University of Dublin

TRINITY COLLEGE

An Investigation into the Krumhansl-Schmuckler Musical Key-Finding Algorithm: Performance, Failure Cases and Potential Improvements

Daniel Browne

B.A. (Mod.) Computer Science

Final Year Project April 2020
Supervisor: Fergal Shevlin

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration:

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

X Daniel Browne

Daniel Browne

Abstract:

The Krumhansl-Schmuckler Key-Finding algorithm is a widely used method of computationally estimating the musical key in which a given piece of music is composed, requiring no more than an audio file containing the chosen piece of music. Accurate estimates are particularly useful to musicians and music academics who aim to gain a further understanding of a specific song. However, this algorithm is not accurate in all cases, and can oftentimes produce an estimate that does not correctly match the key in which the piece of music is truly composed.

This report discusses the process of investigating the performance of this algorithm over a dataset comprised of modern contemporary rock and metal music, as well as the process of experimenting with new implementations of the algorithm in order to attempt to attain more consistently accurate results. In addition, this report will detail a series of musical features frequently found in the genre of music from which this dataset was created that can potentially lead to inaccurate algorithmic estimate generation, as well as the experiments performed to test the validity of the suggestion that these features could be detrimental to the likelihood of the algorithm producing an accurate estimate for a given piece of music.

Acknowledgements:

Firstly, I would like to thank my supervisor, Fergal Shevlin, for allowing me to perform this research and for taking the time to provide vital guidance, feedback, and support throughout the project. In addition, I would like to thank my second reader, Simon Wilson, for taking the time to contribute his expertise to the grading of this project.

I would like to thank Adam Hannigan, John Clay, and Jordan Hannigan for sharing music theory information and pointing me in the direction of useful sources.

I would like to thank Mark Boyle for lending both his time and musical equipment in order to assist me in performing an experiment during this project.

I would like to thank my family, whom I would not have been able to complete this project without.

Finally, I would like to thank Emma Boyle for providing me with endless support, motivation and encouragement for the past four years, and for giving important feedback throughout this entire project, as well as key information and clarification regarding specific areas of music theory and their applications to this project.

Table of Contents:

List of Figures:

List of Tables:

Terminology:

Note: A musical concept labelling a given frequency and all frequencies that can be obtained by halfing or doubling that frequency any number of times, used in music theory for simpler understanding and description.[1]

Melody: The concept of sequences of notes that are musically and/or artistically satisfying to a composer and/or listener. [2]

Harmony: The concept of playing multiple notes simultaneously – generally notes whose frequencies form a 'clean' ratio between one another, for example, 1.5:1, rather than a continuous ratio such as 1.0594…:1. [3]

Interval: The gap between a note and another. [4]

Semitone: The interval between one note and the next note higher or lower than it. [5]

Octave: A set of all notes between a note and at the next occurrence of the same note. Twelve semitones in length. [6]

Whole Step: The movement from a note to the note two semitones higher or lower than it. [7]

Half Step: To move from a note to the note one semitone higher or lower than it. [8]

Major: A scale or key that follows the major interval pattern of steps: Whole, Whole, Half, Whole, Whole, Whole, Half. [9]

Minor: A scale or key that follows the minor interval pattern of steps: Whole, Half, Whole, Whole, Half, Whole, Whole. [9]

Scale: A set of notes, generally containing seven unique notes, that normally follow a major or minor interval pattern to traverse one octave. [9]

Key: A set of notes comprising a chosen scale used as a foundation from which to create musical melodies and compositions. [10]

Accidental: A note played in a piece of music that does not belong to the key in which the piece is written.[11]

Relative Minor: A minor key that contains the exact same notes as a major key. The two keys can be described as Parallel Keys. [12]

Tonic Note: The first note in a scale or key. [12]

Tonal Center: A musical reference point where melodies frequently begin and end – a point which the musical composition tends to gravitate towards. Normally the tonic note of the key in which the song is written. [12]

API: Application Programming Interface – an interface within a system that allows other programs and software to interact with it. [13]

.mp3: MPEG Audio Layer-3. A file type used for storing audio files. [14]

.wav: Waveform Audio File Format. An additional file type used for storing audio files. [15]

.csv: Comma-Separated Value. A file type used for storing lists of information. [16]

1. Chapter One - Introduction:

1.1 Project Motivation

Key is a fundamental foundation of musical composition that helps guide musicians and composers to finding the right notes to use and play in their music. Key is important for various reasons - composers use it in creating new music, as the chosen key can have an impact on the atmospheric tone of the music, effecting the meaning or atmosphere a listener may take from it; Songs in major keys are more likely to be described as happy or uplifting compared to songs in minor keys, for example. For composers, key can be considered a musical building block, using which new works can be created.

Key is important for musicians as it gives them a reference to play from - if they know what key a piece of music should be in, they can infer from that notes that they should avoid or gravitate towards in their playing, as notes that do not belong to a key can sound massively out of place and displeasing to a listener in many occasions. When musicians are composing solos and improvising over a piece of music, knowledge of what key the music is in is vital in ensuring that they do not begin to play out of key with the rest of the musicians that they are playing with, creating a displeasing sound.

Key is also useful to radio hosts, DJ's and live performers that aim to play a series of songs that fit well with one another in sequence, and in the creation of 'mash-ups' – songs that combine different elements from multiple other songs to create a new piece of music. Determining the key of the songs involved is an important part of this process and helps ensure that the end product will be musically satisfying to the listener.

Due to its musical significance, and with the ever-increasing incorporation of technology into music and the music industry, I believe that the presence of an accurate, reliable algorithm that can determine the key in which a piece of music is composed would be highly valuable across the areas of music theory and research, as well as for musicians and performers. The Krumhansl-Schmuckler Key Finding Algorithm has potential to be improved beyond its current capability, further increasing its value to the world of music.

1.2 Project Objective

The central goals of this project are as follows:

1.2.1 Performance & Failure Case Analysis

This is the process of analysing the performance of the algorithm over a dataset of songs to gain an understanding of cases in which the algorithm does not produce an accurate estimate of the key of the song. Through this process, commonalities among songs that produce failure cases could be discovered, leading to an improved knowledge of features in music that can lead to the creation of incorrect estimates by the algorithm.

The information gained from this process can then influence the procedures and the steps taken in new implementations of the algorithm in order to avoid the pitfalls that these features may create, and this will ideally lead to an increase in the accuracy of the algorithm and improve the reliability of the algorithm overall.

1.2.2 New Implementation Investigation and Experimentation

This is the process of creating, testing, and evaluating the performance of new approaches towards implementing the algorithm, and also creating new methods of processing the music in the dataset itself before it is inputted into the algorithm in order to examine the impact these actions have on the resulting estimates that the algorithm produces.

In addition, this includes the process of creating test tracks and experimenting with the application of various effects and changes to the tracks in order to create a new track that can be inputted to

the algorithm in order to attain a series of results that can be compared to one another in order to identify the impact, if any, that each specific feature has over the reliability and the accuracy of results from the algorithm.

1.3 Technologies Used

1.3.1 Python

Python is an open-source, general purpose, high-level programming language that is commonly used in scientific research. As a result, there exists many libraries and resources to assist with the operation of many of the tasks performed throughout this project.[17]

1.3.2 Essentia

Essentia is an open-source Python library that provides many different functionalities surrounding the analysis and description of audio. This includes a function that performs the Krumhansl-Schmuckler Key-Finding Algorithm. This is the function that was used to create estimates and to implement new approaches surrounding the usage of the algorithm. As well as this, it contains a function that can extract the beats-per-minute of an audio file, which was useful for a number of new algorithm implementations that were tested throughout the project. [18]

1.3.3 NumPy & SciPy

NumPy, also known as Numerical Python, and SciPy, also known as Scientific Python, are open-source Python libraries that provide functionality surrounding the creation, management, and operation upon large arrays of numbers, such as ranking and sorting data in an array, selecting specific values from an array, and performing calculations upon every value in an array simultaneously. In addition, SciPy was useful for performing the Fourier Transform on a .wav audio file for analysis. [19]

1.3.4 Pandas

Pandas is an open-source Python library used to manage and manipulate large datasets, and was useful in opening, reading, writing to and saving datasets and sets of results throughout the project. [20]

1.3.5 Youtube_dl

Youtube_dl is an open-source Python library that simplifies the process of interacting with the YouTube API, allowing for the extraction of audio from YouTube videos to be saved as .mp3 files. [21]

1.3.6 Matplotlib

Matplotlib is an open-source Python library that provides functionality for programmatically creating, customising and editing plots, charts, graphs and other data visualisation from data structures within Python. This was used for the visualisation of results for comparison and analysis, and for the visualisation of audio files for investigation and analysis of individual songs. [22]

1.3.7 FFMPEG

FFMPEG is an open source project including a Linux command-line function which provides functionality to work with various multimedia file types. For the purposes of this project, it was used to convert audio files into identical formats, including having the same sample rate, bit rate and codec. [23]

1.3.8 Natsort

Natsort is a simple, open-source Python library that provides functionality for sorting arrays based on labels used in natural language – i.e., "1st, 2nd, 3rd…". This was used for the organisation of labelled sets of results for clearer and more ordered visualisation of results. [24]

1.3.9 PIL

The Python Imaging Library is an open-source Python library that allows for the programmatic creation and editing of images through Python code. This was used in the creation of certain images detailing the performance of implementations of the algorithm over certain metrics in a way that Matplotlib was unable to. [25]

1.3.10 Audacity

Audacity is an open-source program that allows for the creation, editing, manipulation and saving of audio files. It contains effects and operations that can be performed on existing tracks and can also generate sounds for the creation of new tracks. It was used in the testing of many different methods of pre-processing audio in preparation for passing into the algorithm. [26]

2. Chapter Two - State of the Art:

This section will explain in detail the Krumhansl-Schmuckler algorithm in its current form.

The Krumhansl-Schmuckler Key-Finding Algorithm is the product of work and experimentation performed by music psychologists Carol L. Krumhansl and Mark Schmuckler of Cornell University and the University of Toronto Scarborough respectively. Their relevant work is outlined in 'Cognitive Foundations of Musical Pitch' (2001) by Carol Krumhansl.[27] This work included a series of experiments on test subjects, in which they were given a 'musical context'. That is, one or more notes chosen from a certain key were played for them. Once this musical context was established, they were played a 'probe tone' – another note that may, or may not, be a part of the chosen key. Following this, the test subjects were asked to give a rating to indicate how well they felt the probe tone fit the musical context.

From these ratings, generalised weightings were able to be constructed to build a 'tonal hierarchy' for major and minor keys respectively. The tonal hierarchies represent the relative importance and fit of each possible tone to the key, which is different in major and minor keys.

The tonal hierarchy for major keys is shown in Table 2.1, while the tonal hierarchy for minor keys is shown in Table 2.2:

Major Keys:

Do	Do♯	Re	Re♯	Mi	Fa	Fa♯	So	So♯	La	La♯	Ti
6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88

Table 2.1 – Major Key Tonal Hierarchy

Minor Keys:

La	La♯	Ti	Do	Do♯	Re	Re♯	Mi	Fa	Fa♯	So	So♯
6.33	2.68	3.52	5.58	2.60	3.53	2.54	4.75	3.98	2.69	3.34	3.17

Table 2.2 – Minor Key Tonal Hierarchy

The "Do-Re-Mi…" structure is used to generalise the format of the major and minor scale, with each word representing a note. The sharpened tones, represented with the sharp symbol (♯) after the name for the note, indicate the notes that are not part of the key. As the weights relevant to each of the tones can indicate, notes outside the key do not fit the tonal hierarchy of the key as well as those that are part of the key.

It is also worth noting that the minor key begins on the 'La' tone, while the major key begins on 'Do'. This is due to the fact that a minor key will share the exact same notes as the major key of the note three semitones above it – 'Do'. Despite this, the tonal hierarchy is still different when the per-tone values are compared between major and minor keys. This is because the tonal centres of the keys are different. A major key will tend to have its tonal centre at 'Do' – its tonic note. Similarly, a minor key will also have its tonal centre at its tonic note, however, this note is not 'Do', instead, it is 'La'. This difference in tonal centre creates a distinction between the two keys, which is why the values are not identical.

These generalised weightings can then be applied to the context of the notes of each key to generate what is called a Key Profile for each key. For example, if the key profile for C Major were to be generated, the value for 'Do' would be replaced with C, 'Do♯' would become C♯, 'Re' would become D, and so on until all possible notes have been mapped to their corresponding value.

With key profiles generated for every possible key, the algorithm can generate estimates for the key that an audio file is in. This is done by enumerating the occurrences of every individual frequency in the audio file, then mapping the counted frequencies to their corresponding note – if they correspond to one. With this complete, the relative prevalence of each note can be calculated in order to generate what is known as a Pitch Class Profile. This has a similar structure to the key profiles created earlier. Once the pitch class

profile for the track is generated, it can then be compared to the key profile of each possible key in order to discover which key profile it correlates to best. This correlation is calculated using the following formula:

$$R = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where $\bar{x}$ is the mean of the x coordinates and $\bar{y}$ is the mean of the y coordinates, n is the number of sets of coordinates, and R is the correlation coefficient.

Figure 2.1 – Correlation Coefficient Formula

In the context of this algorithm, the x data is comprised of the list of values assigned to each note in the key profile of a particular key. Meanwhile, the y data is a list of the values in the frequency profile – the accumulated duration of each individual note in the audio file. The index of the duration of any particular note in the y data matches the index of the designated value for that note from the chosen key profile in the x data. This creates (x, y) pairs, grouping the weight of a note with the duration of the same note.

This correlation coefficient will return a value between -1 and 1, with 1 indicating a perfect correlation. With a correlation coefficient calculated for each key, the estimate is then chosen by finding the key profile that returns the largest correlation coefficient.

For example, using a sample frequency profile in Table 2.3 that could be generated for an audio file of a song, the correlation coefficient between the sample frequency profile and the C major key profile can be calculated.

C	C♯	D	D♯	E	F	F♯	G	G♯	A	A♯	B
550	0	275	0	323	355	0	433	0	311	0	150

Table 2.3 – Sample frequency profile of composition in C major

The (x, y) pairs for this calculation are shown in Table 2.4.

	C	C♯	D	D♯	E	F	F♯	G	G♯	A	A♯	B
x	6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88
y	550	0	275	0	323	355	0	433	0	311	0	150

Table 2.4 – Sample of coordinates used in correlation coefficient calculation between the sample frequency profile and the C major key profile

Using the formula in Figure 2.1, the correlation coefficient for these pairs of data can be calculated. In this example, the correlation coefficient will be 0.9639.

To estimate the key, this process will be carried out for each key profile, meaning the data in the frequency profile will be paired a total of twenty-four times – once for every key, in major and minor. When calculating the correlation coefficient between the sample frequency profile and the C♯ major key profile, the coordinates, i.e., the data pairings, that are used are shown in Table 2.5.

	C♯	D	D♯	E	F	F♯	G	G♯	A	A♯	B	C
x	6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88
y	0	275	0	323	355	0	433	0	311	0	150	550

Table 2.5 – Sample of coordinates used in correlation coefficient calculation between the sample frequency profile and the C♯ major key profile

The correlation coefficient calculated for these data pairs is -0.576.

This process repeats for each key profile, and the key profile that generates the correlation coefficient closest to 1 will be returned as the key estimate. In this example, this should be C major, as the sample frequency profile has no notes outside of this key, while having frequency durations that are favourable to the tonal hierarchy of the notes within the key. This results in the correlation coefficient calculated being extremely close to 1, which indicates a strong correlation.

3. Chapter Three - Investigation

This section outlines the process of performing the analysis, experimentation and investigations outlined in the project goals section.

3.1 Dataset Establishment

3.1.1 Data Requirements Evaluation

This is the process of compiling the requirements for the type of data that will comprise the dataset upon which each implementation of the algorithm will be evaluated. The required data is a list of audio files of songs, categorised by the key that they are composed in. What must be determined is what type of songs will be compiled into the dataset, as well as the size of the dataset.

From my investigation into research already performed on the Krumhansl-Schmuckler Key-Finding Algorithm, as well as similar projects aiming to achieve the same goal, a majority of evaluation has been done using music from the classical genre, as well as music from the EDM (Electronic Dance Music) genre. These genres tend to be used for analysis as datasets that are categorised by key can be constructed and accessed more easily for these genres than most others. With regards to classical music, this is because classical music tends to have easily accessible sheet music, which indicates the key of the song using the key signature. This means that classical songs can be categorised by key by simply verifying the key signature visible on the sheet music. As for EDM songs, there exist public datasets labelled by key, such as the GiantSteps Key data set.[28] However, these are user-annotated, and can still potentially contain mislabelled data which can mislead conclusions and findings that come from experimentation.

With regards to other genres of music, The McGill Billboard Dataset contains information on over 700 songs that have featured in the Billboard music charts between 1958 and 1991, however the data contained for each song does not specify if the song is in a major or minor key, and it simply states the tonic note of its key.[29] The distinction between major and minor keys is an extremely important difference that must be made clear for the purposes of this investigation. For example, C major and C minor both share an identical tonic note, however, including this tonic note, the keys only share four notes, indicating that there is a significant difference between these keys.

In addition to unclear key categorisation in this dataset, the Billboard music charts will generally be comprised of the most popular music at a given time. As the dataset is comprised of songs that were placed on this chart over a course of almost forty years, there is a wide variance of genres and types of music that would be contained in this dataset – from rock and roll music to pop, hip-hop, electronic, country and folk music, and likely many more genres in addition to this. This extremely broad scope leads to a relatively small representation of each genre, which makes the process of finding common features within genres that can negatively influence the ability of the algorithm to produce accurate estimates much more difficult.

For the purposes of this project, a more specific, more reliable dataset would be ideal. This created a task of manually categorising a number of songs from a particular genre by key. This was done by ear, which is the process of playing along with the piece of music to determine the notes that comprise the melodies that are heard in the music. As well as this, sheet music was sourced where possible to serve as a second reference to confirm that the key signature I selected for the song in my categorisation by ear was correct. This form of categorisation was done despite the fact that a number of resources exist that aim to provide information regarding the key in which a song is composed. For example, TuneBat.com is a service powered by the Spotify API that allows users to find information and details about songs within their database.[30] One of the information fields displayed for each song by this service is the key. However, as this is powered by the Spotify API, this information is extracted from the Spotify database. Spotify generate this information computationally for each song inserted into the database, indicated by how the Spotify API reference specifies that the key data is “The estimated overall key of the track.”, which suggests that this is a generated estimate, and not a manually maintained or verified field.

Similarly, SongKeyFinder.com is a service that aims to provide information regarding the key of songs to users, without the use of algorithmically generated key estimates for songs.[31] Users can search for specific songs to find their key, and users can also search for specific keys in order to find songs written in that key. However, the categorisation method for the songs in the database of this platform is instead reliant on a voting system – users can go to the page of a specific song and vote on what key they believe the song is in, and the song will be categorised into the key for which it has received the most votes. This reduces the reliability of the categorisation, as songs with few total votes can have their key misrepresented by a small number of incorrect votes, intentional or otherwise. For this reason, it would simply be irresponsible and naïve to entirely trust the categorisation of either of these two platforms or any similar platforms in the construction of a dataset for this project. The same is true for the pre-existing datasets that have been labelled by users discussed earlier. The most reliable and trustworthy method of categorisation by key is manually, by ear and via sheet music, which is the method that I decided to proceed with in this project for the purposes of accuracy and reliability of analysis.

Due to the fact that the size of the dataset is limited by the amount that can be manually categorised, I decided that the most reasonable approach to this would be to limit the scope of the dataset to a single genre of music, as this would increase the likelihood of similarities between failure cases to be discovered, and it would allow for a more in-depth investigation into the features that are commonly found in the chosen genre, and the impact that these features have on the estimate of key, if any. This proceeded to raise the question of which genre to choose in order to build the dataset.

As mentioned before, many previous investigations into the performance of this algorithm used classical and electronic dance music in the experiments carried out on the algorithm. This means that, for the purposes of investigating new areas of performance, and to analyse unique and different failure cases, the ideal dataset would be comprised music from outside these genres. A paper entitled “Mozart to Metallica: A Comparison of Musical Sequences and Similarities” by S. Cunningham, V. Grout and H. Bergen from the University of Wales outlines an examination of music samples from genres that are generally considered vastly different in order to discover if any similarities can be drawn between these genres by looking at the music on a strictly notational basis – that is to say that it ignores timbre and artistic atmosphere, and instead simply investigates the notes used to create the composition in order to seek out similarity between genres. This study concluded that there exists similarity between classical music and modern contemporary rock/metal music.[32] This appears to be the case despite noticeable differences in the atmospheric tone and instrumentation of the music that typifies these genres. This finding led me to select modern contemporary rock and metal music to populate the dataset to be used in testing and analysis for this project.

3.1.2 Data Collection & Categorisation

The next portion of this section of the project involved finding, categorising and organising music to be added to the dataset. This was a process of selecting a song, determining the key of the song and adding it to a key-specific table by entering the title of the song as well as a link to the song for later retrieval. For the purposes of this project, I decided that the criteria for a correct key estimate was to have an exact match of the notes that make up the true key of the song. This means that major keys and their relative minor would share a table. This prevents estimates from being considered incorrect when the set of notes comprising the key is correctly estimated, but the tonal centre is not. Additionally, this prevents estimates being considered incorrect in the case that the tonic note is labelled as a valid and possible name for the note but is different to the label that is applied in categorisation. For example, D♯ and E♭ are exactly the same note, however the name that is chosen is contextual based on the movement and structure of the relevant piece of music. I believe that it would be excessively overscrupulous to reject a key estimate that correctly matches both the set of notes comprising the key, as well as the tonic note of the key, based entirely on the name that was used to represent the tonic note.

