat Fresh, Spyda and Tenor Fly)	Dm
Prodigy	Breathe	E♭m
Prodigy	Out Of Space	A♭m
Prodigy	Poison	E
Prodigy	Smack My Bitch Up	B♭m

Artist	**Title**	**Key**
Prodigy	Voodoo People (Pendulum Remix)	A♭m
Radiohead	Idioteque	E♭
Roni Size Reprazent	Brown Paper Bag	Em
Roots Manuva	Witness (1 Hope)	A♭m
Röyksopp	Eple	Fm
Röyksopp	The Girl And The Robot	Am
Saint Etienne	Only Love Can Break Your Heart	Dm
Shimon & Andy C	Bodyrock	Bm
Shy FX	On The Run (feat David Boomah)	B♭m
Shystie	Nu Style (DeeKline & Ed Solo Mix)	Em
SL2	On A Ragga Tip	A
Souvlaki	Inferno (Fired Up Mix)	Em
Spor	Aztec	B
Squarepusher	My Red Hot Car	G♭m
Sub Focus	Triple X	C
Substatic	Wild Horses (Demo)	Em
Tori Amos	Professional Widow (Armand's Star Trunk Funkin' Mix)	Dm
The Wiseguys	Ooh La La	D♭
Wu-Tang Clan	Gravel Pit	Am
Xample	Keep Their Heads Ringing	Fm
Yelle	Je Veux Te Voir	Am

B Beatles data set

179 songs from 12 albums by The Beatles. The key classifications are by Mauch et al. (2009); where a song was annotated as changing key, the global key chosen was that which lasted for the greatest proportion of the song. Note that the atonal *Revolution 9*, track 29 of *The Beatles*, is excluded.

Table B.1: Secondary data set and ground truth key classifications

Album	**#**	**Title**	**Key**
Please Please Me	1	I Saw Her Standing There	E
Please Please Me	2	Misery	C
Please Please Me	3	Anna (Go To Him)	D
Please Please Me	4	Chains	B♭
Please Please Me	5	Boys	E
Please Please Me	6	Ask Me Why	E
Please Please Me	7	Please Please Me	E
Please Please Me	8	Love Me Do	G
Please Please Me	9	P.S. I Love You	D
Please Please Me	10	Baby It's You	G
Please Please Me	11	Do You Want To Know A Secret	E
Please Please Me	12	A Taste Of Honey	G♭m
Please Please Me	13	There's A Place	E
Please Please Me	14	Twist And Shout	D
With The Beatles	1	It Won't Be Long	E
With The Beatles	2	All I've Got To Do	E
With The Beatles	3	All My Loving	E
With The Beatles	4	Don't Bother Me	Em
With The Beatles	5	Little Child	E
With The Beatles	6	Till There Was You	F
With The Beatles	7	Please Mister Postman	A
With The Beatles	8	Roll Over Beethoven	D
With The Beatles	9	Hold Me Tight	F
With The Beatles	10	You Really Got A Hold On Me	A
With The Beatles	11	I Wanna Be Your Man	E
With The Beatles	12	Devil In Her Heart	G
With The Beatles	13	Not A Second Time	G
With The Beatles	14	Money (That's What I Want)	E