This criteria for consideration of a correct estimates means that there will be a total of twelve sets in which a song can be categorised – one for each unique set of notes that a key can contain. Each set of songs will contain at least twelve songs overall, which will result in a total of at least 144 full tracks to be used in the evaluation of performance. I would consider this a reasonable sample size to use in this investigation, as in the testing of the Krumhansl-Schmuckler algorithm, the performance was examined over small excerpts from classical music such as Bach's 'Well-Tempered Clavier' and Chopin's preludes, some of which were only a few notes long, as the test would begin by inputting the first note of the song into the algorithm, followed by the first two notes, and so on, until the algorithm produced a correct estimate, at which point the test would finish. Using full tracks to investigate performance will examine the performance in a real-world scenario that examines how the algorithm performs when the key of the song is completely unknown, as the key will only be compared to the estimate once that estimate is generated using the whole song. This method, that provides the algorithm with as much information as possible at once, makes for a fascinating investigation, as this could result in more strongly-correlating frequency profiles, because there can be more data accessible to the algorithm, however this also introduces the possibility of more features, effects and non-musical sounds being inputted into the algorithm that could potentially increase the difficulty of accurate key estimation for the algorithm.

Throughout the process of song selection, the primary concept that I strived to satisfy was to have representation of the many different types of forms in which the genre can come, and to give consideration to the many areas in which songs from this genre can vary, such as in tempo and instrumentation, as well as the cultural influences that can have an impact on the sound and composition of music from an artist or group. This led me to seek to ensure that non-English-language songs were included in the dataset, as well as songs at various paces, songs from varying sizes of group, and songs in which lead singers had unique vocal quality, be that in range, timbre, or technique – for example, female lead singers tend to have a higher vocal range than male lead singers, and, particularly in rock and metal music, the use of the 'vocal fry' technique in which a singer uses their lowest register to produce a deep, rumbling tone to their singing is frequently present. These factors are taken into consideration when choosing the songs to add to the dataset. In addition, in order to preserve variety in the music comprising the set of each key, I decided to limit the maximum number of songs from a single artist that could be added to the set of a single key to three. This prevents any single artist from comprising a majority of a set of songs in a particular key, avoiding the potential for this to create misleading results in the case that music from a particular artist could be considerably simpler or more complex to create accurate key estimates for.

It was of the utmost importance that the categorisation of songs was accurate, as an unreliable dataset would heavily undermine the accuracy of the results and the validity of the conclusions that can be taken from the entire project. This is why I went to such great lengths to ensure that the songs that comprised the dataset were songs that I had complete confidence in my categorisation for. In the case of certain songs that I could not confidently choose a key signature for, that had no available sheet music to help guide the categorisation, I would instead add the song to a 'Bonus Tracks' section, in which tracks that could not be confidently categorised into any key signature would be added in order to determine what estimates would be made for these tracks by the algorithm. Estimates in this section would not be counted for any metrics or performance analysis – this section purely exists for the purposes of testing how the algorithm performs in extreme edge cases, where songs may be composed using uncommon methods such as by using more obscure musical building blocks such as blues scales, pentatonic scales and diminished scales. These more advanced scales do not always match the sets of unique notes that comprise any of the possible keys, however composers still have the freedom and control to create songs using these scales as an alternative musical building block to key. That being said, they are not a close enough representation of any key to merit having any song written using these building blocks categorised as being written in any particular key. Conversely, there exists some songs that cannot be definitively categorised into a key due to overwhelming simplicity of the song meaning it can potentially fit multiple keys, as is the case with 'Sweet Home Alabama' by the group Lynyrd Skynyrd – the small set of notes used in the song mean that it could be described as in either D major or G major, because all of the notes and chords used in the song fit both of these keys. In

addition, the chords are best described as being in the key of G major, however the melody is best described as being in D major, leading to a phenomenon known as ‘dual tonicity’.[33] For this reason it was added to the bonus track library rather than that of either key.

Songs were chosen both from a personal knowledge of artists and groups that create music in this genre, as well as research into specific types of music that was desired for the dataset, such as groups from outside of primarily English-speaking countries. In addition, as the music was being sourced from YouTube, the YouTube recommendation system began to generate suggestions of bands and artists that I may not have become aware of otherwise. This helped ensure a wide range of artists and subgenres from within the genre of rock and metal music were covered in the dataset. Data was originally built in a table before being stored in a comma-separated values (.csv) format. An example of one of these tables is shown in Table 3.1:

Track	**Link**
Green Day - Jesus of Suburbia	https://youtu.be/JMcNzjzw63I
Guns and Roses - Sweet Child o’ Mine	https://youtu.be/oMfMUfgjiLg
Guns and Roses - Welcome to the Jungle	https://youtu.be/o1tj2zJ2Wvg
Alestorm – Keelhauled	https://youtu.be/ta-Z_psXODw
Evanescence – Lithium	https://youtu.be/8zotMMbK0bE
Breaking Benjamin - The Diary of Jane	https://youtu.be/pcAKbKUBUOQ
Biffy Clyro - Victory Over the Sun	https://youtu.be/hmg0hNFFdi8
Seether – Truth	https://youtu.be/W7UHTjryCtg
Smashing Pumpkins - Bullet With Butterfly Wings	https://youtu.be/IF0UGwWVjLg
Rise Against - Make it Stop	https://youtu.be/SIGqnneLUWc
Sevendust – Enemy	https://youtu.be/8RdaBg4CWr4
Breaking Benjamin - Polyamorous	https://youtu.be/2vAtVlhuvac

Table 3.1 – The categorised dataset for (C♯/D♭) Major/(A♯/B♭) Minor

3.1.3 Track Retrieval and Management

This is the process of retrieving the audio of each track contained within the entire dataset, managing the audio files to ensure consistent formatting, and structuring the data in an accessible and useful way. In order to maintain a navigable and clear organisation of tracks, they will be stored in a single directory, divided into sub-directories by their key, meaning that they can be easily iterated through in Python code. Additionally, an entirely separate directory of the exact same structure will be created in order to store two tracks from the dataset of each key, totalling to 24 tracks. These tracks will remain in this separate directory until a further point in the project, when the best-performing new implementation of the algorithm will be compared to the baseline with two additional tracks per key added to the dataset. This is done to ensure that the best-performing solution does not simply overfit the dataset upon which all implementations will be examined and will serve as an experiment upon how well both the baseline and best-performing new implementation of the algorithm perform in the presence of new data and an expanded dataset. The songs that are added to this ringfenced data directory are chosen completely at random under no specific criteria.

The procedure of retrieving the audio from each track is done using Python and the youtube_dl and Pandas libraries. The portion of the dataset belonging to each key is loaded into the program in the .csv file to which it was saved. Then, this is converted to a Pandas DataFrame, which allows for simpler indexing and navigation. Once this is complete, each entry in the table is iterated over, and both the title of the track and the YouTube link to the song are extracted from the table. The YouTube video ID is extracted from the link and used to query the YouTube API using the youtube_dl library. The query requests the audio track of the file specified by the ID taken from the link. The highest possible quality of audio is requested. The query also specifies the preferred codec, which I selected as .mp3, with a preferred bitrate of 192Kbps.

Once the files are retrieved from the YouTube servers, their codecs and formatting can be verified by using Python code to execute the FFMPEG file conversion command-line functionality

on all files that do not match the correct format, meaning it has a different codec, sample rate, or bit rate. This file conversion can be used to convert the file into the desired format and then overwrite the existing file so that all files in the dataset are of an identical format. This is done to ensure all files can be handled in the same way by the program, for the purposes of indexing to certain positions in a song, for example.

The ability to automate this process through code is what empowers this project to have a dataset of the size and scale that it does. Without the ability to automatically retrieve the audio files from a list of links and convert them to a desired format, if required, removes the need to manually create copies of each track by recording the playback from YouTube, or by using external services that allow the audio tracks of videos to be saved. File conversion would need to also be done manually through a program such as Audacity. These processes would be much more time-consuming than the building and operation of my code. Performing data collection manually in this manner would also not guarantee that the audio retrieved would be of the highest quality attainable, whereas the quality of the audio saved via the code can be assured to be of the highest quality that can possibly be delivered by YouTube servers, as that can be directly requested in the query sent to the YouTube API.

With each track now saved and stored in the directory allocated to the key in which the song is composed, experimentation and analysis of the tracks can begin.

3.2 Baseline Establishment

This section outlines the preparation of a standard implementation of the Krumhansl-Schmuckler Key Finding Algorithm to run over the entire dataset in order to attain a baseline set of results to use in later comparison and analysis Then, the formatting, organising and storage of these results will be outlined to represent the standard structure of results that will be used for each set of results attained in all experiments and tests upon the dataset.

3.2.1 Code Preparation

The intention of this is to create a baseline implementation of the algorithm in order to evaluate how the algorithm performs in its current form, without any pre-processing of the dataset and without the addition of extra steps to the execution of the program.

In this baseline implementation, each track will individually be loaded into the program, then passed into the algorithm, returning an estimate of the key in which the song is composed. This is done by iterating over every file in each sub-directory in the directory of tracks, then using the MonoLoader function from Essentia to load in each file individually and mix the stereo tracks together into a single track, stored as a variable.

With the track loaded in, an Essentia KeyExtractor object can then be instantiated, while specifying that the chosen method of key estimation will be using the Krumhansl-Schmuckler Key-Finding algorithm. Then, the track that was loaded in earlier can be passed into the KeyExtractor function to generate an estimate for the key in which the song was composed. The estimate is returned in three parts – the tonic note, whether the key is major or minor, and the correlation coefficient of the frequency profile of the song passed into the function and the tonal hierarchy of the estimate key. The estimate is formed by concatenating the estimate tonic note with the text that defines if the estimate is major or minor.

For the sake of algorithmic performance evaluation over the entire dataset, the evaluation will be entirely based on the correctness of the estimates outputted by the algorithm. This is because the vitally important factor that must be evaluated is the accuracy of the estimates returned by the algorithm, so for this reason, this is where the focus of the evaluation will remain, meaning the strength of the correlation that is returned by the algorithm is not required for analysis of the performance of individual implementations.

3.2.2 Result Formatting and Storage

Once an estimate has been obtained for a track from the code described in the previous section, it will be entered into a Pandas DataFrame object that has been created for that individual sub-directory. The DataFrame contains a column for the name of the track, as well as a column for the estimate. Before addition to the DataFrame, each estimate is checked to verify that it is correct by determining if the estimate is a substring of the title of the directory from which the file has been accessed. As the title of each directory is a hyphen-separated list of every possible name for that unique set of notes, for example, “C Major – A Minor”, this means that any estimate that matches any valid name for the categorised key of the song will be accepted as correct. If the estimate is correct, a count tracking the number of correct estimates will be incremented by one.

Once all tracks have been inputted into the algorithm, and their estimates have been stored in the DataFrame, an additional row will be added to the DataFrame to specify the total number of correct estimates that were made across the tracks in that sub-directory by using the final value of the aforementioned correct estimate count. Along with this, the total number of tracks in the sub-directory will be displayed in order to provide a simple reference for the total number of correct estimates compared to the total number of tracks. Finally, the DataFrame can then be saved to a results directory in a .csv format on disk. The .csv file will be given the same name as the sub-directory it contains estimates for. Each .csv file can be stored in a single results directory, meaning that the results directory mimics the layout of the tracks directory, simply replacing the sub-directories with .csv files containing the results of the key estimate for each file in that specific sub-directory. This results directory will then be backed up and labelled as the baseline set of results.

For future tests of new implementations of the algorithm, these will have their results gathered, saved, and managed in an identical manner to that of the baseline described above. The results directory will be named according to which approach to usage of the algorithm was taken to gather the results in question.

3.3 New Approach Testing

This section summarises the many different new approaches to the usage of the algorithm that were attempted throughout the project in order to determine the impact that these new steps and adjustments have on the output of results, if any at all. There were a number of initial adjustments made to the baseline code in order to become familiar with the usage of Essentia’s functions, such as implementing the same functionality with different, yet similar functions. Once familiarised with the best method of executing the desired system, the following adjustments were made in order to analyse the resulting change in performance:

3.3.1 Low-Pass Filters

A low-pass filter is an operation that can be performed on a signal that attenuates frequencies in the signal that are above a specified frequency threshold. This was used in order to evaluate if removing extremely high frequencies from the audio would improve performance. From my analysis of the frequencies most often found in the audio used in the dataset, it appears that a vast majority of the frequencies in the audio are under around 3000Hz. This means that any frequencies above this threshold are likely to be unintended noise or sounds that are not required for the determination of the key of the song. By removing these frequencies from the song, the frequencies created by the actual music and the sound being created by the instruments and vocals will gain more influence over the balance of the tonal hierarchy that is generated for the song. This also removes the possibility for the algorithm to determine high-frequency noise as the instance of a note if the frequency of the noise can be found by doubling the frequency of a note any number of times. For example, a frequency at 14,080Hz technically corresponds to the note A, however it is exceedingly unlikely that it will actually be intentionally used by a composer

throughout a piece of music. This means that removing this frequency from the audio could prove beneficial in obtaining a more accurate tonal hierarchy for the song, as levels of noise in the music can potentially be greatly reduced.

For this implementation, the process of applying a low-pass filter involves using the audio editing program Audacity. Audacity has a feature known as chains, that allows for the same operation or set of operations to be applied to multiple tracks at once. A chain can be created to apply a low-pass filter to a track at a chosen frequency, and then export a copy of the track, keeping the original track completely unchanged. The chain can then be applied to every track in the dataset before running the new, updated dataset through the algorithm and obtaining the results.

The low-pass filter was applied at 50Hz, 125Hz, 250Hz, 500Hz, 750Hz, 1000Hz, 1500Hz, 2000Hz, 2500Hz, and 3000Hz. This was done in order to evaluate how performance changed when the threshold at which frequencies were attenuated began to move into the space where a majority of the frequencies were intentional, musical sounds that were created directly by instruments and vocals. I was interested to see if removing sounds such as high-pitched vocals or guitars, thereby increasing the prevalence of instruments that operate at lower frequencies, such as bass guitar, would lead to an improvement in performance. I also did this in order to see if there became a point where the usage of the low-pass filter became detrimental to performance, as opposed to a method for improvement of results.

3.3.2 High-Pass Filters

A high-pass filter essentially performs the mirror operation to that of a low-pass filter. While a low-pass filter attenuates frequencies from above a specified threshold, a high-pass filter attenuates frequencies from below a specified threshold. This was used to investigate the impact to performance, if any, that could be found by removing low-pitched sounds from the audio in the dataset, and, like the low-pass filter approach, to examine the change in performance as the threshold used begins to move into the frequency range in which a majority of the sound is intentional sound created by the instruments and vocals, in order to determine how removing the instruments that operate at lower frequencies, such as drums and bass guitar, would change the output of the algorithm for a given track. This filter was implemented in an identical fashion to the low-pass filter – by using the chain function in Audacity, a new chain can be created that applies the same effect to the entire dataset, exporting these to new tracks in order to preserve the original audio. The new tracks can then run through the algorithm in order to attain new results.

The thresholds applied for the high-pass filter were exactly the same as those used for the low-pass filter: 50Hz, 125Hz, 250Hz, 500Hz, 750Hz, 1000Hz, 1500Hz, 2000Hz, 2500Hz, and 3000Hz. This applied the high-pass filter effect at different points right across the standard range of frequencies generally created by the instruments and vocals in the dataset.

3.3.3 Band-Pass Filters

A band-pass filter combines the effects of a low-pass filter and a high-pass filter to attenuate all frequencies in a signal that are outside a specified frequency range. This was used in order to test the impact of applying the effects of the best performing low-pass filter threshold and the best performing high-pass filter threshold at the same time.

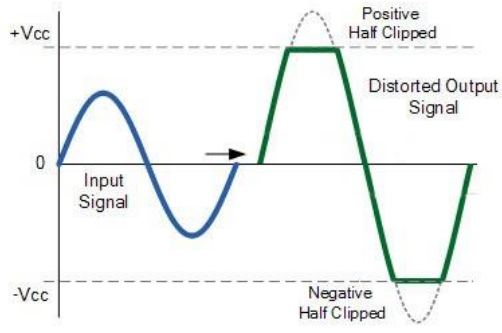
This operation was performed in two different ways. Firstly, it was performed in an identical manner to the low-pass and high-pass filter implementations – by creating a new chain in Audacity to apply the effects of a band-pass filter with the desired parameters to each track in the dataset. The second implementation of the band-pass filter was created in code by using the low-pass filter function from Essentia, followed by the high-pass filter function. This effect was attempted in both ways in order to investigate the impact of roll-off in filters on the estimates returned by this implementation of the algorithm. Roll-off describes the application of effects such as this in

analogue filters, where, once a given condition is met, such as a frequency threshold being surpassed, the filter will begin to apply its effect to the signal – however, the intensity of the effect is based on the degree to which the signal breaks the threshold set for the application of the effect of the filter; If a low-pass filter, for example, has a frequency threshold of 100Hz, any signal with a frequency above this threshold will be acted upon by the filter, but a signal with a frequency of 101Hz may not be acted upon in the same way that a signal with a frequency of 10,000Hz will. This is because these filters have what is known as a transition region, throughout which the application of the effect of the filter will be intensified as the degree to which the activation threshold is surpassed increases. Roll-off describes how steep the slope of the activation region is for the filter in question. A theoretical low-pass filter with an infinitely steep roll-off would act exactly the same on all signals with frequencies over its threshold, while a low-pass filter with a very small value for its roll-off, indicating a wide transition region, will impact signals that narrowly surpass the frequency threshold in a much less intense way than signals that surpass the frequency threshold to an extreme degree.[34]

The intention of applying this effect in two different ways is to investigate the difference in the output as a result of a change in roll-off. Filter effects in Audacity have parameters that allow for an element of control over the roll-off of the filter, whereas the filter functions in Essentia do not give any control regarding this. When applying the filter effects in Audacity, the roll-off was set to be as steep as the program would allow. The filters in Essentia, as mentioned before, cannot have their roll-off controlled in this way. Additionally, another implementation of this approach was attempted that applies the same filter a number of times in order to evaluate if this approach leads to a meaningful change in the performance over the dataset. The intention behind this is to see if applying an identical filter effect to a signal multiple times repeatedly will increase the level of attenuation to frequencies that fall within the transition region of the filter.

3.3.4 Amplification

The next adjustment to the algorithm that was tested was to amplify the audio in the dataset, which increases the magnitude of the entire signal by a specified number of decibels. The intention of this adjustment is to assess the influence that boosting the quieter elements of the audio will have on the estimates produced by the algorithm. Amplification also provides an insight into how the algorithm reacts in the presence of clipping in the audio, as amplifying sounds that are already loud in the signal can lead to a distortion in the signal. This occurs when the amplitude of the signal surpasses the maximum possible value, resulting in a change in the shape of the wave. An example of this can be seen in Figure 3.1:



[35]

Figure 3.1 – An example of the effect of clipping on a signal

The amplification effect was, like the previously described approaches to usage of the algorithm, applied via Audacity by using chains to perform the operation to the entire dataset before running the dataset through the algorithm to obtain results.

Amplification was applied at +5dB, +7dB, +10dB, +12dB and +15dB in order to create an overview of how the impact of amplification changes with increased intensity, if it changes at all.

3.3.5 Compression

Compression is an operation that reduces the dynamic range in a signal by attenuating louder elements of a signal, while boosting the quieter elements of the signal. This is generally done in audio engineering with the intention of providing more balance to the audio. This was done in this investigation with the intension of discovering if boosting the quieter elements of the audio, while maintaining or lowering the amplification of louder elements of the audio would impact or improve performance in any meaningful way.

Like the previously described approaches, this effect was applied using the chains functionality in Audacity. Various different compression ratios were tested for this approach, which dictates how much compression is applied to the signal - the higher the ratio, the more intense the level of compression applied. The compression ratios used were 1.1:1, 2.5:1, 5:1 and 7:1.

An important reason that led me to investigate the impact of compression on algorithmic performance was an interest in 'The Loudness War', which is a term used to describe the general music industrial shift towards producing louder and louder songs.[36] Part of this process included reducing the dynamic range of the audio signal, before amplifying the resulting signal. Although this does increase the loudness of the music, it can also introduce audio distortion and clipping if not done perfectly. This trend has seen large amounts of discussion since the introduction of digital audio processing and engineering in the mainstream music industry, meaning this process can be implemented more aggressively as a result of the ease of access and control of the filters and tools required to perform this. As different levels of amplification of each track in the dataset had previously been tested, I believed that, in the interest of producing as thorough an investigation as possible, different levels of compression of each track should also be tested.

3.3.6 Beat Subsampling

This process involves breaking the song down into multiple segments, running each segment of the song through the algorithm individually to gather a number of estimates, and then performing analysis on these estimates in order to determine the final estimate to be returned by the program for the track in question. This method aims to give consideration to sections of sound that may exist in songs that are almost entirely non-musical, for example, portions of a song comprised of speech, sound effects and/or crowd noise. These elements that may possibly exist in songs contribute frequencies to the audio that can potentially be mapped to instances of notes by the algorithm during its creation of a tonal hierarchy for the track. This means that this non-musical data could potentially be negatively impacting the resulting correlation coefficient that will be calculated between the frequency profile generated for the song and the key profile of the actual key of the song. In addition, this aims to reduce the negative impact of portions of songs that are written in a different key than the rest of the song, as these can add occurrences of notes that do not belong to the key that the majority of the song is composed in.

The goal of this method is to reduce the impact that non-musical and out-of-key portions of sound that may exist in songs have on the overall tonal hierarchy that is generated for the song, therefore the final estimate, while minimally impacting the actual music. Ideally, if a portion of non-musical audio exists in a track, provided that the total length of this portion of the song is shorter than the total length of the musical portion of the song, once the song is split into segments, the total number of segments involving the actual musical elements of the song will outnumber the segments involving the non-musical portion. Then, provided the musical elements of the song mostly generate relatively strongly-correlating, correct estimates, the analysis of estimates should then select the correct key for the final estimate, when this may not have been done in the case of estimates generated by the baseline implementation of the algorithm.

I determined that the best way to divide the song into segments was to define each segment to be a specified number of beats in length. This is because music is, in many cases, cyclical, where a majority of a song is built from a looping pattern that is a certain number of beats in length.[37] This

method of dividing the song into segments has a greater potential benefit to performance than using another approach to determining segment size, such as defining a segment to contain a certain percentage of the total number of samples, or to define a segment as a certain number of seconds in length. The benefit this method provides is that it has the ability to lead to a more balanced distribution of notes throughout each segment of the song, that will hopefully match the tonal hierarchy of the key of the song more closely. If this is the case, this can improve the likelihood of the tonal hierarchy of the segment creating a much stronger correlation coefficient when calculated with the key profile of the true key of the song. For this reason, this method was the best way of deciding the size of segments, however it also introduces more calculation that needs to be done before dividing the song into its segments.

To break the song into segments a specified number of beats in length, information regarding the tempo of the song must be gained. Essentia has a function that can extract rhythmic data from an audio file, and this includes being able to estimate the beats per minute of a song by analysing the peaks in the waveform of the audio. With the beats per minute of the song found, and the sample rate of the song already known, the following sequence of equations can be used to determine the number of samples required per segment of the song, as well as other information regarding the track, such as its length in seconds and the number of segments that will be created for the analysis of the track:

$$Length\ of\ Song\ in\ Seconds = \frac{Number\ of\ Samples}{Sample\ Rate}$$

$$Beats\ Per\ Second = \frac{Beats\ Per\ Minute}{60}$$

$$Samples\ Per\ Beat = \frac{Sample\ Rate}{Beats\ Per\ Second}$$

$$Samples\ Per\ Segment = Samples\ Per\ Beat * Beats\ Per\ Segment$$

$$Number\ of\ Segments = round\left(\frac{Number\ of\ Samples}{Samples\ Per\ Segment}\right)$$

Figure 3.2 – Equations for determining segment size and information for beat subsampling method

With segment size finally determined, the song can be divided into these segments, with each segment being run through the algorithm. All segments will be equal in length, except for the final segment in the case where the segment size does not evenly divide into the total size of the track. In this case, the final segment will have any additional remaining samples added to it in order to ensure that all of the track is accounted for in analysis.

Once an estimate has been returned for each segment, analysis can be done on the key estimates and their correlation coefficients in order to determine which key should be returned as the overall estimate for the track. A number of different methods can be used to determine this, for example:

- Calculating a total correlation coefficient for each key estimated by adding the correlation coefficients of each estimate for the same key, then selecting the highest value. This intends to select the key that is estimated frequently with a strong correlation.

- Calculating the average correlation coefficient for each key estimated by dividing the total correlation coefficient for each key by the number of times it was estimated. This intends to select the key that consistently correlated strongest.

- Selecting the key that was estimated most often. This simply intends to select the most common key.

This method was tested with segment sizes of 60, 96, 120 and 144 beats per segment.

3.3.7 Beat Range Subsampling

This process aims to provide an additional benefit to the beat subsampling method described in Section 3.3.6. An issue that I realised would arise with the standard beat subsampling method that I created was regarding selecting an ideal number of beats to divide segments into. Because there are so many rhythmic and cyclical variables in music, in that songs can have such vastly different tempos, and can be built on loops that are of massively varying length, it is unlikely that there will be a 'magic number' of beats per segment to specify that will break up each song in the ideal manner. For this reason, I decided to add an additional layer of complexity to the previously described method. This implementation will run the track through the beat subsampling method a number of times, specifying a different number of beats per segment each time. Like the standard beat subsampling method, this gathers a number of estimates that can be analysed further in order to determine the best option to return as the final estimate. However, due to the fact that not all segments that are analysed will be the same size, the process for final estimate key selection is changed.

Calculating a total correlation coefficient for each key, or selecting the key estimated most often will no longer be a fair method of selection, as this heavily favours the iterations of the algorithm that use smaller segment sizes. Smaller segment sizes result in more segments being tested, which can lead to more correlation coefficients being generated, giving the key estimates generated by the algorithm runs with the smaller segment size parameters a greater chance of being selected. This method requires average correlation coefficient to be used for selecting between the estimates returned from each iteration of the beat subsampling implementation of the algorithm. That being said, this does not require average correlation coefficient to be metric for estimate selection on a per-segment basis. This means that this method can be tested in multiple ways – with beat subsampling returning the key estimate with the highest total correlation coefficient, which can then have its average calculated once returned, and also by simply returning the key estimate with the highest average correlation coefficient.

The beat subsampling method is run ten times in total, starting with a beat per segment size of 36 beats, increasing by 12 beats per segment each run. This means the final run will have a beats per segment size of 144 beats. The beats per segment size was increased by 12 beats per run in order to increment the size by a number that is a common multiple of a number of common values that tend to represent the number of beats in a bar in music. A large amount of music is written to contain 3, 4 or 6 beats per bar, meaning that incrementing segment size by 12 beats per segment will increase the segment size by 4, 3 or 2 bars respectively.

3.3.8 Tuning Frequency Adjustment

In my analysis of songs in the dataset, I found that a number of songs that the baseline implementation of the algorithm produced incorrect estimates for were slightly out of tune. The standard method of tuning musical instruments is to tune A4, that is, the note A on the 4th octave, to 440Hz. From there, each note on the instrument can be tuned using the formula in Figure 3.3:

$$F_{note} = \sqrt[12]{2} * (F_{Previous\ Note})$$

Where F_n represents the frequency of note n

Figure 3.3 – Formula for standard instrument tuning

This method is known as the Twelve-Tone Equal Temperament tuning system.[38] There are, however, different methods for tuning instruments, which can all result in small differences in the frequency values that each note will be set at. These tuning differences can come about by

inaccurate tuning or by usage of one of these non-standard tuning methods. Using the standard method, the ratio between the same note one octave apart will be exactly 2:1. However, the ratio between different notes may not be as ideal as it could be if tuned using a different method of tuning. Alternative tuning methods, such as Just Intonation[39], may produce ideal intervals between different notes within the same octave, but can then fail to maintain accurate ratios between the same note across multiple octaves. These differences in frequency can often be barely distinguishable by the human ear in most circumstances, however these small differences in values can lead to the notes not being recognised as instances of the intended note when the tonal hierarchy of the track is being generated. This can lead to a weaker correlation coefficient between the tonal hierarchy of the track and the key profile of the key of the song, increasing the possibility of an incorrect estimate being made. Another reason why songs may be slightly out of tune is due to the age of the song – songs that were originally recorded onto tape may have been recorded perfectly at the time, however it is possible for tape to warp, shrink and fold depending on the conditions in which it was stored. This means that when the music was transferred to digital formats, the music could sound slightly different to the original recording depending on the conditions in which the original tape recording was stored. In addition, if a tape head reading the original tape was not set at the same speed as the tape head that recorded onto the tape, playback speed could be altered, resulting in different pitches being recorded in the digital transfer.

In order to try to improve the detection of all intended notes in the music, this approach runs the song through the algorithm multiple times, with each iteration running the algorithm using a different tuning frequency for A4 each time. Then, once a range of tuning frequencies have been tested, the estimates returned can be evaluated in order to select the best possible choice for the overall estimate for the track. The possible methods used to evaluate estimates are identical to those used in the beat subsampling method.

This approach to the algorithm was implemented using two different ranges of tuning frequencies. Firstly, this was attempted over the range of 432Hz – 448Hz. Then, with a range of 428Hz – 452Hz. Additionally, this range was tested in steps of 1Hz per iteration and 0.5Hz per iteration.

3.3.9 Approach Combination

Once all of the previously discussed implementations had been created and tested individually, I began to combine the effects of a number of these implementations together in order to evaluate if adding together the benefits of various new approaches would have an increased benefit to the overall performance. In addition, this was done to investigate if the performance of these combined methods would be less than, equal to, or greater than the sum of its parts, so to speak.

The pairs of methods that were implemented in a combination were:

- Compression and Amplification: This was done to simulate the process of increasing loudness of a music track as described in the Compression section (Section 3.3.5). Tracks first had the compressor effect applied, before having the resulting signal then being amplified.

- Beat Range Subsampling and Band-Pass Filtering: This was done as a result of finding promising results for the two individual performances of these implementations. The approaches were combined by running each track through the band-pass filter as described in Section 3.3.3, before then running the resulting tracks through the beat range subsampling method described in Section 3.3.7.

3.4 Specific Feature Testing & Experimentation

3.4.1 Drum Kits

Drums are an extremely common instrument within the rock and metal genre. They are used to provide rhythmic quality to a song, as well as to keep the performance of a group of musicians in time. Drums are a rather unique instrument compared to other common instruments used in this genre of music, such as the electric and bass guitar, because the drums are not melodic in the same sense that any of these other instruments are. They exist primarily for their rhythmic contribution, while in comparison, vocals and guitars, for instance, provide more melody and harmony to the music than the drums. As a result, drums are generally not as finely tuned as these other instruments in terms of the pitch the instruments create. For this reason, I decided to investigate the frequency contribution made by drums to tracks in which they are present.

A number of different resources were required for this investigation. The first element of this involved performing analysis of tracks that were completely comprised of drums and drums alone, as the intention was to analyse the common frequencies in sounds that were made by the drums. In order to ensure an accurate representation of the sounds that can be made by the drums, samples of audio from three entirely different drum kits were collected.

To perform analysis on the audio, I wrote code in Python to load the audio into the program and perform the Fourier Transform on the signal. The results of this can be analysed to gain an understanding of the significance of each frequency involved in the audio signal. This allows for an improved insight into the most common frequencies created in each drum track, which is hugely beneficial for determining the influence the presence of drums could have on the estimate generated for a track that includes them. Additionally, each drum track is then run through the baseline implementation of the algorithm in order to determine what key is estimated for a track comprised purely of drum sounds, as well as to evaluate the correlation coefficient that is generated between the tonal hierarchy of each drum track and the key profile of its estimated key. Finally, a piano-only track was examined in the same way in order to determine if there was any noticeable difference in the results of the analysis.

The next portion of this investigation involves obtaining the ‘stems’ of a song. These are isolated tracks that contain only one element of a song each. This means a song will be divided up over a number of tracks - one for drums, one for vocals, one for guitar, and so on. Generally, these stems are rather rare and difficult to source, as music labels and rightsholders will not release full, studio-recorded stems of any of the songs in their libraries – at best, acapella (vocals only) or instrumental (instruments only) tracks may be released, but fully isolated stems of a track are rarely publicly released. Luckily, there is an exception to the rule that allows for this investigation to go ahead for this project, and it is thanks to the popularity of Guitar Hero.

Guitar Hero is a series of rhythm games that released in 2005, releasing over ten games following its first release on a variety of gaming platforms. The objective of the game is to replicate a performance of the part of a chosen instrument in a song from the library of music in the game. The part of each playable instrument is represented in-game by a series of buttons, and the player must press the correct button (or buttons) at the correct time in order to successfully play the game.[40] In early releases from the series, the only instruments a player could choose were the lead or bass guitar, however in later games in the series, the playable instruments were expanded to include more parts for each song, namely drums. The music libraries of these games are comprised of licensed music from a wide variety of artists from many different music genres, including rock and metal music. When a player selects a song, the track will begin to play, and the part of the instrument that the player has chosen to play along with will only be heard in the music while the player correctly plays the game, pressing the correct buttons at the correct time. In order to do this, the game requires isolated stems for each song in the track list. As this game series uses officially licensed music, the game developers are granted access to the official, studio-recorded stems of each song that they have licensed for use. This means that the stems of each song in the track list of a Guitar Hero game are contained within the files of that game. Due to the

fact that the many of these games have been released on PC, the process of extracting these stems from the game files is rather simple. In order to acquire the stems needed for this experiment, any song from the rock/metal genre of music that has been included in a Guitar Hero game released on PC that features drums can be selected. A song that fits this description is 'Nothing Else Matters' by Metallica, which was the song selected for this experiment.

After sourcing the stems for the song, they could be mixed together to create two new tracks – one is made by mixing together all of the stems, resulting in a track identical to the original release of the song, while another is made by mixing together every stem except for the drums. The resulting track is what the song would sound like without the presence of drums. These tracks can then both be inputted into the baseline implementation of the algorithm and their resulting estimates and correlation coefficients can be compared in order to determine the impact made by the presence of drums in a track.

Once this experiment was complete, similar comparisons were made by mixing in drum tracks to other songs and comparing the estimates returned before and after the addition of the extra drum sound to the track.

3.4.2 Effects Pedals

Effects pedals are common tools used in modern rock and metal music. They are devices that will take an input signal, normally from an electric guitar, and will perform a specific operation on the signal, adjusting it before it is outputted via an amplifier or inputted to a recording device. An example of an operation that an effect pedal could apply is a reverberation effect. However, there are various effects that can be applied that can drastically change the resulting output signal. The intention of these devices is to add a specific tone and timbre to the instrument being played. Common effects pedals used in metal and rock music include Distortion, Overdrive and Phaser pedals. As part of my investigation into the impact of common features from the rock and metal genre of music on performance of this particular algorithm, I wanted to investigate the change to estimates that is made in the presence of these effects.

In order to analyse this, a standard input signal is required. This input signal should be a piece of music that does not have any effects applied to it, and should be composed in a manner that allows for its key to be recognizable, that is, it should not include notes from outside of the chosen key signature. This input signal will then be re-recorded with one or more of a various number of effects applied to it, and the baseline algorithm will be used to generate an estimate for the resulting audio. This process will be repeated until each effect has been applied and an estimate has been attained for each resulting audio signal. The estimates and correlation coefficients for these tracks can then be compared to investigate the impact, if any, of these effects on the ability for the algorithm to estimate the key of a piece of music accurately.

The sample track used in this experiment was composed digitally in order to minimise external noise generated in the recording. The composition is the "1, 1-2-1, 1-2-3-2-1…" musical exercise, in the key of C major. This warm-up exercise begins on the tonic (first) note of the key, which is played once. Then, the first note will be played again, followed by the second note in the key, then the first note again. This pattern continues, following the structure of adding one more unique note with each run through the pattern, until one octave has been traversed, meaning there are 7 unique notes. Overall, the tonal hierarchy of this composition should correlate to the C major key profile strongly, as the note C occurs the most often by far, and each note in the C major key occurs, with no notes from outside of this key occurring at all.

To apply the effects to the track, a pedal board is used. This is a physical board with a number of effects pedals daisy-chained together on it, allowing for the effects of each pedal to be switched on and off as desired, as well as allowing the dials on each pedal to be adjusted in order to change the intensity of each effect that is currently being applied. The sample track then has the pedal effects applied to it in the following manner:

- The sample track is repeatedly outputted from a playback device via 3.5mm audio jack.
- The signal is inputted to the first pedal on the pedal board via ¼ inch audio jack.
- The desired pedal(s) on the pedal board are activated and adjusted until the desired setup is attained.
- The output of the last pedal on the pedal board is inputted to a digital audio interface via ¼ inch audio jack.
- The digital audio interface is connected to a computer allowing the resulting signal outputted by the pedal board to be recorded in software, before being saved as a .mp3 file.

This process eliminates the need for microphones or speakers in the recording process, which ensures the absolute minimum interference and noise possible in the recording, minimising the variables at play that can have their own impact on the resulting estimate.

Figure 3.4 – The pedal board in use during the effects pedal experiment

Looking at Figure 3.4, the black cable on the bottom right of the image carries the audio input to the first pedal, which outputs the resulting signal to the next pedal, and so on, until the final pedal outputs the signal to the audio interface via the green cable seen at the bottom of the image. Each pedal will only apply its own specific effect to the signal if it has been activated. This means that despite the signal being routed through each pedal, not every pedal will be activated at once.

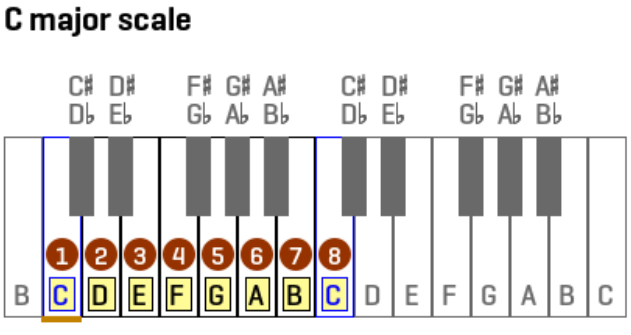
Once a reasonable sample size of new tracks has been created by applying pedal effects to the original track, these tracks can be inputted into the algorithm in order to obtain key estimates and correlation coefficient values that can be compared. In addition, the individual tracks can be analysed further by applying the Fourier Transform and plotting the results along with a waveform of the track in order to compare the resulting visualisations. The following pedals and combinations of pedals were used for this experiment:

- No Effects
- Distortion Pedal 1 (at Low, Medium and High Settings)
- Distortion Pedal 2 (High Settings)
- Chorus Pedal
- Phase Shifter Pedal
- Delay Pedal (Adjusting Settings during playback)
- Overdrive Pedal
- Distortion and Chorus Pedals (High Settings)

3.4.3 Modes

Modes are a more advanced element of music theory that can be considered an extension or expansion of the concept of keys. As mentioned in the terminology section, a key is a set of notes taken from a chosen major or minor scale. The tonal centre of a key is its tonic note, which is the starting note of the scale, therefore the first note in the key. A mode, simply put, is a rotation of a

given major scale, so to speak. It uses the same notes, however where the scale begins can be changed depending on the mode. This can result in a difference in the tonic note, and therefore the tonal centre. Take, for example, the C major scale, as shown in Figure 3.5:



[41]

Figure 3.5 – The C major scale represented on a piano

The individual notes in this key, from first to last, are C, D, E, F, G, A, and B. If this key were to described as a mode instead of a key, this can be called called C Ionian. A useful way of thinking about modes is to consider Ionian to indicate that the base/tonic note, in this example, C, is the first note from the key that it shares notes with. In this case, this means that it is completely identical to the C major scale. As there are seven unique notes in the key, there are seven unique notes that a mode can begin at, meaning there are seven modes. All of the modes, and the position of their tonic notes in the original key, are as follows: Ionian (1st note), Dorian (2nd note), Phrygian (3rd note), Lydian (4th note), Mixolydian (5th note), Aeolian (6th note), Locrian (7th note).

Applying these modes to the key of C major, the resulting modes can be seen in Figure 3.6.

C Ionian						
C	D	E	F	G	A	B
D Dorian						
D	E	F	G	A	B	C
E Phrygian						
E	F	G	A	B	C	D
F Lydian						
F	G	A	B	C	D	E
G Mixolydian						
G	A	B	C	D	E	F
A Aeolian						
A	B	C	D	E	F	G
B Locrian						
B	C	D	E	F	G	A

Figure 3.6 – A visualisation of the notes comprising each mode of the C major scale

As can be seen from Figure 3.6, these modes all use the exact same notes, in the same way that relative major & minor scales do. In fact, just as the C major scale can be described as C Ionian, the A minor scale can be described as A Aeolian. As with relative major and minor scales, the notes used are the same, however the usage of the notes within these modes is what separates them from one another musically. The difference in tonic note of these modes impacts the tonal centre of the music, which is the note that the piece of music will gravitate towards and return to frequently in order to begin and complete melodies.

Modes are arguably as useful and important as key is for musicians and composers. Modes can be considered musical building blocks in the same way that keys can, as composers are completely free to compose a song using a mode as their musical reference rather than a key. Particularly with metal music, which is potentially more likely to have a more emotionally dark tone in terms of musical and lyrical content, certain modes may be more suitable than certain keys in fulfilling the goal of the composer to evoke the desired response from a listener, and to establish the desired

atmosphere in a piece of music. This is because the varied tonal centre of the modes results in a change in the level of tension among the notes that comprise the mode.

A noteworthy feature of all modes other than Ionian and Aeolian is that these modes do not necessarily match any major or minor keys with regards to both unique notes used and tonal centre.

In order to investigate the impact of modes and varied tonal centres on accurate algorithmic key recognition, I used the “1, 1-2-1, 1-2-3-2-1…” musical warm up exercise sample track that was composed for the effects pedal experiments described in Section 3.4.2, and transposed the composition into each mode. This means that an identically composed sample track is obtained, where the only difference between each track is the mode in which it is written. Each sample track is composed using the exact same notes, the only feature separating each track is the number of occurrences of each note. The tracks are then able to be inputted into the baseline implementation of the algorithm in order to analyse the estimates outputted, as well as the correlation between the tonal hierarchy of the track and that of the key that the algorithm estimated the track to be in.

3.5 Performance Evaluation Metrics

This section describes the various different metrics and methods of evaluating the performance of the sets of results obtained by the numerous new approaches to the usage of the algorithm that were used throughout this investigation.

3.5.1 Total Correct Estimates

This is perhaps the simplest of all the metrics that were used in the evaluation of how a set of results performed. This simply measures how many correct key estimates were made for each track. This is measured on a per-key basis as well as over the entire dataset.

3.5.2 Total Note Matches

This metric is the first of many metrics that places greater focus on the incorrect estimates than the correct ones. Despite the fact that estimates can essentially only be correct or incorrect, there is a lot of valuable insight that can be gained from analysis of the incorrect estimates. This method involves determining how many of the notes that comprise the estimate key match notes that comprise the actual key of the song that the estimate has been generated for. Each key is made up of 7 unique notes, and there are 12 possible unique notes that any key can include. This means that, at an absolute minimum, any two keys will share at least two notes. This also means that correct key estimates can be considered to match all 7 notes that comprise the true key. Any key estimates that are not correct can have their number of matched notes calculated by determining the notes that comprise the estimate scale, then the notes that comprise the true scale, and by evaluating the number of identical notes within each set of notes.

This metric can give us an indication of how close an estimate was to being correct, despite the fact that it was incorrect. For example, C major and G major share 6 notes with one another, meaning that if a song was written in G major, but the algorithm estimated the track to be in C major, the estimate can be considered to be relatively close to being correct, as the two keys share 6 out of 7 unique notes. Contrarily, if the algorithm estimated the key of the track to be in C♯ major, the two keys would only share two notes with one another, which would indicate a very poorly chosen estimate as there is such a significant difference between the two keys.

Like the total correct estimate metric discussed in Section 3.5.1, this can also be measured on a per-key basis and over the entire dataset.