Album	**#**	**Title**	**Key**
A Hard Day's Night	1	A Hard Day's Night	G
A Hard Day's Night	2	I Should Have Known Better	G
A Hard Day's Night	3	If I Fell	D
A Hard Day's Night	4	I'm Happy Just To Dance With You	E
A Hard Day's Night	5	And I Love Her	E
A Hard Day's Night	6	Tell Me Why	D
A Hard Day's Night	7	Can't Buy Me Love	C
A Hard Day's Night	8	Any Time At All	D
A Hard Day's Night	9	I'll Cry Instead	G
A Hard Day's Night	10	Things We Said Today	Am
A Hard Day's Night	11	When I Get Home	C
A Hard Day's Night	12	You Can't Do That	G
A Hard Day's Night	13	I'll Be Back	A
Beatles For Sale	1	No Reply	C
Beatles For Sale	2	I'm A Loser	G
Beatles For Sale	3	Baby's In Black	A
Beatles For Sale	4	Rock And Roll Music	A
Beatles For Sale	5	I'll Follow The Sun	C
Beatles For Sale	6	Mr. Moonlight	G♭
Beatles For Sale	7	Kansas City-Hey-Hey-Hey-Hey!	G
Beatles For Sale	8	Eight Days A Week	D
Beatles For Sale	9	Words Of Love	A
Beatles For Sale	10	Honey Don't	E
Beatles For Sale	11	Every Little Thing	A
Beatles For Sale	12	I Don't Want To Spoil The Party	G
Beatles For Sale	13	What You're Doing	D
Beatles For Sale	14	Everybody's Trying To Be My Baby	E
Help!	1	Help!	A
Help!	2	The Night Before	D
Help!	3	You've Got To Hide Your Love Away	G
Help!	4	I Need You	A
Help!	5	Another Girl	A
Help!	6	You're Gonna Lose That Girl	E
Help!	7	Ticket To Ride	A
Help!	8	Act Naturally	G
Help!	9	It's Only Love	C
Help!	10	You Like Me Too Much	G
Help!	11	Tell Me What You See	G
Help!	12	I've Just Seen a Face	A
Help!	13	Yesterday	F
Help!	14	Dizzy Miss Lizzy	A
Rubber Soul	1	Drive My Car	D
Rubber Soul	2	Norwegian Wood (This Bird Has Flown)	E
Rubber Soul	3	You Won't See Me	A
Rubber Soul	4	Nowhere Man	E
Rubber Soul	5	Think For Yourself	G
Rubber Soul	6	The Word	D
Rubber Soul	7	Michelle	Fm

Album	**#**	**Title**	**Key**
Rubber Soul	8	What Goes On	E
Rubber Soul	9	Girl	Cm
Rubber Soul	10	I'm Looking Through You	A♭
Rubber Soul	11	In My Life	A
Rubber Soul	12	Wait	G♭m
Rubber Soul	13	If I Needed Someone	A
Rubber Soul	14	Run For Your Life	D
Revolver	1	Taxman	D
Revolver	2	Eleanor Rigby	Em
Revolver	3	I'm Only Sleeping	E♭m
Revolver	4	Love You To	Cm
Revolver	5	Here, There And Everywhere	G
Revolver	6	Yellow Submarine	G
Revolver	7	She Said She Said	B♭
Revolver	8	Good Day Sunshine	A
Revolver	9	And Your Bird Can Sing	E
Revolver	10	For No One	B
Revolver	11	Doctor Robert	B
Revolver	12	I Want To Tell You	A
Revolver	13	Got To Get You Into My Life	G
Revolver	14	Tomorrow Never Knows	C
Sgt Pepper's Lonely Hearts Club Band	1	Sgt Pepper's Lonely Hearts Club Band	G
Sgt Pepper's Lonely Hearts Club Band	2	With A Little Help From My Friends	E
Sgt Pepper's Lonely Hearts Club Band	3	Lucy In The Sky With Diamonds	G
Sgt Pepper's Lonely Hearts Club Band	4	Getting Better	C
Sgt Pepper's Lonely Hearts Club Band	5	Fixing A Hole	F
Sgt Pepper's Lonely Hearts Club Band	6	She's Leaving Home	E
Sgt Pepper's Lonely Hearts Club Band	7	Being For The Benefit Of Mr Kite	Dm
Sgt Pepper's Lonely Hearts Club Band	8	Within You Without You	D♭
Sgt Pepper's Lonely Hearts Club Band	9	When I'm Sixty-Four	D♭
Sgt Pepper's Lonely Hearts Club Band	10	Lovely Rita	E
Sgt Pepper's Lonely Hearts Club Band	11	Good Morning Good Morning	A
Sgt Pepper's Lonely Hearts Club Band	12	Sgt Pepper's Lonely Hearts Club Band (Reprise)	F
Sgt Pepper's Lonely Hearts Club Band	13	A Day In The Life	G
Magical Mystery Tour	1	Magical Mystery Tour	D
Magical Mystery Tour	2	The Fool On The Hill	D
Magical Mystery Tour	3	Flying	C
Magical Mystery Tour	4	Blue Jay Way	C
Magical Mystery Tour	5	Your Mother Should Know	C
Magical Mystery Tour	6	I Am The Walrus	A
Magical Mystery Tour	7	Hello Goodbye	C
Magical Mystery Tour	8	Strawberry Fields Forever	B♭
Magical Mystery Tour	9	Penny Lane	B
Magical Mystery Tour	10	Baby You're A Rich Man	G
Magical Mystery Tour	11	All You Need Is Love	G
The Beatles	1	Back In The USSR	A
The Beatles	2	Dear Prudence	D