3.5.3 Weight-Based Key Closeness Scoring

This metric aims to achieve a similar goal to the previously described metric, which is to determine and convey how close an incorrect estimate is to being correct, however this metric is used to do this with greater specificity. This metric goes further than simply evaluating which notes from the true key are present in the estimate key, and accounts for the location of the matching notes and determines the closeness of two keys by using the values given to each possible note in each scale by Krumhansl and Schmuckler through their experiments as described in Section 2. These values can be considered to be 'weights', where the larger the weight of the note, the greater the importance of that note to the key, and the better it tends to fit the music written in that key. The values from Table 2.2 can be applied to the true key of the song in question in order to attain values to represent the importance of every possible note to the key in which the song is composed. Then, the notes from the estimate key can be mapped to their weights in relation to the true key of the song, and the values allocated to each note from the estimate key can be added together, and this total can then be divided by the total of the weights allocated to each note in the true key. The closer the resulting value is to 1, the closer the key is to the true key.

Using the previous examples from Section 3.5.2, the process of scoring these keys can be applied to these in order to outline the process of determining how close the keys are to one another.

In this example, the true key of the song is G major. The key profile of G major can be seen in Table 3.2:

G Major Key Profile											
G	G♯	A	A♯	B	C	C♯	D	D♯	E	F	F♯
6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88

Table 3.2 – The G major key profile

Due to the fact that relative minor keys are considered as also being correct estimates, the key profile for E minor, the relative minor of G major, will also be required. This can be seen in Table 3.3:

E Minor Key Profile											
E	F	F♯	G	G♯	A	A♯	B	C	C♯	D	D♯
6.33	2.68	3.52	5.58	2.60	3.53	2.54	4.75	3.98	2.69	3.34	3.17

Table 3.3 – The E Minor key profile

Mapping the notes of C major, the estimate in this example, to these key profiles will lead to the highlighted values in figures 3.4 and 3.5 to be used:

C Major in the G Major Key Profile											
G	G♯	A	A♯	B	C	C♯	D	D♯	E	F	F♯
6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88

Table 3.4 – The notes from the key of C major applied to the G major key profile

C Major in the E Minor Key Profile											
E	F	F♯	G	G♯	A	A♯	B	C	C♯	D	D♯
6.33	2.68	3.52	5.58	2.60	3.53	2.54	4.75	3.98	2.69	3.34	3.17

Table 3.5 – The notes from the key of C major applied to the E minor key profile

As mentioned in Section 2, the values mapped to a note in a major key profile will not be identical to the value mapped to that note in a minor key profile. In fact, the total of all values in each key profile will not be the same, either. In a major key profile, the largest total that can be attained by 7 notes is 30.03, whereas in a minor key profile this value is 31.03. In either case, this total will be attained by the notes from the key signature to which the key profile belongs, or its relative minor, as they share the same notes. Because the maximum values are different, the totals for each key profile are then divided by the maximum value attainable for that key profile. In this example, those values will be ~0.98 for the major key profile and ~0.97 for the minor key profile. To give the estimate the benefit of the doubt in a sense, the larger of the two values can be selected to allocate

to the estimate as its score. As this score is quite close to 1, it is clear that, despite this key estimate being incorrect, these keys are relatively close to one another.

If this estimate was a different key that also shared 6 out of 7 notes with the true key, this can lead to a different score, despite the fact that they share the same number of notes. This is because the scoring is based off of the weights applied to each note. If, for example, an estimate matched 6 out of 7 notes, but the missing note was weighted as one of the largest values in the key profile, the omission of this note is of much more significance than the omission of a note with one of the lowest weights in the key profile out of the notes that are actually part of the key. Likewise, the note that is in the place of the missing note from the true key is of importance, also. If the note included from outside the key is weighted as one of the largest values for a note outside the key, this error is less detrimental to the score than if the incorrectly included note was weighted with one of the smaller values for notes outside the key.

This process introduces variation of score within keys that share the same number of notes with the true key of the song in question, which can allow for the possibility of closer inspection into the performance of each set of results.

The possible score values range between ~0.7 and 1. The reason why no score can be lower than ~0.7 is because there will always be a minimum of 2 notes that will match the true key, and then with the addition of 5 more notes to contribute to the score, this means that the score cannot possibly go lower than this.

This score metric can be evaluated in a number of different ways, such as by measuring total or average score can be measured on a per-key basis as well as over the entire dataset. This measurement gives a general overview of how accurate estimates were within a set of results.

3.5.4 Average Score of Incorrect Estimates

This metric uses the scoring method described in the previous section and ignores the correct estimates in order to give a clearer indication of how far away the incorrect estimates are from being correct in a set of results. Average score of only incorrect estimates can be calculated in order to examine the performance only in the cases where estimates are not correct, both on a per-key basis and over the entire dataset.

3.5.5 Relative Performance Scoring

This metric aims to provide an overview of the overall performance of an implementation of the algorithm across all of the metrics previously described in this chapter. This is done in order to create a metric that can indicate the top performers out of the dataset at a glance.

This metric works by using the RankData function in SciPy, which can take a list of values and assign a rank to each value indicating where it positions among the other entries in the list. From this set of rankings, the top 30% of the results can be selected, and the implementations to which these sets of results belong to can be awarded points based on the position of the results within this top 30%. Results that finish higher will worth more points than results that finish lower. This is done across all of the previously described metrics, both on a per-key basis and across the entire dataset. When awarding points for performance over a metric measured across the entire dataset, scores allocated to implementations are multiplied by 12, as performance over the entire dataset is more important than performance over a single key or set of keys.

In order to ensure fair scoring in the case of tied results that are excluded from the top 30%, any sets of results outside the top 30% with the same value as the bottom performer of the top 30% will receive the same number of points as the implementation at the bottom of the top 30%. For example, when evaluating total correct estimates, if the last value in the list of the top 30% of performers over this metric attained 6 out of 10 correct estimates, and the next highest value out of

all sets of results also attained this level of performance, both implementations will be scored identically.

This method aims to reward the implementations of the algorithm that generate sets of results that perform well across all metrics and outperform as many other implementations as possible.

3.5.6 Tone-Specific Misses

This metric is a more specific measurement that is intended to be used in order to attempt to draw comparisons between failure cases, both on a per-key basis and over the entire dataset. This measures the tones (Do, Re, Mi…) that the incorrect estimates missed and keeps a count of these misses. The goal of this metric is to indicate which portions of keys are frequently being missed by incorrect estimates in order to help examine if there are any similarities in the notes that are not being included in estimate scales. For example, if the “Ti” tone from the true key was frequently being missed by incorrect estimates, this would be an interesting feature to notice, and could provide an insight into an area in which the algorithm could require improvement. Measuring this over individual keys can provide as much, if not more information regarding performance of each implementation than measuring over the entire dataset does.

4. Chapter Four - Results

This section describes the results of the various different experiments, tests and new algorithmic approaches that were carried out and implemented throughout this project, and details noteworthy information that can be taken from these results.

4.1 Experiment Results

This section will detail and discuss the results attained for the experiments carried out as described in Section 3.4.

4.1.1 Drum Experiments

As described in Section 3.4.1, this experiment can be divided into three different sections: Firstly, there is analysis of numerous tracks that include only drums. Secondly, there is a comparison between analysis of a drum-only track and a piano-only track. Finally, there is comparison between two tracks of the same song, with one of these tracks having drums omitted, as well as the comparison between a track that has had additional drum sounds added to it and the unedited original track.

Starting with analysis of drum-only tracks, the results of the analysis of each track can be seen in Table 4.1:

Track:	Kit 1, Sample 1	Kit 1, Sample 2	Kit 2	Kit 3
Most Common Frequency Range:	60Hz – 250Hz	65Hz – 250Hz	75Hz – 150Hz	75Hz – 150Hz
Most Common Frequency:	76Hz	136.4Hz	87.3Hz	85Hz
Note Closest to Most Common Frequency:	D♯	C♯	F	~ E/F
Notes at/near Other Common Frequencies:	D, C, B	D, C	D♯, E, G♯	G, G♯, A
Estimated Key:	C Major	C Major	F Minor	C Major
Correlation Coefficient	0.56069881…	0.62867594…	0.379003614…	0.401711255…

Table 4.1 – Results of Isolated Drum Kit Analysis

As can be seen in Table 4.1, not only do each of these different drum kits produce vastly different results in terms of the most common frequencies, but the strongest correlation coefficients that can be generated between the frequency profiles of these sample tracks and any key profile is generally considerably weak, with the largest correlation coefficient result of around 0.62 appearing to be an outlier in the data. In addition, it can be noted that two different samples of the exact same drum kit produced both different correlation coefficients, as well as different values for the most common frequency. This indicates that not only does the type of drum kit used make an impact on the frequency profile of the track, but also the way that the instrument is played can seemingly have an influence on the tonal hierarchy that is generated for the audio, as indicated by the fact that two different sample recordings from the same drum kit can produce varying outputs from the algorithm, and can lead to different observations during analysis.

This low correlation coefficient suggests that the frequencies that are contributed by the drums to an overall frequency profile of an audio track can oftentimes be likened to noise, as the musical contribution of the drums is demonstrably more rhythmic than melodic. The most common frequencies that can be associated with notes may be mapped to notes that do not exist together within any key. For example, in the results for drum kit 3, the most common frequency is close to that of both E and F, and the other common frequencies are close to G, G♯ and A. There is no key

that contains all of these notes, which can explain why such a low correlation coefficient is generated for the frequency profile of this track. Likewise, the common frequencies could also potentially map to notes that do exist together in the a key, however there is the possibility that the key in which they exist is not the key in which the song is composed, meaning that the tonal hierarchy of the track correlates less closely to the key profile of the actual key of the song. Overall, it can be considered extremely unlikely that the most common frequencies from a drum track will directly map to the frequencies of notes that are part of the key that the relevant song is composed in.

In addition to frequencies mapping to notes that do not coexist within any standard keys, there are many common frequencies that do not map to notes whatsoever, which seems to relate the overall frequency contribution of audio signals created by drums to noise even further.

When comparing the analysis of a drum track and a piano track, the waveform of the track as well as the results of the Fourier Transform were plotted in order to compare the difference between the results through visualisation. The plots produced can be seen in figures 4.1 and 4.2:

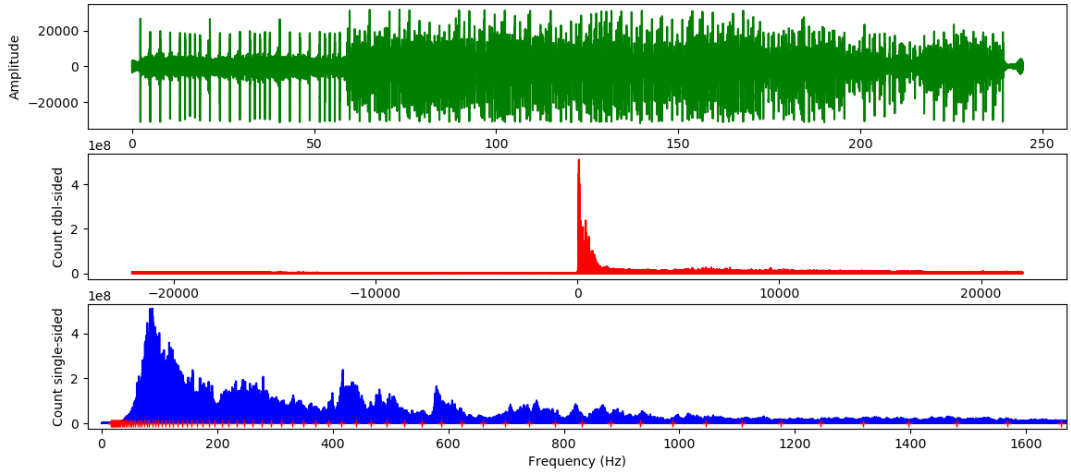


Figure 4.1 – The waveform and Fourier Transform results of an isolated drum track

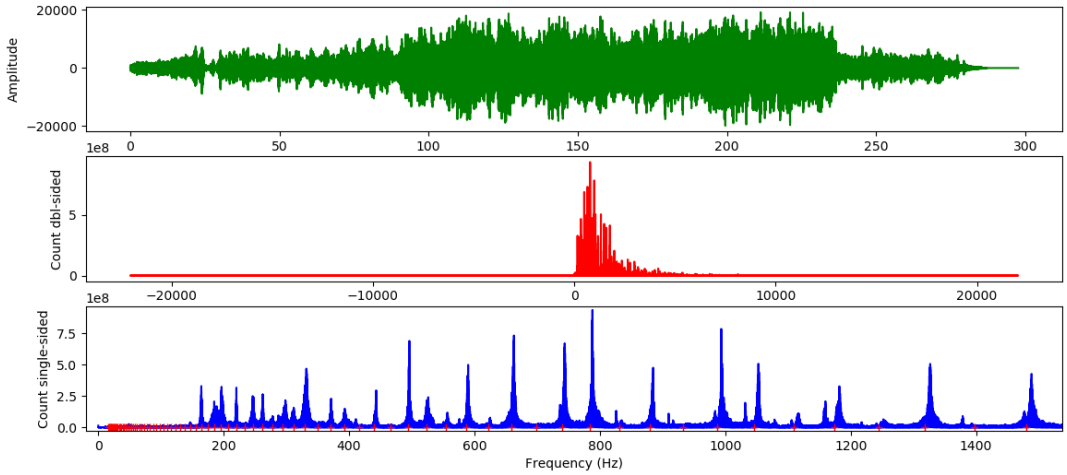


Figure 4.2 – The waveform and Fourier Transform results of an isolated piano track

Clearly, there are many differences between these graphs, which indicate a vast difference between the tonal structure of the two tracks.

Looking at the waveform of the tracks, which is the green plot in both Figure 4.1 and Figure 4.2, a noticeable difference between the two plots is obvious. In the drum track, there are a number of harsh, evenly spaced peaks, which indicate the strong rhythmic quality of the drum track. In comparison, the waveform of the piano track is more continuous, with less distinct peaks, completely unlike the drum track.

Moving onto analysis of the results of the Fourier Transform, the remaining two plots can be observed in each figure to notice the many differences in these results also. The red plot in each figure features the entire frequency spectrum, with negative frequency removed from the results for greater clarity. The blue plot features the same results; however, it is focused in on a frequency range of 0Hz to around 1500Hz. The red marks along the x-axis of this plot represent frequencies that correspond to notes.

Inspecting the structure of the plots of the results of the Fourier Transform on the drum track, the peaks of the plot can clearly be compared more to mounds, as they have wide bases, with the tallest point not being particularly large. The plot forms a relatively smooth curve, which indicates a huge number of common frequencies that do not correspond to musical notes. Frequency occurrences are comparatively quite evenly distributed along the range of 50Hz to 600Hz, which does not allow for a strong correlation coefficient to be calculated between the frequency profile generated for this track and the key profile of any key, as the even distribution of frequency significance means that almost every possible note will have a relatively even weight when the tonal hierarchy of the track is generated.

Comparing this to the same plots that are created for the piano track, we can see extremely distinct, tall, narrow peaks in the plots, directly at the frequencies to which notes are mapped. As well as this, these peaks are of various sizes, where some notes visibly have much larger peaks than others. Some notes have little to no tonal significance at all in this piece of music, and the size of the peaks at these frequencies signify this. The frequency occurrences in this track are concentrated around the notes that make up the key in which this piece of music is written. What this plot can tell us is that the piano used in the recording is in tune, and that the music that it plays remains in key for a vast majority of the time. Even without using the algorithm, from the plot alone the key of the track can be estimated with a relatively high level of confidence – the most common frequencies map to the notes B, E, G, F♯, C, A, and D. From these notes, G is the most common. With this knowledge, it can be assumed with relatively high confidence that the music is in the key of G Major, as this key includes all of these notes, and has G as its tonal centre.

Continuing onto the test between the tracks with and without drums included, the results returned by the algorithm for each track analysed in this test can be seen in Table 4.2:

Track:	Estimated Key:	Correlation Coefficient:
Nothing Else Matters (Original)	E Minor	0.8851658701896667
Nothing Else Matters (No Drums)	E Minor	0.9131715893745422
Love Reign O'er Me (Original)	E♭ Minor	0.7926527261734009
Love Reign O'er Me (Additional Drums)	E♭ Major	0.803708553314209

Table 4.2 – Results of algorithm estimates on tracks with drums, without drums, and with additional drums

In Table 4.2, we can see that no two estimates of the same song are identical when drum tracks have been omitted or added. Looking at the estimates returned for the original 'Nothing Else Matters' track, we can see that the estimate is correct, as indicated by the colour-coding in the table. The correlation is quite strong, at around 0.88, however comparing this to the estimate returned for the same track with drums omitted shows that another correct estimate is returned, but this time with a stronger correlation coefficient of around 0.91. These results show that the presence of the drum track in the recording has had a negative impact on the correlation value of the estimate. In the case of this song, both estimates are still correct, however the estimate made with the omission of the drum track is more reliable. The estimate made with the inclusion of the

drums has a correlation coefficient around 4.1% lower than the estimate made where the drum track is not included. However, in this particular song, the drums track does not account for as much of the sound in the song as the drums potentially could in other songs - this song is around 6 minutes and 30 seconds long, and the drums do not make a sound until after a minute of music has already been played. In addition to this, the drums are not played for the final 1 minute and 10 seconds of the song, meaning the drums are only played for a total of around 4 minutes and 17 seconds of the song. This lowers the amount of the sound within the signal that is created by the drums, meaning that the melodic elements of the audio have a greater influence over the total contribution of frequencies to the frequency profile generated for the file by the algorithm. This means that for other songs that feature drums that play more consistently and more aggressively throughout the entire song, the negative consequences of this effect could potentially be intensified.

Focusing on the results of 'Love Reign O'er Me', we can see that a correct estimate is returned by the algorithm for the original track, albeit with a comparatively low correlation coefficient when compared against 'Nothing Else Matters'. By simply adding additional drum sounds to this track, however, the estimate returned completely changes to an incorrect one, with a stronger correlation coefficient than the correct estimate returned for the original track. This is a significant result as it clearly indicates that, in the case that a song is perhaps not the most distinguishable in terms of accurately and confidently selecting a key estimate, the extent to which the drums feature in the song can have a defining influence over whether or not an accurate key estimate is made. The noise-like impact of the presence of drums can be the feature that moves the tonal hierarchy of a track past the point of correlating most strongly to the key in which the song is actually composed.

From these results it can be inferred that, generally, the less the drums are present in a song from this genre, the more capable the algorithm is of accurately estimating the key in which the song is composed, provided that there are no other musical features at play that can also negatively impact the performance of the algorithm.

4.1.2 Effects Pedal Experiment

This experiment, as described in Section 3.4.2, involved applying various effects to a sample track in order to create new audio tracks which could be tested by running the algorithm on each track and then comparing the estimates and correlation coefficient returned. In addition, the tracks were each individually analysed like the drum and piano tracks in Section 4.1.1 to evaluate and compare the structure of the results of the Fourier Transform when applied to each track.

The estimate and correlation coefficient returned by the algorithm for each track created by using a pedal or combination of pedals can be seen in Table 4.3:

Effect Applied:	Estimated Key:	Correlation Coefficient:
None	C Major	0.8793013691902161
Distortion Pedal 1 (Low Settings)	C Major	0.943261444568634
Distortion Pedal 1 (Medium Settings)	C Major	0.9523058533668518
Distortion Pedal 1 (High Settings)	C Major	0.9532442688941956
Distortion Pedal 2 (High Settings)	C Major	0.9255363941192627
Chorus Pedal	C Major	0.8362850546836853
Phase Shifter Pedal	C Major	0.8196940422058105
Delay Pedal (Adjusting Setting During Playback)	C Major	0.792957603931427
Overdrive Pedal	C Major	0.9469732046127319
Distortion and Chorus Pedals (High Settings)	C Major	0.9560806751251221

Table 4.3 – Results of algorithm estimates on tracks with pedal effects applied

Despite the fact that all of the estimates produced for these tracks are correct, there remains a considerable amount of variance between the strength of the correlation coefficients that is returned for each track. In numerous cases, the correlation coefficient calculated for the track is higher than the one returned for the original sample track. In order to explain this, it is beneficial to

look at the plots of the waveform and Fourier Transform of each track. These can be seen in figures 4.3 to 4.13:

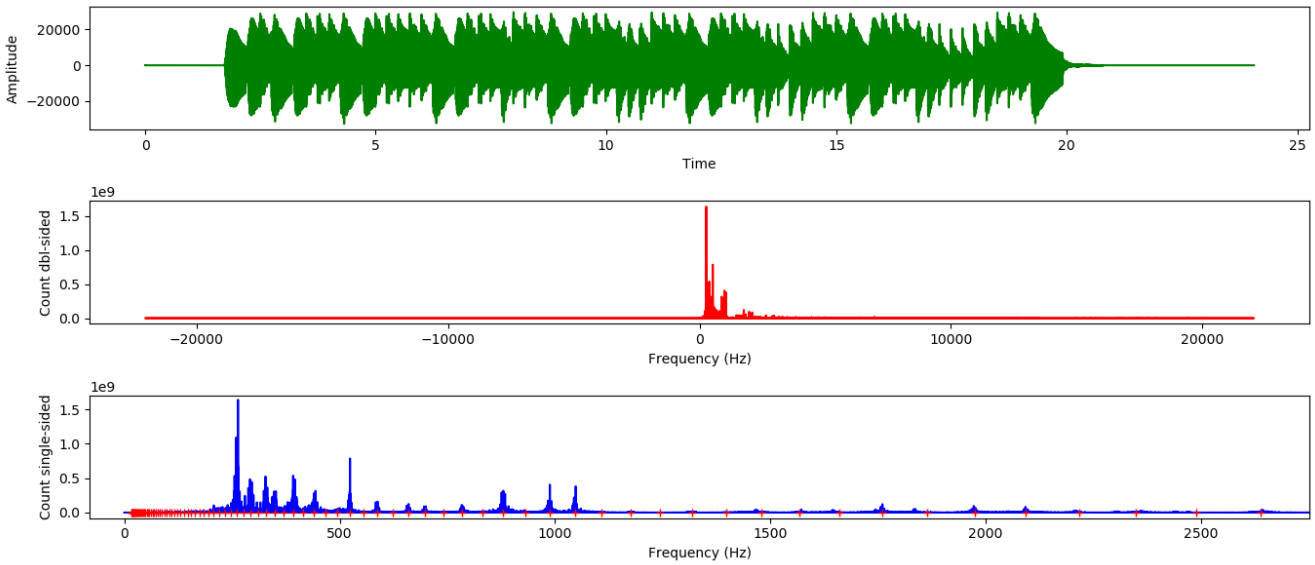


Figure 4.3 – The waveform and Fourier Transform results of the sample track with no effects applied

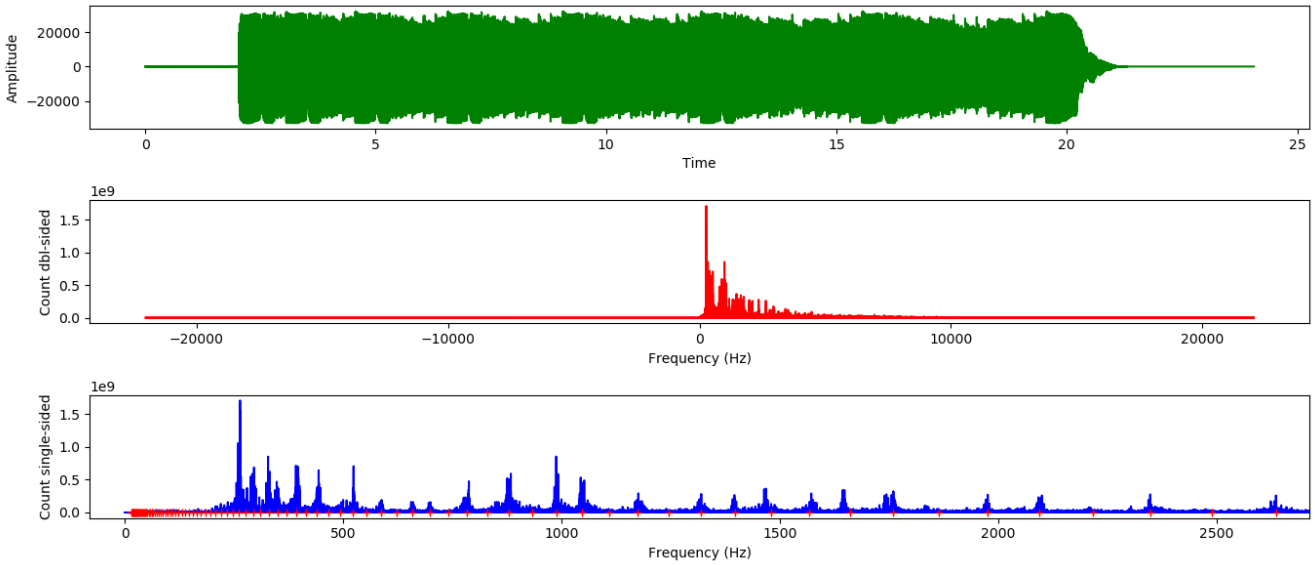


Figure 4.4 – The waveform and Fourier Transform results of the sample track with the first distortion pedal applied with low settings

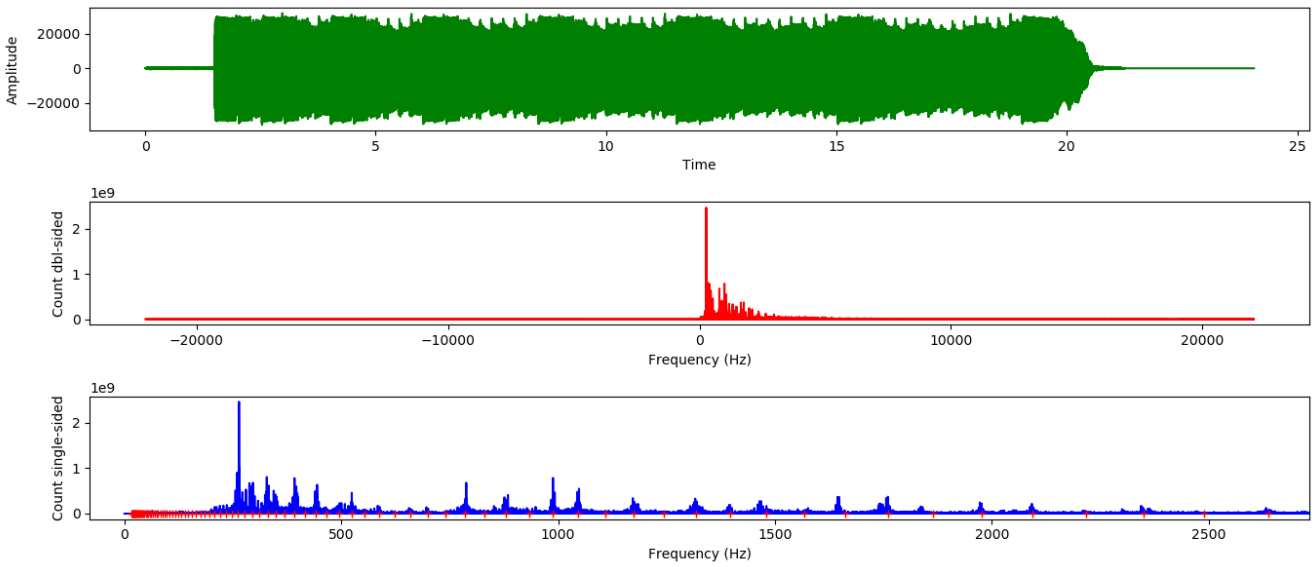


Figure 4.5 – The waveform and Fourier Transform results of the sample track with the first distortion pedal applied with medium settings

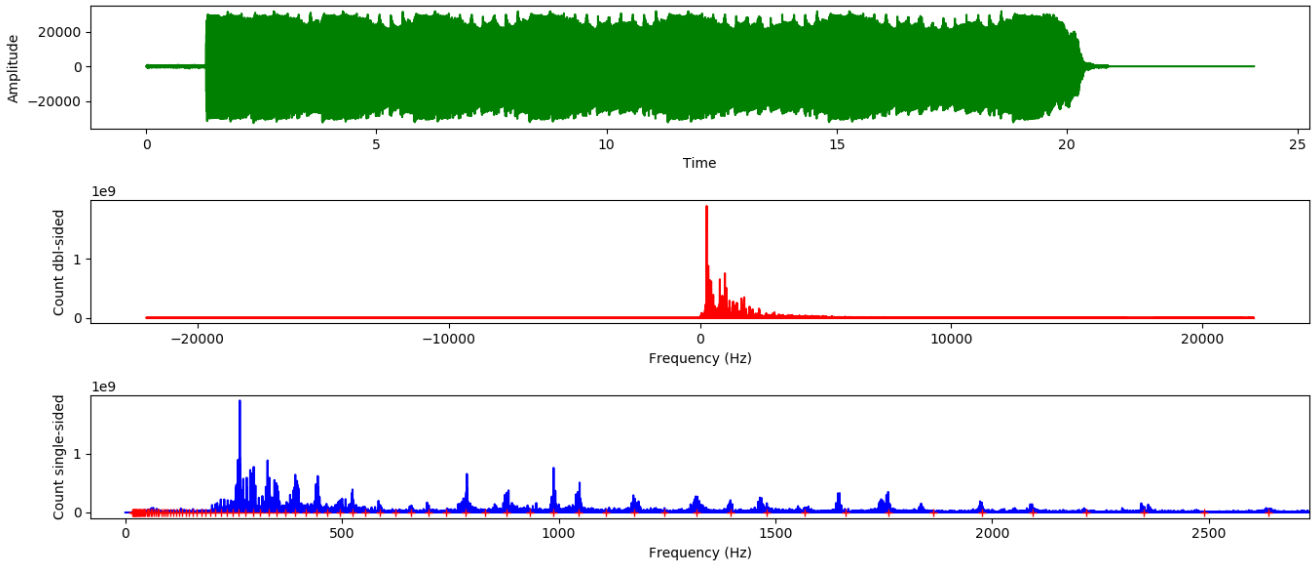


Figure 4.6 – The waveform and Fourier Transform results of the sample track with the first distortion pedal applied with high settings

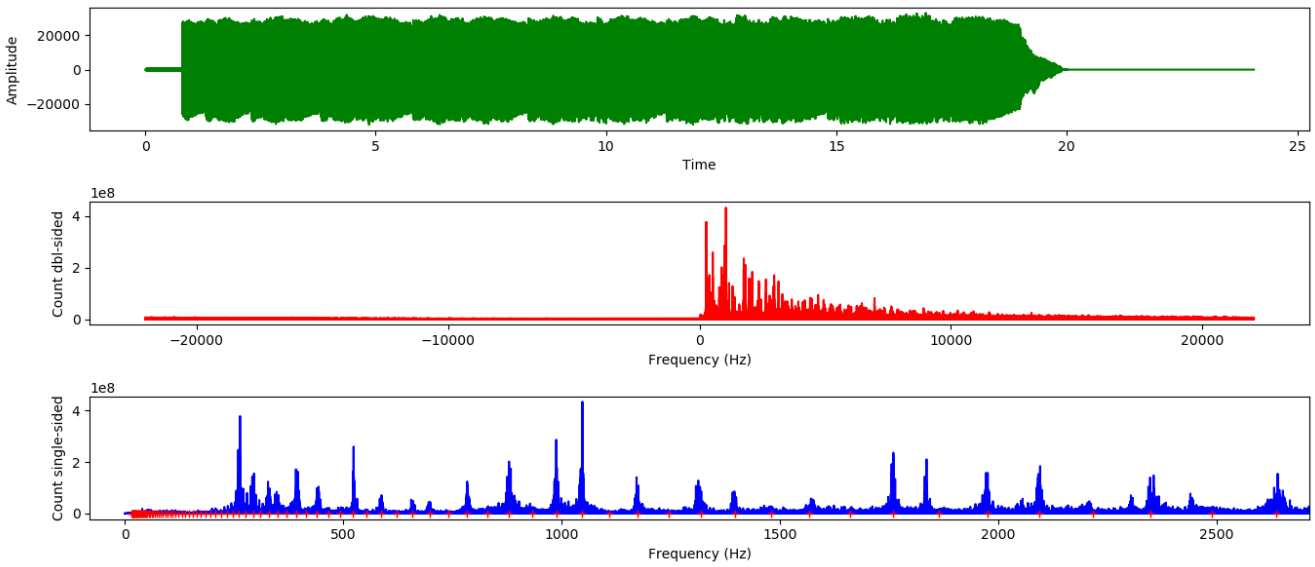


Figure 4.7 – The waveform and Fourier Transform results of the sample track with the second distortion pedal applied with high settings

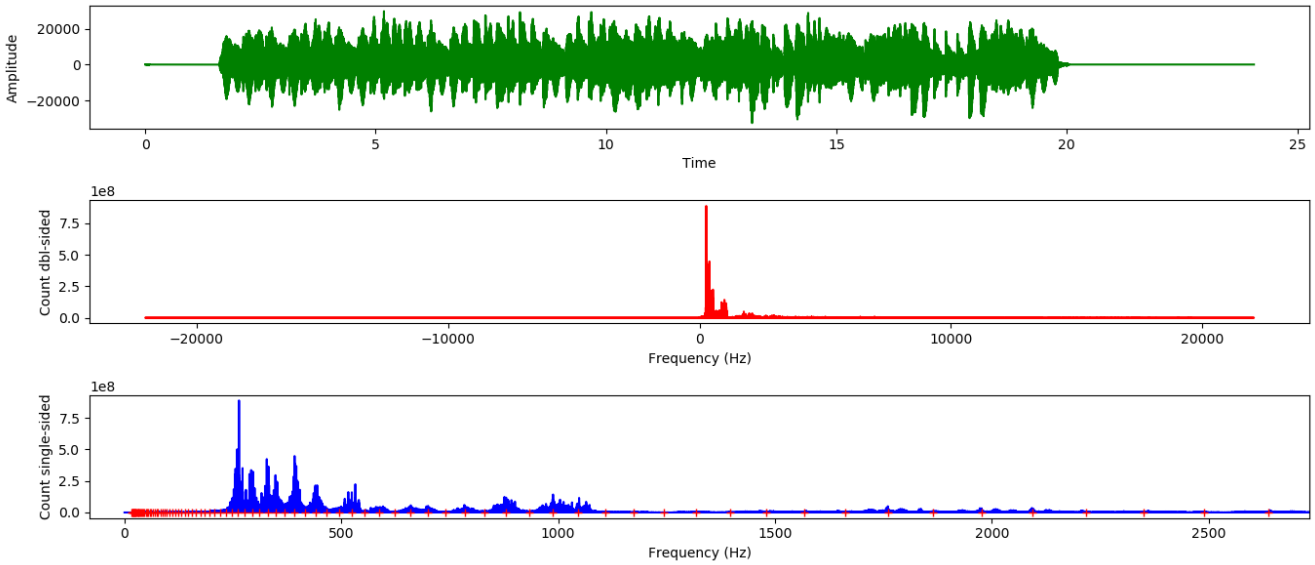


Figure 4.8 – The waveform and Fourier Transform results of the sample track with the chorus pedal applied

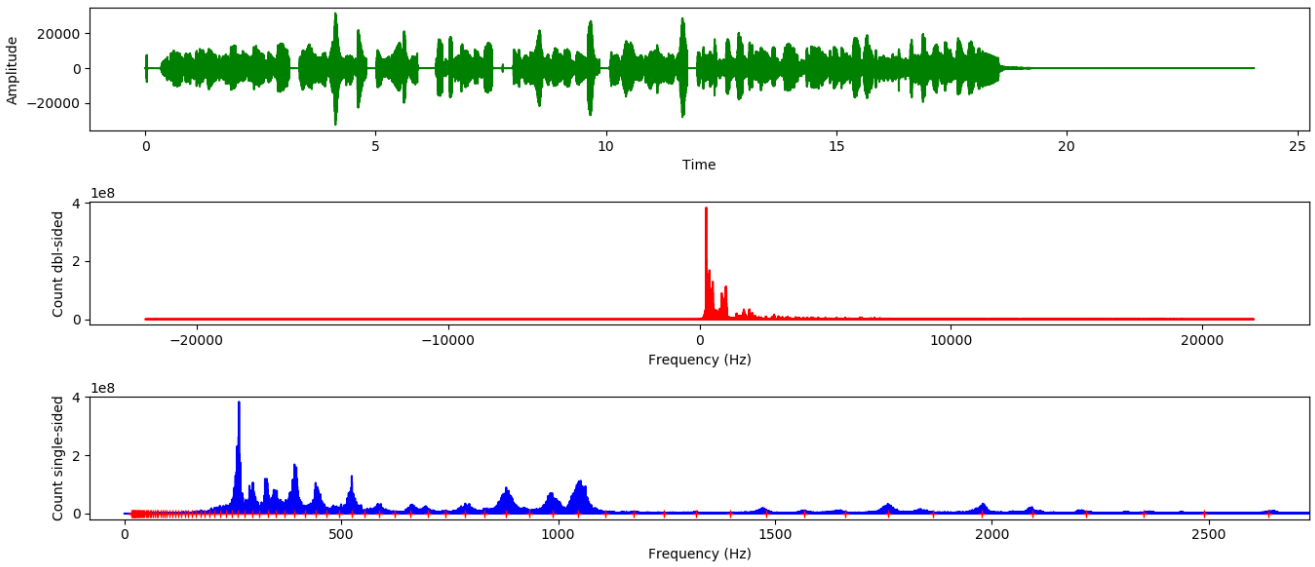


Figure 4.9 – The waveform and Fourier Transform results of the sample track with the phase shifter pedal applied

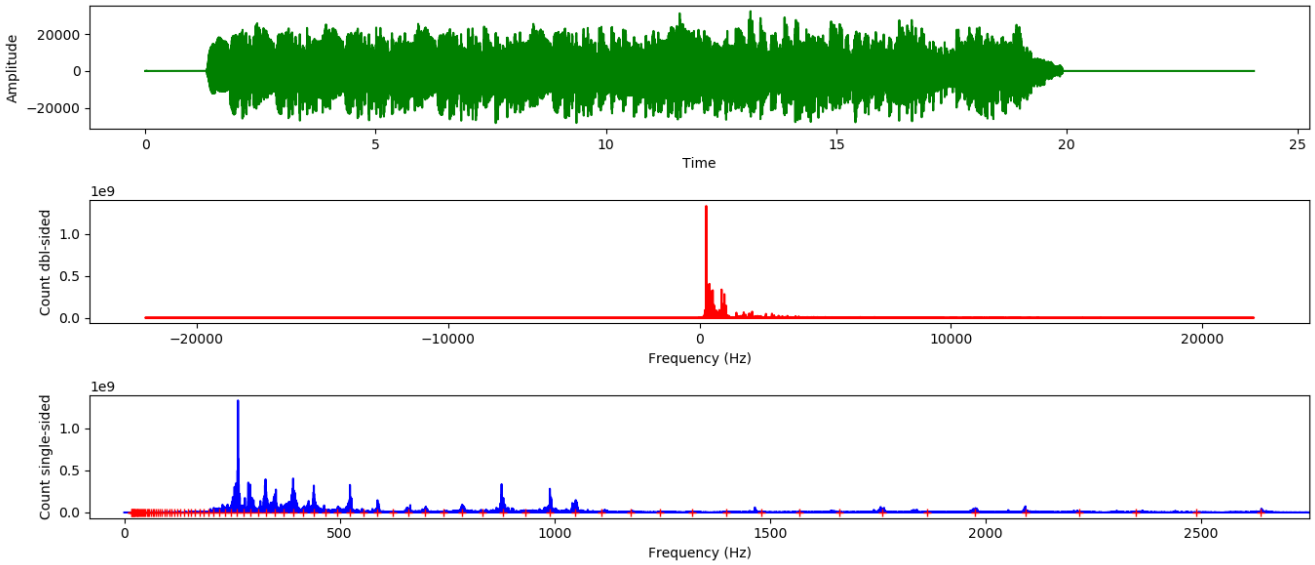


Figure 4.10 – The waveform and Fourier Transform results of the sample track with the delay pedal applied, while adjusting settings throughout the playback

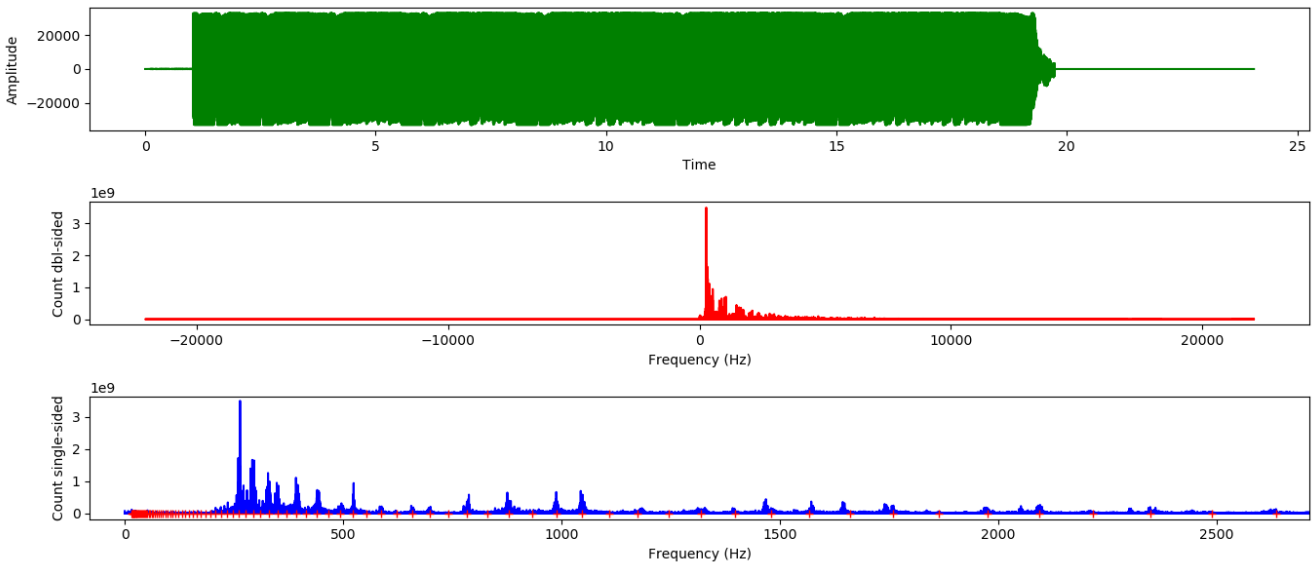


Figure 4.11 – The waveform and Fourier Transform results of the sample track with an overdrive pedal applied

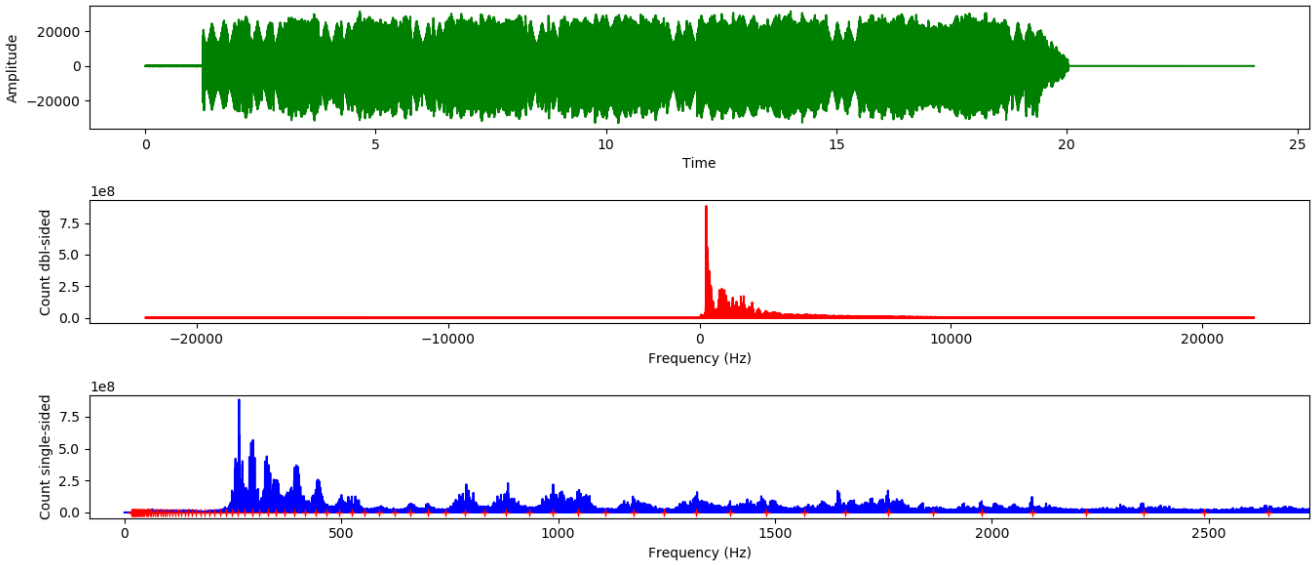


Figure 4.12 – The waveform and Fourier Transform results of the sample track with the first distortion pedal and chorus pedal simultaneously applied with high settings

Figure 4.3, which represents the results for the sample track with no effects applied, can be used as a baseline result to which the other results can be compared. Noteworthy features about these plots include the definition of the waveform – the end of each pattern of notes can be distinctly recognised by the peak that leads directly into a concave downwards slope. As well as this, in the Fourier Transform results, the peaks in the plot are extremely narrow, with essentially no frequencies above 1050Hz peaking to any significant extent.