Album	**#**	**Title**	**Key**
The Beatles	3	Glass Onion	Am
The Beatles	4	Ob-La-Di, Ob-La-Da	B
The Beatles	5	Wild Honey Pie	G
The Beatles	6	The Continuing Story Of Bungalow Bill	C
The Beatles	7	While My Guitar Gently Weeps	Am
The Beatles	8	Happiness Is A Warm Gun	C
The Beatles	9	Martha My Dear	E♭
The Beatles	10	I'm So Tired	A
The Beatles	11	Blackbird	G
The Beatles	12	Piggies	A
The Beatles	13	Rocky Racoon	C
The Beatles	14	Don't Pass Me By	C
The Beatles	15	Why Don't We Do It In The Road	D
The Beatles	16	I Will	F
The Beatles	17	Julia	D
The Beatles	18	Birthday	A
The Beatles	19	Yer Blues	E
The Beatles	20	Mother Nature's Son	D
The Beatles	21	Everybody's Got Something To Hide Except Me And My Monkey	E
The Beatles	22	Sexy Sadie	G
The Beatles	23	Helter Skelter	E
The Beatles	24	Long, Long, Long	F
The Beatles	25	Revolution 1	B
The Beatles	26	Honey Pie	G
The Beatles	27	Savoy Truffle	G
The Beatles	28	Cry Baby Cry	G
The Beatles	30	Good Night	G
Abbey Road	1	Come Together	Dm
Abbey Road	2	Something	C
Abbey Road	3	Maxwell's Silver Hammer	D
Abbey Road	4	Oh Darling	A
Abbey Road	5	Octopus's Garden	E
Abbey Road	6	I Want You (She's So Heavy)	Dm
Abbey Road	7	Here Comes The Sun	Am
Abbey Road	8	Because	D♭m
Abbey Road	9	You Never Give Me Your Money	A
Abbey Road	10	Sun King	E
Abbey Road	11	Mean Mr Mustard	E
Abbey Road	12	Polythene Pam	E
Abbey Road	13	She Came In Through The Bathroom Window	A
Abbey Road	14	Golden Slumbers	Am
Abbey Road	15	Carry That Weight	C
Abbey Road	16	The End	A
Abbey Road	17	Her Majesty	D
Let It Be	1	Two Of Us	G
Let It Be	2	Dig A Pony	A

Album	**#**	**Title**	**Key**
Let It Be	3	Across The Universe	D
Let It Be	4	I Me Mine	Am
Let It Be	5	Dig It	F
Let It Be	6	Let It Be	C
Let It Be	7	Maggie Mae	G
Let It Be	8	I've Got A Feeling	A
Let It Be	9	One After 909	B
Let It Be	10	The Long And Winding Road	E♭
Let It Be	11	For You Blue	D
Let It Be	12	Get Back	A

C Experiment parameter listings

This appendix lists the parameters for all the experiments detailed in section 4. Left-aligned parameters are those which are required for all experiments. Any indented parameters are dependent on the left-aligned parameter above.