Comparing these results to those of each of the tracks produced by applying the first of two distortion pedals with varying levels of strength, which can be seen in figures 4.4, 4.5 and 4.6 respectively, a similar phenomenon can be seen taking place among each set of results, with increasing levels of significance as the strength of the effect of the pedal increases. The waveform of the audio signal becomes much more amplified, creating much less visible definition visible in the graph. As regards the Fourier Transform results, there are a large number of new peaks that

are much more pronounced in the plot than there were in the same plot produced for the original sample track. A number of these occur above 1050Hz, which was the point at which there were little to no peaks in the plot of the results of the original sample track. The scale at which the graph is plotted also changes, with the upper limit of the scale increasing from the original in each plot. This is worth noting as peaks that appear to not change significantly when comparing the plots still increase in scale, which is a noteworthy change from the plot of the original sample track.

This increase in the number of individual peaks at various frequencies throughout the spectrum is a result of this particular distortion pedal boosting the harmonics of the signal when applying its effect. Harmonics in a signal are the frequencies that are multiples of a frequency that occurs in a signal. Part of the effect that this pedal appears to apply is that it increases the significance of these frequencies in the signal, which, in turn, increases the significance of frequencies that can be mapped to notes that produce relatively clean ratios with other notes in the signal, as they are multiples of the frequencies of the notes that comprise the signal. This essentially creates additional information about the key in which the song is written by providing more instances of notes within the key in which the song is composed that can be added to the tonal hierarchy generated for the track by the algorithm, which in turn leads to a stronger correlation coefficient being generated between the tonal hierarchy of the song and the C major key profile. As these features are intensified and the strength of the effect increases, it makes sense as to why the correlation coefficient increases along with this. It is worth noting, however, that the improvement in correlation coefficient from the original sample track to the track produced at low settings is greater than the improvement from the low settings track to the medium settings track, which is greater than the improvement from the medium settings track to the high settings track, indicating that this effect has diminishing returns as the intensity of the effect is increased.

Looking at the results of the plot of the second distortion pedal in Figure 4.7, it can clearly be seen that, despite this intended effect of this pedal being much the same as that of the previously tested distortion pedal, the way that this effect is implemented on the signal is not the same, resulting in many differences being noticeable in the resulting graphs. Peaks along the harmonics of the signal are much, much more significant, to the point where the frequency with the largest significance in the plot is not even the same as in the previous four sets of results. This indicates that the same type of audio effect can be achieved in a number of different ways, and the method used to carry out this effect depends on the manufacturer and model of the effect pedal in use. This particular pedal still has many of the effects previously discussed in relation to the first distortion pedal, such as the change in waveform shape and change in scale, although with this particular pedal this effect is intensified, as even the scale at which the graph is plotted is increased by an even larger factor, and the peaks that previously were not significant are taller, more defined and more pronounced to an extreme degree. As a result of this, a similar increase in correlation coefficient is seen for this track, although the improvement is not as great as that of the tracks with the effect of the first distortion pedal applied, as although this track has more distinct peaks throughout, some of these peaks are at frequencies that correspond to notes that should not be this significant in the key of C major, such as the note B, which does fit the key of C major, but according to the weights of the C major key profile, this note should be the least significant note of all the notes that belong to the key. The effect of this pedal appears to boost the frequency of this note past what it should in order to result in an even higher correlation coefficient between the frequency profile of this track and the C major key profile. That being said, the correlation coefficient is still a considerable amount higher than that calculated for the original track.

Next, the results of the effect of the application of the chorus pedal (Figure 4.8) can be observed. The primary feature of the effects of this pedal to notice is the substantial increase in the scale of the Fourier Transform plots. Despite not having as impactful a change to the overall layout of these graphs compared to the distortion pedals, the massive change that is made here has critical ramifications when it is considered that the base of the peaks appears to be widened to an extent by the effect of this pedal also. What this means is that there is an addition of many frequencies around the frequency values of notes that occur throughout the track. This means the effect of the pedal introduces frequencies that may not correspond to notes in large quantities. Additionally, in some cases these frequencies can correspond to notes, which can upset the tonal hierarchy

calculated for the track, thereby lowering the correlation coefficient that is generated for the estimate key.

The phase shifter pedal (Figure 4.9) appears to have a somewhat similar effect, albeit to a lesser extent. The scale of the Fourier Transform plots are increased again, but this does not occur with the same intensity as it does in the case of the chorus pedal. Likewise, the harmonics are not boosted to the same extent as the chorus pedal, however what is impacted to a greater extent is the contribution of additional frequencies that do not correspond to the frequencies of notes that occur in the original track. The pedal adds a 'wobbling' tone to the music, where the frequency of moves slightly above and below the original frequency of the signal, resulting in a smoother curve with less distinct peaks, as peaks have wider bases due to the contribution of these frequencies. As a result, there is an outcome in this case similar to that of the case of the chorus pedal track - this leads to a lower correlation coefficient being calculated between the frequency profile of the track and the C major key profile. This is because some frequencies from the original signal that, unchanged, would correspond to notes, are instead changed to a frequency slightly lower or higher than the original frequency, leading to less detected occurrences of the note to which the original frequency can be mapped, leading to a less representative tonal hierarchy being generated for the track.

When observing the plots for the delay pedal (Figure 4.10) it is important to note that the various settings dials on the pedal were being adjusted and changed throughout the recording of the track. This change in settings introduces a warping type of effect to the signal as the pedal adjusts to the change in settings. This is a phenomenon that was only observed when changing the settings of this particular pedal and was not the case with the other pedals used in this experiment. The warping effect on the signal introduced by using the pedal in this way results in the 'bending' of frequencies, which moves the frequency slightly up or down from the original frequency from the signal. As a result of this, many frequencies that, in the original signal, correspond to notes, are then adjusted to a different frequency by this effect. This leads to less frequencies that correspond to the correct notes occurring in the signal, as well as more frequencies that either do not correspond to notes at all, or correspond to notes that do not belong to the key. This results in a much lower correlation coefficient in this case, although the effect is not enough to turn the estimate into an incorrect one.

The results of the analysis of the overdrive pedal (Figure 4.11) show somewhat similar effects to the distortion pedal. The waveform appears to lose almost all definition, while the harmonics appear to see a boost in volume the Fourier Transform results plots. This increase however does not appear to be as extreme as in the case of the distortion pedals. That being said, the scale sees a rather significant increase compared to the results of the analysis of the original track. The fact that the overdrive pedal appears to have a similar, but less intense effect as the distortion pedals would serve as a reasonable explanation as to why the track effected by the overdrive pedal sees a similar, but not as large, improvement in correlation coefficient.

Finally, the track effected by both the first distortion pedal and the chorus pedal, both at high settings, attains the highest correlation coefficient of all tracks tested in this experiment. Due to the chorus pedal test attaining a lower correlation coefficient than the original sample track, it is surprising that combining this with the first distortion pedal at high settings attains a higher correlation coefficient than by using the distortion pedal individually. From looking at the results of the analysis of the track (Figure 4.12) features of both pedals can be seen – the wider base of peaks can be seen similarly to the results of the track effected by the chorus pedal (Figure 4.8), while the larger peaks, increased scale and boosted harmonics can be seen, similarly to the results of the tracks involving distortion pedal usage (Figure 4.6). What I believe is a key factor in the explanation of why this combination of pedals has attained a higher correlation coefficient is that the distortion pedal was the first to act on the signal of the sample track – the output of the distortion pedal was then directed into the input of the chorus pedal. This means that the effect of the chorus pedal was not based on the original input, rather it was based on the output of the distortion pedal. This means that the effects applied to the signal may not be identical to those applied to the original test track signal. It is possible that the greater amplification of the signal

caused by the distortion pedal lessened the negative effects applied by the chorus pedal, leading to a more strongly correlating tonal hierarchy for the track.

4.1.3 Mode Experiments

The estimates and correlation coefficients returned by the algorithm for the tracks created for this experiment, described in Section 3.4.3, can be seen in Table 4.4:

Mode:	Key Estimate:	Correlation Coefficient:
C Ionian	C Major	0.8793013691902161
D Dorian	D Minor	0.8295986652374268
E Phrygian	C Major	0.7157548666000366
F Lydian	F Major	0.7731274366378784
G Mixolydian	G Major	0.9105077981948853
A Aeolian	A Minor	0.9308153390884399
B Locrian	E Minor	0.6522256731987

Table 4.4 – The estimates and correlation coefficients returned for each track used in the modes experiment

These results clearly indicate that the accuracy of the estimate returned by the algorithm can be significantly impacted when a piece of music is written in a mode rather than a key - despite the fact that these modes use the exact same notes as the key of C major, the algorithm is unable to find the key that best indicates the notes used in four out of seven cases as a result of the change in tonal centre of the music.

In the F Lydian track, for instance, the note F features most prominently, while C occurs less than half the number of times F does. The relative volume of occurrences of the notes in relation to one another will not correlate closely to the C major key profile, as the tonic note of the C major key, which is C, occurs infrequently in comparison to the fourth note in the C major scale, which is F. The weights of the C major profile expect the note C to occur around 55.25% more than the F. In this composition, the F occurs around 128.6% more than C does. As a result of this, the correlation between the pitch class profile generated for the audio file does not correlate closely to the C major key profile at all, leading to the estimate predicting the key to be F major, where F is the tonic note. This key profile correlates the best as F occurs the most, and the F major scale contains six out of the seven notes in the C major scale. This means that despite the complete absence of B♭ as a note in sample track, and the presence of B as a note instead, the tonal hierarchy of the F Lydian composition still correlates to the F major key profile more strongly than it does to the C major key profile.

This is just one example of the four instances of incorrect estimate generation in this experiment. The only correct estimate generated in this experiment from a mode that does not share its tonal centre with an existing key is in the case of E Phrygian. However, even in this case, the correlation coefficient is drastically lower than that of either of the other two correct estimates.

The exception to the rule of non-Ionian modes negatively impacting estimate quality appears to be in one specific case - when the mode used is Aeolian. The reason for this is that the Aeolian mode maps directly to the scale of the relative minor to the major key in question. This means that a composition in A Aeolian will directly match the same composition in A minor, for instance. In this experiment, the composition using the A Aeolian mode was rightly estimated to be in the key of A minor, which makes sense, as it is identical, note-for-note, to the composition in A minor.

4.2 New Approach/Implementation Results

This section will detail the results of the performance evaluation of each of the implementations of the algorithm described in sections 3.2 and 3.3, under the performance metrics described in Section 3.5. Table 4.5 features the number or label allocated to each implementation of the algorithm that appears in the results included in this section.

Label:	Implementation:	Notes:
Baseline	Standard implementation	The baseline is the 5th implementation, chosen after comparing the first 6 implementations.
1st	Using the MusicExtractor function in Essentia	This implementation uses a different function to perform the same analysis.
2nd	KeyExtractor with no parameters	The baseline specifies to use Krumhansl-Schmuckler Key Profile values.
3rd	Loading tracks with the MonoLoader function	MusicExtractor takes a file path, while KeyExtractor uses files loaded into variables.
4th	Loading the track with the StereoLoader function	StereoLoader does not mix audio streams.
6th	Changed KeyExtractor weight type parameter	The KeyExtractor function includes a parameter for weighting frequency contribution. This tests the performance impact of changing from 'none'. All other implementations keep standard weighting.
7th	2:1 Compression & +12dB Amplification	
8th	1000Hz Low-Pass Filter	
9th	500Hz Low-Pass Filter	
10th	750Hz Low-Pass Filter	
11th	1500Hz Low-Pass Filter	
12th	2000Hz Low-Pass Filter	
13th	3000Hz High-Pass Filter	
14th	2500Hz High-Pass Filter	
15th	2000Hz High-Pass Filter	
16th	1500Hz High-Pass Filter	
17th	750Hz High-Pass Filter	
18th	500Hz High-Pass Filter	
19th	250Hz High-Pass Filter	
20th	125Hz High-Pass Filter	
21st	50Hz High-Pass Filter	
22nd	1000Hz High-Pass Filter	
23rd	50Hz Low-Pass Filter	
24th	125Hz Low-Pass Filter	
25th	250Hz Low-Pass Filter	
26th	2500Hz Low-Pass Filter	
27th	3000Hz Low-Pass Filter	
28th	+5dB Amplification	
29th	+7dB Amplification	
30th	+10dB Amplification	
31st	+12dB Amplification	
32nd	+15dB Amplification	
33rd	1.1:1 Compression	
34th	2.5:1 Compression	
35th	5:1 Compression	
36th	7:1 Compression	
37th	1.1:1 Compression and +10dB Amplification	
38th	96 Beat Subsampling	
39th	60 Beat Subsampling	
40th	120 Beat Subsampling	
41st	144 Beat Subsampling	
42nd	Beat Range Subsampling: Setup 1	Estimate for each iteration selected by evaluating average correlation coefficient per key per estimate
43rd	Beat Range Subsampling: Setup 2	Estimate for each iteration selected by evaluating highest total correlation coefficient per key
44th	Beat Range Subsampling: Setup 2 with 50Hz-750Hz Band-Pass Filter	Band Pass Filter implemented in Audacity
45th	50Hz-750Hz Band-Pass Filter	Band-Pass Filter implemented in Audacity
46th	Beat Range Subsampling: Setup 2 with 50Hz-750Hz Band-Pass Filter	Band-Pass Filter implemented in code
47th	Beat Range Subsampling: Setup 2 with 50Hz-750Hz Band-Pass Filter	Band-Pass Filter implemented in code and applied multiple times
48th	Tuning Frequency Adjustments, 428Hz-452Hz	Each iteration increments tuning frequency by 1Hz
49th	Tuning Frequency Adjustments, 432Hz-448Hz	Each iteration increments tuning frequency by 1Hz
50th	Tuning Frequency Adjustments, 432Hz-448Hz	Each iteration increments tuning frequency by .5Hz

Table 4.5 – Labels for each tested implementation of the algorithm

4.2.1 Total Correct Estimates

The total correct estimates metric performance for each implementation can be seen in Figure 4.13

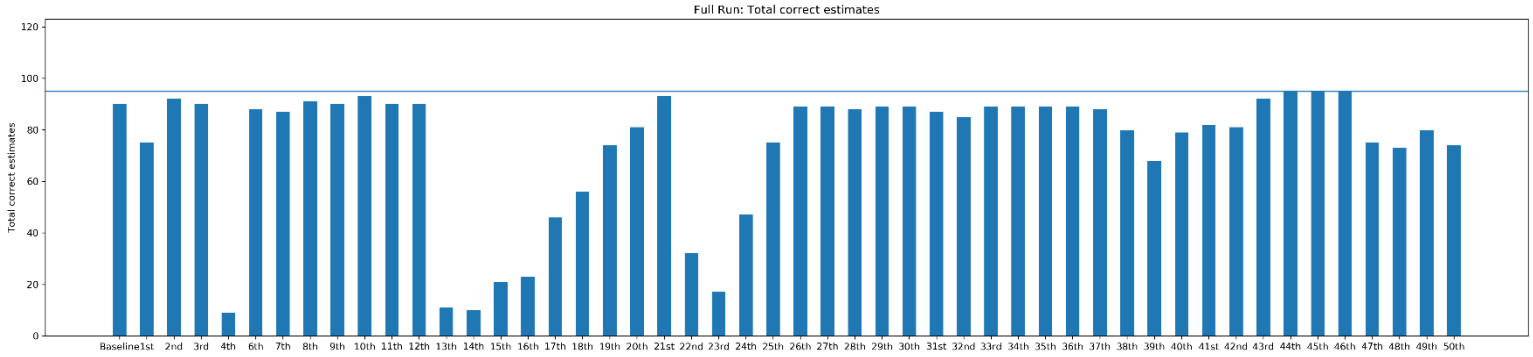


Figure 4.13 – Results for total correct estimates for all implementations over the entire dataset

Clearly a wide range of performance levels can be seen within the sets of results, with a number of the implementations outperforming the baseline. The blue line along the top part of the chart indicates the largest value out of all results in the chart. Three implementations can be seen to reach this value – all of which involve the usage of a band pass filter. This shows incredibly promising signs for the usage of filters to remove noise from the audio. A key-specific example of this can be seen in the key of D major/B minor, in which the best performing sets of results all involve usage of high-pass, low-pass or band-pass filter. These results can be seen in Figure 4.14:

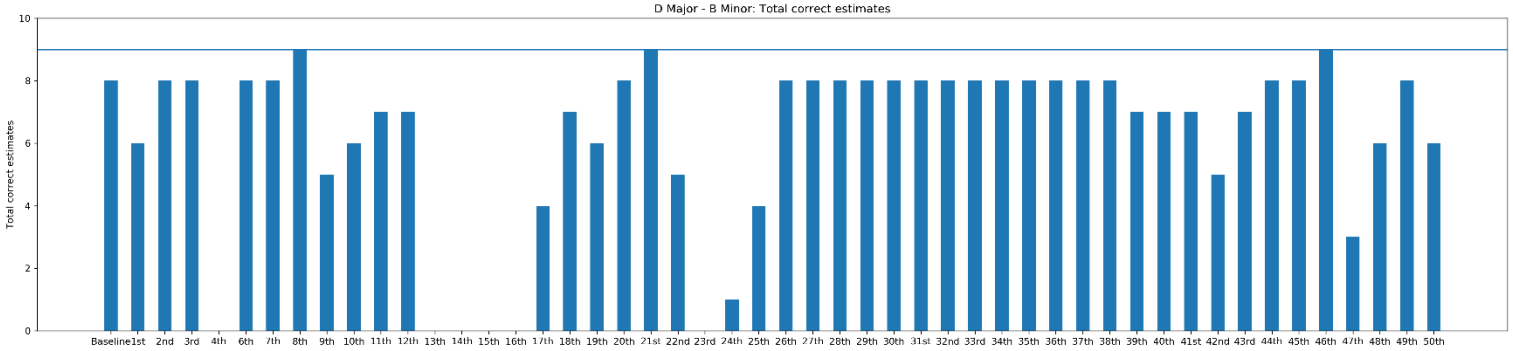


Figure 4.14 – Results for total correct estimates for all implementations in the key of D Major/B Minor

Overall, these results indicate strong performance from both the implementations surrounding beat range subsampling and usage of filters, as well as a combination of both. This gives strength to the reasoning made regarding the creation of these implementations, such as how the removal of audio from outside the general musical frequency range and the segmentation of tracks could each provide a unique benefit to this process. However, the benefit provided by audio filtering depends on the parameters chosen for the filter, as indicated by the abysmal results attained by sets of results such as the 13th, which uses a high-pass filter with a threshold of 3000Hz. This eliminates a majority of the frequencies from the original track, making accurate key estimation essentially impossible. This indicates that the frequencies at or above around 3000Hz are not particularly helpful in accurate algorithmic key estimation.

4.2.2 Total Note Matches

Results for each implementation under the Total Note Matches metric can be seen in Figure 4.15.

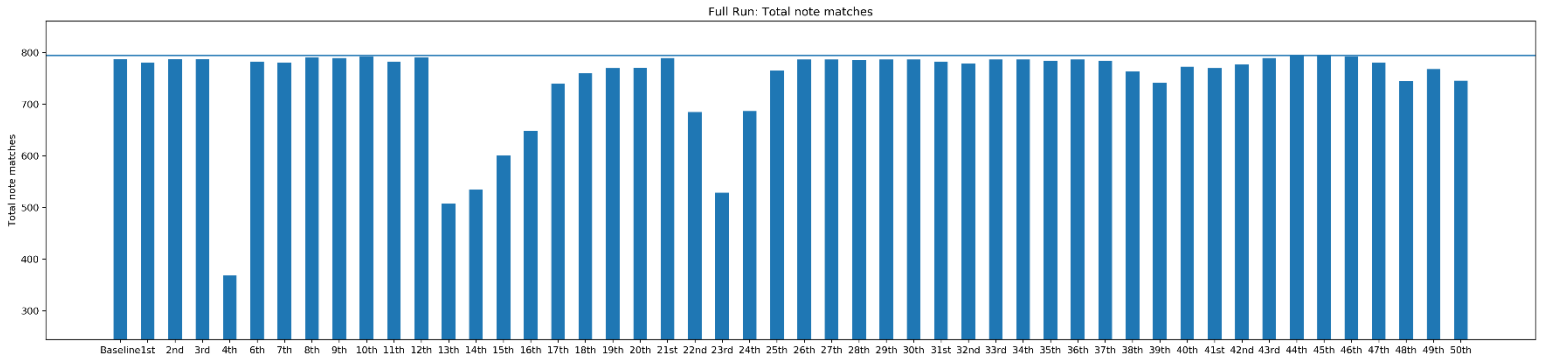


Figure 4.15 – Results for total note matches for every implementation over the entire dataset

Clearly the differences among the top performers under this metric are rather marginal according to this chart. For further investigation, it is worthwhile to visualise only the top performing results in order to inspect the differences between them. The graph of the fifteen best performing sets of results under this metric can be seen in Figure 4.16.

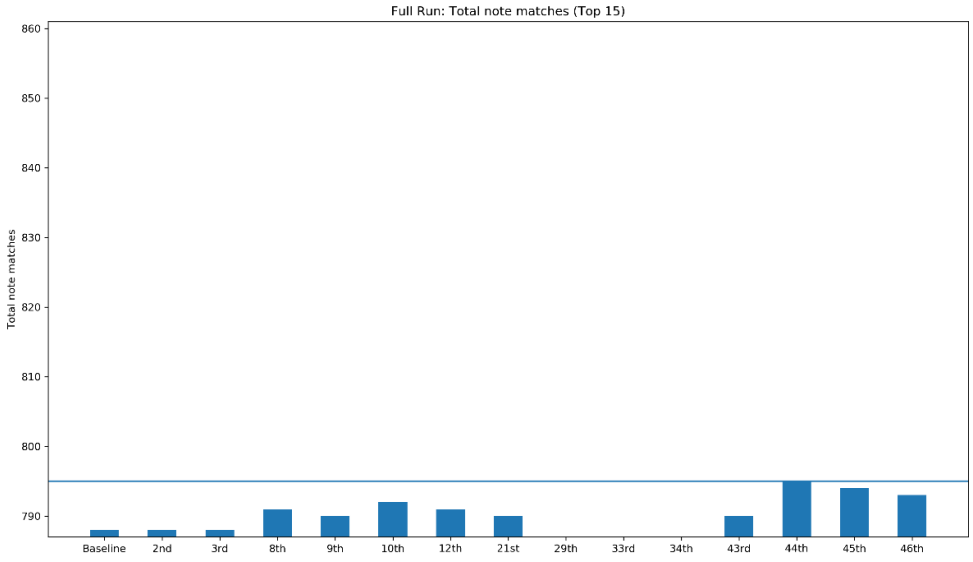


Figure 4.16 – Top fifteen sets of results based on total note matches over the entire dataset

This graph more clearly indicates the differences between the top performing implementations. It is worth noting that a single additional matching note can translate into an additional correct estimate in some cases. Here we can see strong performance from the Beat Range Subsampling and filtering methods yet again, with the implementations combining these methods seeing the highest performance levels, outperforming the baseline again.

Many of the other implementations, such as those using compression and/or amplification, appear to match the results or slightly underperform when compared to the baseline. Those that underperform most heavily appear to be the implementations involving poor filter parameter choice, such as the 13th set of results, discussed in the previous section. Another implementation that underperforms when compared to the baseline - albeit to a much lesser extent than the 13th set of results – is the implementations involving adjustment of tuning frequencies for the algorithm. As discussed in Section 3.3.8, this implementation was developed in order to evaluate if adjusting this parameter would assist the algorithm in attaining improved performance by adjusting the tuning frequency for the algorithm. This appears to unfortunately not be the case. An explanation

for this is that in some cases in which elements of a song are out of tune, there may be one or more elements of the song that remain in tune, meaning that changing the tuning frequency may consider these elements no longer in tune in order to compensate for a single element of a song that is out of tune. An additional obstacle for accurate key estimation is noise and drum frequencies in the audio. There is a strong possibility that in cases where these are present, changing the tuning frequency could begin to map the frequencies at which these effects occur to notes in the music, drawing the tonal hierarchy that will be generated further from the key profile of the true key of the song.

4.2.3 Weight-Based Key Closeness Scoring

Performance results for each implementation under this metric can be seen in Figure 4.17.

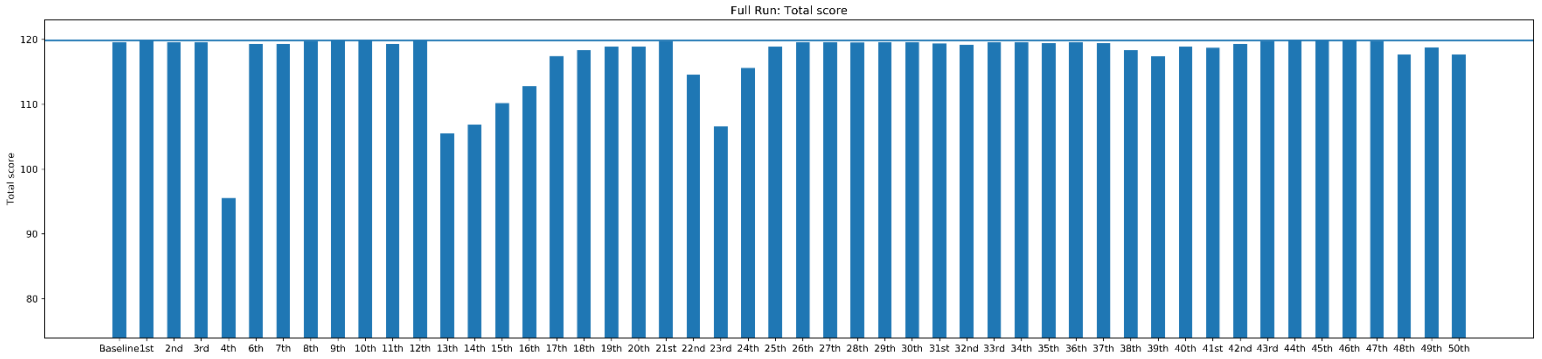


Figure 4.17 – Results for weight-based key closeness scoring for all implementations over the entire dataset

When viewing at this scale, the results of this metric are rather similar to that of the total note matches, which makes sense as both are based either somewhat or entirely on the notes that comprise the estimate key when compared to the categorised key of the song in question. What this metric provides is with a more minute level of detail than the total note matches metric, as outlined in Section 3.5.3. While total note matches is a useful metric in helping generally categorise results into rough performance groups, such as strong, average and weak, this metric can be considerably more valuable than total note matches when closely comparing performance among many implementations that attain relatively similar levels of performance. This can be seen be inspecting the graph of the top fifteen sets of results under this metric in Figure 4.18.

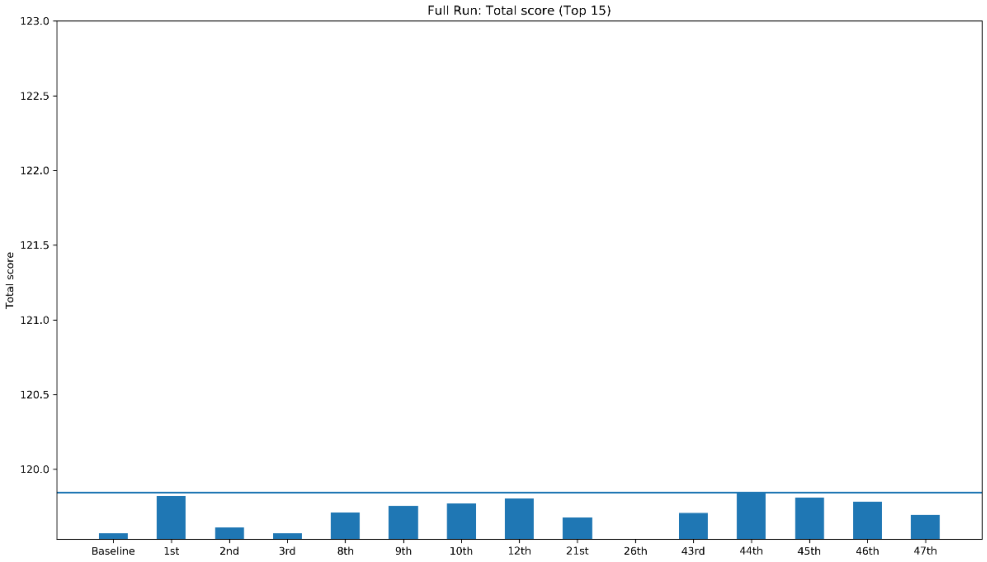


Figure 4.18 – Top fifteen sets of results based on weight-based key closeness score over the entire dataset

Comparing this chart to its total note matches counterpart (Figure 4.16) it can be observed that a number of the implementations that are included in the top fifteen sets of results are changed. In addition to this, the positions of a number of these implementations relative to one another changes. For example, under the total note matches metric, the 12th implementation (2000Hz low-pass filter) performed worse than the 10th set of results (750Hz low-pass filter), while under this scoring metric, the 12th implementation outperforms the 10th. This indicates that, while the two implementations are rather close in performance level, they each have different strengths. For instance, the 10th implementation outperforms the 12th in terms of total correct estimates, however it could be said that by attaining a better score over the entire dataset, the estimates of the 12th implementation, on average, are more close to the correct key than those of the 10th estimate, despite having less overall correct estimates.

Additionally, it should be noted that despite the changes in ranking position for many sets of results, the best performing implementation under this metric involves both the usage of a band-pass filter as well as beat range subsampling once again.

4.2.4 Average Score of Incorrect Estimates

The value of this metric appears to be highly based on context, which can be seen by analysing the results for each implementation under this metric (Figure 4.19)

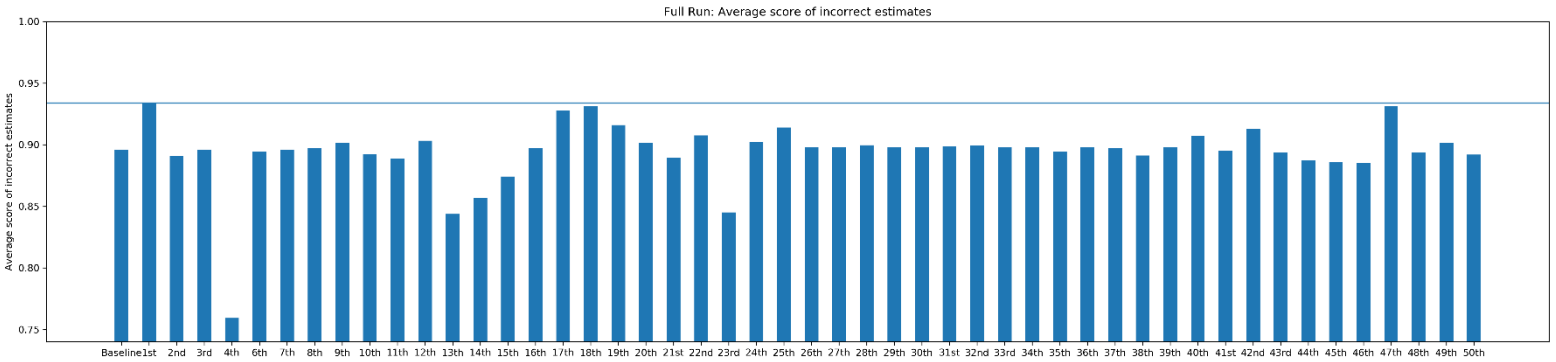


Figure 4.19 – Results for average score of incorrect estimates for all implementations over the entire dataset

This metric, while still meaningful and valuable to investigation, is not useful on its own, as in order to make many observations regarding the performance level of an implementation under this metric, additional context is required.

For example, the 18th implementation (500Hz high-pass filter) is one of the top performers under this metric, outperforming the baseline by a significant margin. However, when looking at the previously discussed metrics, this implementation clearly underperforms compared to the baseline, and in addition, there are only a small number of implementations (generally less than fifteen) that perform worse than it for any individual metric. The reason why this implementation performs well under this metric is because it tends to produce estimates that are close to the correct key, but incorrect. This is not ideal as the primary goal of each implementation of the algorithm is in order to improve the accuracy of the estimates produced, by improving the number of correct estimates, and making the incorrect estimates as close to the correct key as possible. Similarly, results without context can be misleading in the case of the 44th implementation (Beat Range Subsampling with Band-Pass Filter) where performance under this metric can be considered average, if not poor, which without context would appear as though this implementation provides no improvement over the baseline. However, with the knowledge that this implementation is the top performer under each of the previously discussed metrics, it is clear that the reason why this implementation performs poorly under this metric is because a majority of the incorrect estimates generated are for particularly difficult songs, in that there is a feature of the track that makes accurate key estimation extremely difficult in a way that is not accounted for by the implementation, such as inaccurate tuning. These tracks may obtain low-scoring estimates, however as these are

some of the only incorrect estimates made, the low score generated for the estimates negatively impact the average score of incorrect estimates considerably.

When observing results under this metric on a per-key basis, the need for context is elevated further, as can be seen in the results contained within Figure 4.20.

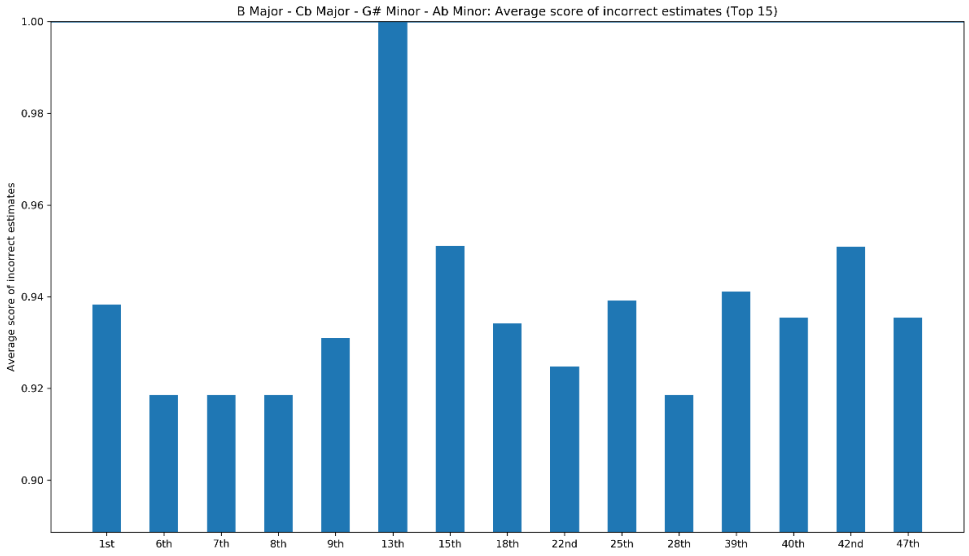


Figure 4.20 – Top fifteen sets of results based on average score of incorrect estimates in the key of (B/C♭) Major/(G♯/A♭) Minor

In these results, excellent performance from the 13th implementation can be observed. Attaining an average score of 1 for incorrect estimates indicates that every estimate in this key is correct. However, considering the context of the performance of the 13th implementation over the entire dataset, which can be seen in Figure 4.19, it is clear that the performance in this key is an extreme outlier. This implementation performs well in this key because it produces the same estimate for a majority of songs, which happens to be this key. This means performance for this implementation in this key appears perfect, but this is purely coincidental.

For these reasons, results from this Section can be considered valuable when the context of the wider performance of each implementation is taken into account. The ideal implementation of this algorithm would be correct in every case. However, this is unlikely to be possible, so the ideal alternative is to have an implementation that is correct as much as possible, which produces estimate keys as close to the true key as possible when incorrect. This is why this metric is still useful despite the confusion that can come from a lack of context, and why it merits observation.

4.2.5 Relative Performance Scoring

Results for performance under this metric can be seen in Figure 4.21.

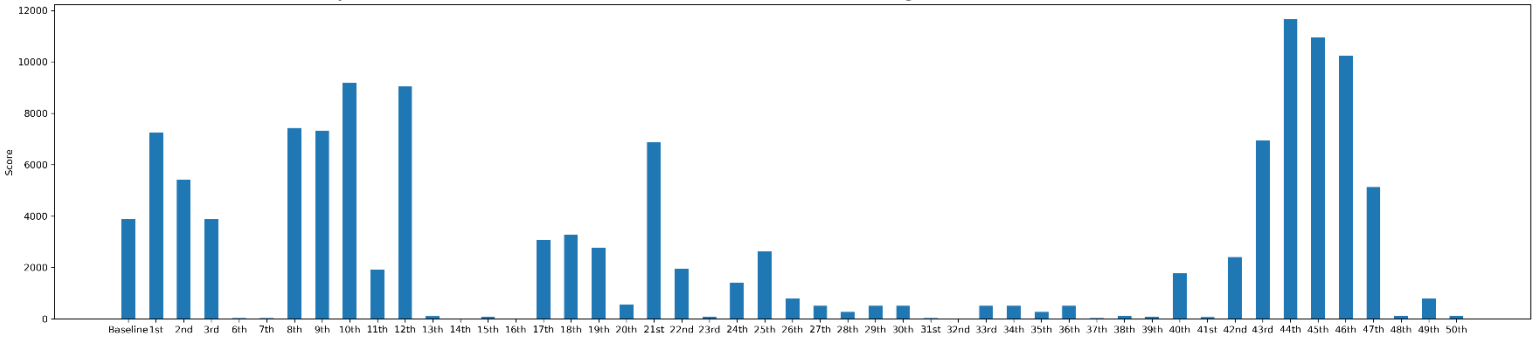


Figure 4.21 - Results for relative performance score for all implementations over the entire dataset

This metric has a tendency to magnify marginal differences in performance between implementations, as a result of how the scoring system is based purely upon how many other implementations are outperformed within the best performing 30% of implementations over each metric. This means that an implementation that outperforms another by an extremely narrow margin on a number of occasions will have this small difference magnified to a seemingly much larger extent when evaluating the differences between entries in this chart. The intention of this metric, as well as this chart, is not to display the extent to which each implementation outperforms one another, but rather to display, at a glance, which implementations attain noteworthy performance across the dataset, and which implementations are not worth investigating further in their current form.

With this in mind, it becomes clear that there are a number of approaches to new implementations that perform rather well and could merit further testing and investigation. These implementations all involve the usage of filters and/or beat subsampling. That being said, there are numerous implementations that use filters or beat subsampling that do not perform well. Namely, these are:

- Low-pass filters with thresholds set at 250Hz or lower or 1500Hz, 2500Hz or 3000Hz
- High pass filters with a threshold above 50Hz
- Standard beat subsampling methods (i.e., the method outlined in Section 3.3.6 that uses a static value for segment size)
- Beat range subsampling Setup 1 (Selects estimate from each iteration via average correlation coefficient)

These results, despite not performing well, provide useful insight into the areas in which the stronger-performing implementations gain their performance advantages over these implementations that perform poorly in comparison. For example, the methods involving high-pass filters that do not perform well, when analysed as a group rather than individually, indicate that once the threshold parameter increases pasta specific point, performance begins to degrade rapidly.

Other results that do not perform well include all implementations involving amplification, compression and/or adjustment of tuning frequency.

Similarly, these results, despite not performing well with the chosen parameters, indicate areas that are likely to not be as beneficial to investigate further as others used in the investigation. In addition, the results involving amplification and compression of audio performing worse overall compared to the baseline, in theory indicates that these processes can be detrimental to the likelihood of accurate key estimation to be performed on an audio signal to which these processes have been applied. Additionally, this highlights potential flaws in the approach of the implementations involving tuning frequency adjustment, as described in Section 4.2.2.

The best-performing implementation under this metric, as well as all others previously discussed in this chapter, is the 44th – using beat range subsampling, selecting estimates for each iteration by evaluating the highest total correlation coefficient per key, while also using a 50Hz - 750Hz band-pass implemented in Audacity.

4.2.6 Tone-Specific Misses

From my analysis of this metric, I have been unable to come to a single concrete conclusion regarding similarity or consistency among the missed tones from the true key of a track that are not included in incorrect estimate keys. My analysis consisted of inspecting the tone-specific miss counts for both the baseline and best-performing implementation of the algorithm and comparing the tones missed both on a per-key basis and overall. I discovered that the ‘fa’ tone, that being that fourth note of the key, was missed the most in incorrect estimates from both implementations investigated, as can be seen in Table 4.6.

Tone-specific Miss Counts		
Tone:	**Baseline:**	**Best-Performing:**
‘Do’	15	13
‘Re’	5	4
‘Mi’	2	4
‘Fa’	24	19
‘So’	13	12
‘La’	5	6
‘Ti’	9	8

Table 4.6 – Tone-specific miss counts for the baseline and best-performing implementations over the entire dataset

There are a number of reasons why this may be the case – one possibility is that the dataset is skewed, meaning it happens to include a number of songs that make this tone particularly difficult to recognise, in that there may be where the relevant note is played or sung out of tune or not played often, for example. Another possibility is that the ‘fa’ weighting in the key profiles is set at a value too high or too low, leading tonal hierarchies generated for songs to not correlate closely enough to the key profile of the true key, correlating to that of a different key as a result instead.

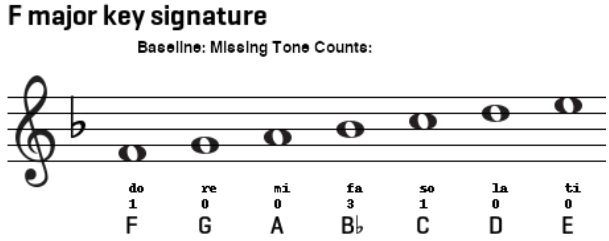


Figure 4.22 – An example of a per-tone miss count generated for the baseline implementation

The example in Figure 4.22 shows a case in which the ‘fa’ tone is the only tone that is missed to any considerable extent, although it is difficult to come to a definitive assertion as to why this is the case. In the best-performing implementation of the algorithm, this tone is missed less times than in the baseline implementation, but the occurrences of misses of a number of other tones, such as ‘mi’, increases. This could potentially serve as an indication that a number of the tracks that tend to be more difficult to generate correct key estimates for tend to have particularly difficult-to-detect ‘fa’ tones, which would mean that the best-performing implementation deals with this particular issue in a much better way than the baseline implementation.

Overall, what can be said with absolute certainty is that this metric, above all others discussed within this chapter, would benefit the most from expanding the dataset to include a larger number of tracks within each key.