Table C.1: Parameters of frequency analysis range experiment, figure 4.1

Parameter	**Value**
Frequency analysis range	**VARIANT**
Bins per semitone	1
Downsample factor	10
Hop size	2^{12}
Spectral analyser	Direct spectral kernel transform
Frame size	2^{14}
Q stretch	0.8
Temporal window	Blackman
Segmentation	None
Tone profiles	Sha'ath
Similarity measure	Cosine

Table C.2: Parameters of FFT resolution experiment, figure 4.2

Parameter	**Value**
Frequency analysis range	C1 - B6
Bins per semitone	1
Downsample factor	10
Hop size	**VARIANT**
Spectral analyser	Direct spectral kernel transform
Frame size	**VARIANT**
Q stretch	0.8
Temporal window	Blackman
Segmentation	None
Tone profiles	Sha'ath
Similarity measure	Cosine

Table C.3: Parameters of CQT and DSK comparison, figure 4.3

Parameter	**Value**
Frequency analysis range	C1 - B6
Bins per semitone	1
Downsample factor	10
Hop size	2^{12}
Spectral analyser	**VARIANT**
Frame size	2^{14}
Q stretch	0.8
Temporal window	Blackman
Segmentation	None
Tone profiles	Sha'ath
Similarity measure	Cosine

Table C.4: Parameters of DSK bandwidth experiment, figures 4.4 and 4.5

Parameter	**Value**
Frequency analysis range	C1 - B6
Bins per semitone	1
Downsample factor	10
Hop size	2^{12}
Spectral analyser	Direct spectral kernel transform
Frame size	2^{14}
Q stretch	**VARIANT**
Temporal window	Blackman
Segmentation	None
Tone profiles	Sha'ath
Similarity measure	Cosine

Table C.5: Parameters of Harte tuning algorithm experiment, figure 4.6

Parameter	**Value**
Frequency analysis range	C1 - B6
Bins per semitone	**VARIANT**
Tuning algorithm	Harte
Downsample factor	10
Hop size	2^{12}
Spectral analyser	Direct spectral kernel transform
Frame size	2^{14}
Q stretch	0.8
Temporal window	Blackman
Segmentation	None
Tone profiles	Sha'ath
Similarity measure	Cosine

Table C.6: Parameters of bin-adaptive tuning algorithm experiment, figure 4.7

Parameter	**Value**
Frequency analysis range	C1 - B6
Bins per semitone	**VARIANT**
Tuning algorithm	Bin-adaptive
Downsample factor	10
Hop size	2^{12}
Spectral analyser	Direct spectral kernel transform
Frame size	2^{14}
Q stretch	0.8
Temporal window	Blackman
Segmentation	None
Tone profiles	Sha'ath
Similarity measure	Cosine

Table C.7: Parameters of tone profile and similarity measure experiment, figure 4.9

Parameter	**Value**
Frequency analysis range	C1 - B6
Bins per semitone	1
Downsample factor	10
Hop size	2^{12}
Spectral analyser	Direct spectral kernel transform
Frame size	2^{14}
Q stretch	0.8
Temporal window	Blackman
Segmentation	None
Tone profiles	**VARIANT**
Similarity measure	**VARIANT**

Table C.8: KeyFinder default parameters, used for comparison to other software, figure 4.10

Parameter	**Value**
Frequency analysis range	C1 - B6
Bins per semitone	1
Downsample factor	10
Hop size	2^{12}
Spectral analyser	Direct spectral kernel transform
Frame size	2^{14}
Q stretch	0.8
Temporal window	Blackman
Segmentation	None
Tone profiles	Sha'ath
Similarity measure	Cosine

References

Lawrence Abbott. *Approach To Music*. George G. Harrap & Co, London, 1942.

Jonathan Berger. Website of Stanford University's Center for Computer Research in Music and Acoustics - Creative Arts: New Tools and Technology and the Democratization of Craft, March 2009. URL `https://ccrma.stanford.edu/~brg/`.

Benjamin Blankertz. Website of Westfälische Wilhelms-Universität Münster's Mathematics and Computer Science Department - The Constant Q Transform, 2001. URL `http://wwwmath.uni-muenster.de/logik/Personen/blankertz/constQ/constQ.html`.

Bill Brewster and Frank Broughton. *Last Night A DJ Saved My Life: The History Of The Disc Jockey, Second Edition*. Headline, London, 2006.

Judith C. Brown. Calculation of a Constant Q Spectral Transform. *Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

Judith C. Brown and Miller S. Puckette. An Efficient Algorithm for the Calculation of a Constant Q Transform. *Journal of the Acoustical Society of America*, 92(5):2698–2701, 1992.

Camelot Sound. Website of Camelot Sound - Easymix Harmonic Key Selection. URL `http://www.camelotsound.com/Easymix.aspx`.

Elaine Chew. The Spiral Array: An Algorithm For Determining Key Boundaries. *Proceedings of the Second International Conference, ICMAI 2002*, pages 18–31, 2002.

Erik de Castro Lopo. Website of the *Secret Rabbit Code* project, February 2009. URL `http://www.mega-nerd.com/SRC/`.

Erik de Castro Lopo. Website of the *LibSndFile* project, October 2010. URL `http://www.mega-nerd.com/libsndfile/`.

Tony Fisher. Website of the University of York's Computer Science Department - Interactive Digital Filter Design, September 1999. URL `http://www-users.cs.york.ac.uk/~fisher/mkfilter/`.

Eric Freeman, Elisabeth Freeman, Kathy Sierra, and Bert Bates. *Head First Design Patterns*. O'Reilly, Sebastopol CA, USA, 2004.

Matteo Frigo and Steven G. Johnson. Website of the *Fastest Fourier Transform in the West* project, July 2009. URL `http://www.fftw.org/`.

Emilia Gómez and Perfecto Herrera. Estimating the tonality of polyphonic audio files: Cognitive versus machine learning modelling strategies. In *Proceedings of the 5th International Conference on Music Information Retrieval, Barcelona, Spain*, 2004.

Emilia Gómez Gutiérrez. *Tonal Description of Music Audio Signals*. PhD thesis, Universitat Pompeu Fabra, 2006.

Christopher Harte. *Towards Automatic Extraction of Harmony Information from Music Signals*. PhD thesis, Queen Mary University of London, 2010.

ISMIR. Website of the *MIREX* campaign, 2005. URL `http://www.music-ir.org/evaluation/mirex-results/audio-key/index.html`.

Özgür İzmirli. Template Based Key Finding From Audio. In *Proceedings of the International Computer Music Conference, ICMC'05, Barcelona, Spain*, 2005.

Otto Károlyi. *Introducing Music*. Penguin, 1973.

Carol L. Krumhansl. *Cognitive Foundations of Musical Pitch*. Oxford University Press, New York, 1990.

LibAV community. Website of the *libav* project, June 2011. URL `http://libav.org/`.

William Lovelock. *The Rudiments of Music*. G Bell, 1957.

Richard G. Lyons. *Understanding Digital Signal Processing, Third Edition*. Prentice Hall PTR, 2010.

Matthias Mauch, Chris Cannam, Matthew Davies, Simon Dixon, Christopher Harte, Sefki Kolozali, and Dan Tidhar. OMRAS2 Metadata Project 2009. In *10th International Conference on Music Information Retrieval Late-Breaking Session, Kobe, Japan*, 2009.

Nokia et al. Website of the Nokia corporation - Qt framework, June 2011. URL `http://qt.nokia.com/products/`.

Katy Noland. *Computational Tonality Estimation: Signal Processing and Hidden Markov Models*. PhD thesis, Queen Mary University of London, 2009.

Harry F. Olson. *Modern Sound Reproduction*. Van Nostrand Reinhold, 1972.

Steffen Pauws. Musical key extraction from audio. In *Proceedings of the 5th International Conference on Music Information Retrieval, Barcelona, Spain*, pages 96–99, 2004.

Geoffroy Peeters. Musical Key Estimation of Audio Signal based on Hidden Markov Modeling of Chroma Vectors. In *Proceedings of the 9th International Conference on Digital Audio Effects, DAFx-06, Montreal, Canada*, 2006.

Ulf Poschardt. *DJ Culture*. Quartet, 1998.

Stanley Sadie, editor. *The New GROVE Dictionary of Music and Musicians*. Macmillan, London, 1980.

David Temperley. What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered. *Music Perception: An Interdisciplinary Journal*, 17(1):65–100, 1999.

Scott Wheeler. Website of the *TagLib* project, March 2011. URL `http://developer.kde.org/~wheeler/taglib.html`.