4.2.7 Expanded Dataset Testing

Key	Baseline	Best-Performing
A Major / (F♯/G♭) Minor	10/12	11/12
(A♯/B♭) Major / G Minor	7/12	7/12
(B/C♭) Major / (G♯/A♭) Minor	9/12	9/12
C Major / A Minor	8/13	9/13
(C♯/D♭) Major / (A♯/B♭) Minor	6/12	10/12
D Major / B Minor	10/12	10/12
(D♯/E♭) Major / C minor	11/12	10/12
E Major / (C♯/D♭) Minor	10/12	10/12
F Major / D Minor	9/12	11/12
(F♯/G♭) Major / (D♯/E♭) Minor	10/13	11/13
G Major / E Minor	10/13	9/13
(G♯/A♭) Major / F Minor	10/12	10/12
Total:	110/147	117/147

Table 4.7 – Total correct estimates for the baseline and best-performing implementation over an expanded dataset

The results from Table 4.7 compare the performance between the baseline and best implementations of the algorithm over an expanded dataset. This process involved adding the extra two tracks per key, which were separated from the dataset as described in Section 3.1.3. The performance is compared based on total correct estimates as there are fewer sets of results to compare. These results show that, even with an extended dataset, the best-performing implementation of the algorithm still outperforms the baseline, obtaining around 6.4% more correct estimates. This improvement is distributed across most keys, outperforming the baseline by as many as four estimates in some keys, while only underperforming in two keys, in each case by a single estimate. These results serve as further indication that the best-performing implementation from this investigation is worthy of further investigation. In addition, further details regarding tone misses were collected, resulting in the updated table which can be seen in Table 4.8.

Tone-specific Miss Counts		
Tone:	**Baseline:**	**Best-Performing:**
'Do'	17 (+2)	15 (+2)
'Re'	5 (+0)	4 (+0)
'Mi'	2 (+0)	4 (+0)
'Fa'	28 (+4)	21 (+2)
'So'	13 (+0)	13 (+1)
'La'	5 (+0)	6 (+0)
'Ti'	9 (+0)	8 (+0)

Table 4.8 – Updated tone-specific miss counts for baseline and best-performing implementations over the expanded dataset (changes are in brackets)

The results of this are much the same. There is an increase in misses for 'Do', 'Fa', and 'So', however the additional 'So' miss only occurs in the best-performing implementation. The degree to which 'Do' misses increase is identical across both implementations, however the degree to which the 'Fa' misses increase is double that of the baseline implementation. The particularly large difference in 'Fa' misses could be an indication of the best-performing implementation of the algorithm being capable of dealing with the individual issue that leads to misrecognition of the 'Fa' note of the key during estimate generation. However, given that such a large number of these particular tones are missed, it is more likely that, when a random incorrect estimate is changed to a correct one, this will lead to a correct 'Fa' tone being discovered in place of the incorrect one. As a result, it should not be considered impossible that this noteworthy difference in tone misses for this particular tone is coincidental rather than the result of the best-performing implementation introducing a better method of obtaining the correct 'fa' tone specifically, as its improvements can be considered more comprehensive across all tones that comprise a key.

4.2.8 Bonus Tracks Performance

As outlined in Section 3.1.2, during song categorisation, if a song could not confidently be categorised into a key, it would be added to an additional dataset along with an explanation of why the song was added. Estimates were gathered for these tracks in order to investigate how the algorithm performs on more obscure and difficult tracks such as these. The results for these tracks can be seen in Table 4.9.

Track	Key/Unique Feature	Baseline Estimate
Elena Siegman - Pareidolia	Built on the C Diminished scale	C♯ Minor
Thin Lizzy - The Boys are Back in Town	A♭ major, bridge part out of key, but no key change	A♭ Major
Alice in Chains - Check My Brain	E♭ major with bending of notes, part towards end out of key	F Minor
Slayer - Necrophiliac	In F minor but extensive use of accidentals	E♭ Major
Lynyrd Skynyrd - Sweet Home Alabama	D major/G major/D Mixolydian? Key signature debated	G Major
Mountain - Mississippi Queen	E Major, guitar tuning is slightly off, heavy use of accidentals	E Major
Jimi Hendrix - Purple Haze	Built on the E major blues scale, includes use of accidentals	E Minor
Boston - More than a Feeling	Verses are in D Major, Chorus is in G Major	G major
Lamb of God - Laid to Rest	D Minor, extensive use of accidentals and growling in vocals	D Major
System of a Down - Suite Pee	Built on the F Harmonic Minor scale	C Major
Metallica - The Unforgiven	A Dorian	A Minor
Van Halen - Eruption	Key changes (A♭ to E♭) & copies a classical song for a portion	A♭ Major
Metallica - One	D Major, changes to E minor midway through, a portion of non-musical audio at the beginning of the song	E Minor
Rage Against the Machine - Killing in the Name of	D Minor with heavy use of accidentals	D Major
The Rolling Stones - Sympathy for the Devil	E Mixolydian	A Minor
Living Colour - Cult of Personality (from ~:07)	G Dorian	C Major
A Day to Remember - Sticks and Bricks	D♭ Mixolydian	F Minor

Table 4.9 – Estimates for bonus tracks from baseline implementation

These results indicate that there is a certain margin to which the algorithm can manage out-of-key notes being involved in a song, as well as slightly imperfect tuning. Songs such as ‘The Boys are Back in Town’ and ‘Mississippi Queen’ have their keys correctly estimated, despite the presence of out-of-key notes in portions. Meanwhile, ‘Laid to Rest’ does not obtain a correct key estimate, as the extent to which accidentals are used is simply too much, leading to an imbalanced tonal hierarchy for the song. Additionally, the results obtained here give further strength to the theory that music composed in modes rather than keys will generally struggle to obtain estimates that match the key that the mode in question shares its notes with. Likewise, music composed using diminished scales, harmonic scales or blues scales result in similar inaccurate estimates being generated

5. Chapter Five - Conclusion and Future Work

I believe that there are a number of rather valuable findings that have been obtained throughout the experimentation and testing performed throughout this project. Investigation into algorithmic performance over a considerably large dataset of music has provided insight into numerous possible explanations for generation of incorrect estimates and features that are generally detrimental to the likelihood of accurate key estimation for a song. These can be classified under the following headings:

- Non-key musical foundation usage:
  This describes all possible sets of notes from which a song can be composed that are not keys, for example, modes, diminished scales, and blues scales. These may have identical notes to certain keys, but can have a differing tonal centre, making estimating the key that contains the same notes as the actual set used to compose the song much more difficult than in the case where a key is used instead, as evidenced by the experiment results in Section 4.1.3. In some cases, these sets may not match any set of notes that comprise a key, meaning that accurate categorisation under the label of a key is simply impossible.

- Non-musical sound, noise, imperfect tuning, and/or presence of drums:
  This specifies features that either contribute frequencies generated by melodic instruments that do not map to musical notes when they should, or frequencies generated by non-melodic instruments that map to musical notes when they should not. In the case of either or both of these occurrences, the tonal hierarchy generated for the song will not be as accurate a representation of the music from a tonal perspective as it would otherwise. The extent to which the music is misrepresented will be based on the extent to which this feature or combination of features are present within the wider context of the music as a whole. If non-musical sound such as speech comprises a large portion of a song, contributing as many or more sounds to the audio signal for the song than other melodic portions of the music, the negative impact of the presence of this feature is magnified compared to the case in which the feature was only present for a brief period of the song compared to the instruments and musical features intended to contribute melody and harmony to the music.

- Usage of specific effects pedals and/or specific usage of certain effects pedals:
  As indicated by the experiments surrounding the usage of effects pedals outlined in Section 3.4.2, the results of which can be seen in Section 4.1.2, the impact that effects pedals can have on the accuracy and reliability of an estimate produced for a signal on which the effect has been applied is noteworthy. In some cases, this strengthened the correlation coefficient between the frequency profile generated for the test track and the key profile of the key in which it was composed. However, there exist cases in which the application of an effects pedal was detrimental to performance. This may be due to the effect applied by the pedal in general, or as a result of the specific settings used that impacted the effect that was applied in the test case, producing the output signal that obtained the lower-quality result. Nevertheless, this demonstrates that usage of effects pedals can, in some cases, be detrimental to algorithmic performance.

- Intensive use of accidentals:
  This is the usage of notes that do not belong to the key in which the song is written. As the notes do not belong to the key of the song, their presence can create imbalance in the tonal hierarchy generated for the song, resulting in a lower correlation coefficient being generated between the frequency profile generated for the song and the key profile of the true key of the song. The more often accidentals occur within the song, the further this negative impact is intensified. This can eventually lead to inaccurate key estimation if the extent to which accidentals occur is excessive.

- Lack of information:
  This is the case when a song provides so little information regarding the key of the song, either via being extremely short in length, or extremely repetitive. If the song does not feature enough samples of notes from within the key in which it is written, it can be extremely difficult to correctly generate a key estimate for the song, as the notes included in the song may fit multiple keys.

In addition to information regarding failure cases, this project has led to the creation of an alternate implementation of the algorithm that has shown promising results when implemented over the dataset gathered for this project. This is the implementation that uses a 50Hz to 750Hz Band-Pass filter before using the Beat Range Subsampling method, as described in sections 3.3.3 and 3.3.7 respectively. The specific method for selection of estimates within each iteration of the beat subsampling method in this implementation is based on the total correlation coefficient for each key estimated within that particular iteration of the beat subsampling method. The final estimate is based on average correlation coefficient of these methods returned from each iteration.

The results of this implementation when compared to the baseline directly indicate that performance improvements are possible for this algorithm, and a knowledge of the features that can create failure cases is hugely important in the development of new, more accurate implementations of the algorithm. Features such as noise being filtered out by usage of the band-pass filter, as well as non-musical portions of audio being accounted for through the use of beat range subsampling can lead to the negative impact of these features being diminished to an extent, allowing for more correct estimates to be made.

Furthermore, there are additional implementations of the algorithm that were tested throughout the duration of this project that can be used to make tentative theories surrounding the success of the implementation, or the lack thereof. For instance, the implementations involving the use of amplification and/or compression performed rather poorly in comparison to the baseline. With this knowledge, it could be inferred that the presence of clipping in audio introduces issues that make key estimation more difficult, meaning that songs that has been impacted by ‘The Loudness War’ , outlined in Section 3.3.5, could be considered less likely to be able to have an accurate key estimate generated for them. However, it would be unreasonable to assert this without the caveat that the implementations involving the processes of amplification and/or compression were applying the operation to tracks that could possibly have already had these processes applied during the course of editing and mixing the song in the studio before its release. What can be said with certainty is that this potential issue is deserving of further investigation and experimentation in order to determine if this is an element of music production that could be introducing obstacles for the algorithm in its current form. Ideally, this experimentation would involve the sourcing or creation of completely unaltered, unedited, original studio recordings of each element of a song from this genre of music in order to allow the various elements of the song to be edited and mixed at will, allowing for the impact of not only amplification or compression, but each possible technique involved in music production to be evaluated.

Additional new experimentation and research could be done by performing experiments similar to the effects pedal test outlined in Section 3.4.2. Although the experimentation carried out in these tests was enough to gather a number of interesting findings regarding the impact of various effects pedals on estimate quality, there is potential for much more to be done in this space. For instance, the sample track used was perfectly in key, and did not feature any notes that did not belong to the key of C major. I believe that it would be interesting to inspect performance of the same effects on a similar track that includes accidentals, noise, and other sounds that are not at the frequencies of notes within the key of the composition. As well as this, new effects pedals, settings levels and combinations of effects could be tested. As the experiments involving effects pedal usage were carried out in mid-March 2020, access to recording equipment and additional effects pedals were extremely limited as a result of the government social distancing measures put into place in response to the COVID-19 pandemic. This limited the number of test tracks that could be created, meaning that there are more potential findings to be made regarding the impact of effects pedals on algorithmic performance on the resulting signals they can produce.

Similarly, further investigation regarding the performance impact of varying threshold parameters for high-pass, low-pass and band-pass filters would certainly be beneficial for the future improvements of the algorithm. The parameters used for the band-pass filter that is utilised in the best-performing implementation of the algorithm from this project comes from combining the best-performing low-pass filter with the best-performing high-pass filter. It is absolutely possible that there are other parameters for the filters that were unfortunately not tested during this project that could obtain even better results if used in place of the current parameters. Additional testing and research into this area of performance could prove to be a fruitful endeavour.

The discovery of the conditions that increase the likelihood for incorrect estimate generation fulfils the goal set out at the beginning of the project to investigate failure cases in order to determine a variety of features

that can make accurate key estimation more difficult for a song. The knowledge of how and why the algorithm fails to correctly estimate the key in which songs are composed will be of great assistance moving forward with future investigation and refinement of the performance of this algorithm. With these known conditions that create issues for the algorithm in its current form, further research and investigation can be directed into determining potential methods for dealing with these conditions and thereby improving the performance of the algorithm as a whole. Further improvements could be obtained by determining a reliable method to distinguish between melodic and non-melodic frequencies, meaning that audio from drums, non-musical sounds and noise could potentially be removed from a signal, even within the frequency range of the music itself. This would strip away problematic portions of audio leaving the true melodic and harmonic segments of the audio to be analysed, which would surely lead to improved performance, as indicated by the results of the experiments that found audio contributed by drums to generally decrease the quality of estimates, which can be found in Section 4.1.1.

Further analysis that could be performed using the existing implementations of the algorithm that have been created for this project could include testing over a larger dataset in order to obtain more accurate and reliable results regarding the different performance levels between implementations of the algorithm. As well as this, the dataset could be expanded outside of the rock and metal music genre in order to evaluate the performance of the implementations on other genres that are different in musical structure, and compositionally further from classical music. Genres such as pop, hip-hop, folk and country music could highlight new and interesting obstacles that are not accounted for by any of the implementations created thus far.

A further improvement that could possibly be made to the algorithm that was unable to be tested within this project is the inclusion of additional profiles that the frequency profile of a track being analysed can be compared to. These profiles could represent other musical building blocks other than keys, such as modes and diminished scales. The reason why this could not be tested during the course of this project is because the addition of these profiles would require the development of tonal hierarchies for each mode, scale, et cetera. In order to develop these profiles to the same standard of the currently existing key profiles, identical research to that performed to create the key profiles would be required in order to determine the correct weights to apply to each possible note. This was not feasible for this project, however the addition of these profiles, if implemented well, could lead to a large portion of previously uncategorizable songs being able to have correct estimates generated for them. Furthermore, the estimates generated by the algorithm would gain an additional layer of specificity through being able to determine what mode a song is in, rather than what key shares its notes with the mode of the song, for example.

Overall, this project can be deemed to have achieved the goals laid out at the beginning, in that a number of potential causes for failure cases have been identified, and a vast array of new implementations of the algorithm were developed and tested, attaining a wide variety of performance levels. From these tests, a number of approaches have been designed that could potentially merit further investigation in future, as they have shown promise over the dataset used for this project.

Many of the improvements made to the algorithm throughout this project were simply made by combining approaches that provided improvements of small margins in order to obtain an enhanced level of performance overall. I expect that there are many more adjustments that can be made to obtain similar, or perhaps even larger, margins of improvement, and that it is the combination of these various factors that will drive the progression of this algorithm forwards.

This algorithm has proven to be an extremely interesting and thought-provoking technology to investigate for my final year project, and I believe that there are a number of improvements and refinements that can continue to be made to this algorithm in order to create an even stronger-performing, more reliable tool that can see major benefits and usage throughout the music industry – from individual musicians, composers and music academics, to large-scale, top-level music industry operations.

The algorithm may never be perfect, as a result of the sheer vastness of possibilities in musical composition, however this should not prevent us from striving towards strengthening performance as much as possible, in order to bring the quality of algorithmic estimation of musical key to the highest standard attainable, and I hope that my research into this space will have aided this effort to some extent.

A1. Appendix

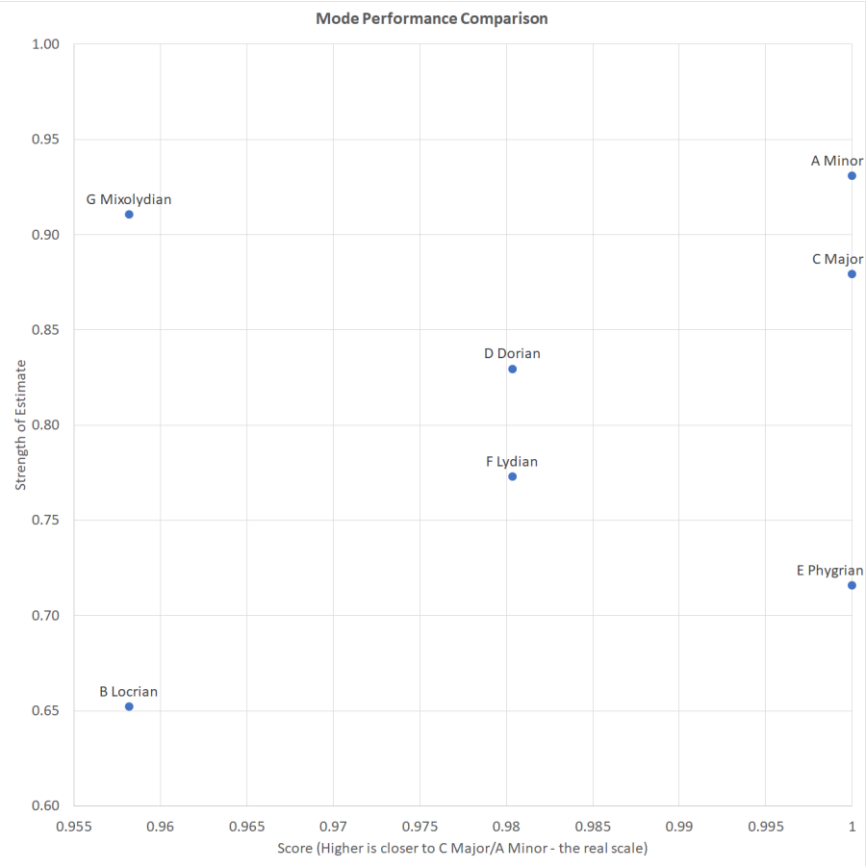
A1.1 Fourier Transform

This is an operation that can break a signal down into the various individual frequencies from which the signal is comprised. This operation transforms a signal from the time domain to the frequency domain, allowing for investigation and analysis of the frequencies involved in the signal. Namely, the Fast Fourier Transform is used for the purposes of this project.

A1.2 Alternative Visualisation

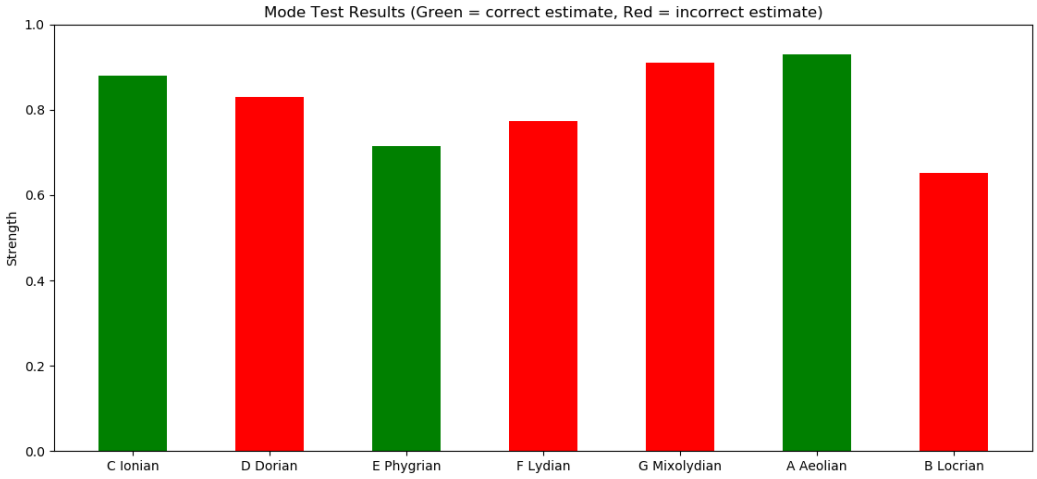
Additional visualisations were created for various different results throughout the project. These can be seen here:

- Mode Experiments 1:



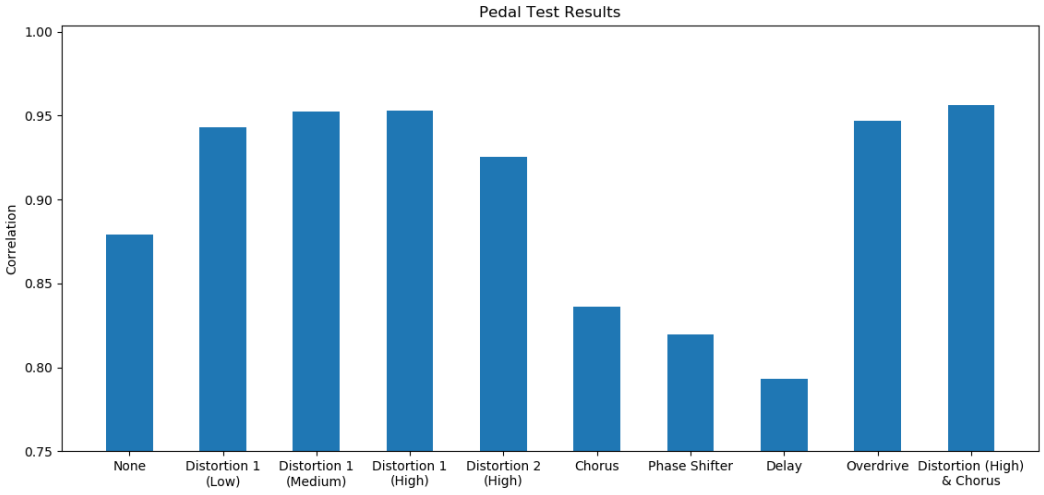
In this visualisation, the further to the top right corner a plot is, the better the performance can be considered. This was not chosen over the method of result discussion used in the Results section as it can be slightly confusing. It is worse for a result to correlate with an incorrect key strongly than it is to not correlate well at all, as this indicates an extremely misrepresentative tonal hierarchy being generated for the track. Similarly, it is better to correlate weakly to a correct estimate than to not correlate to a correct estimate at all.

- Mode Experiments 2:



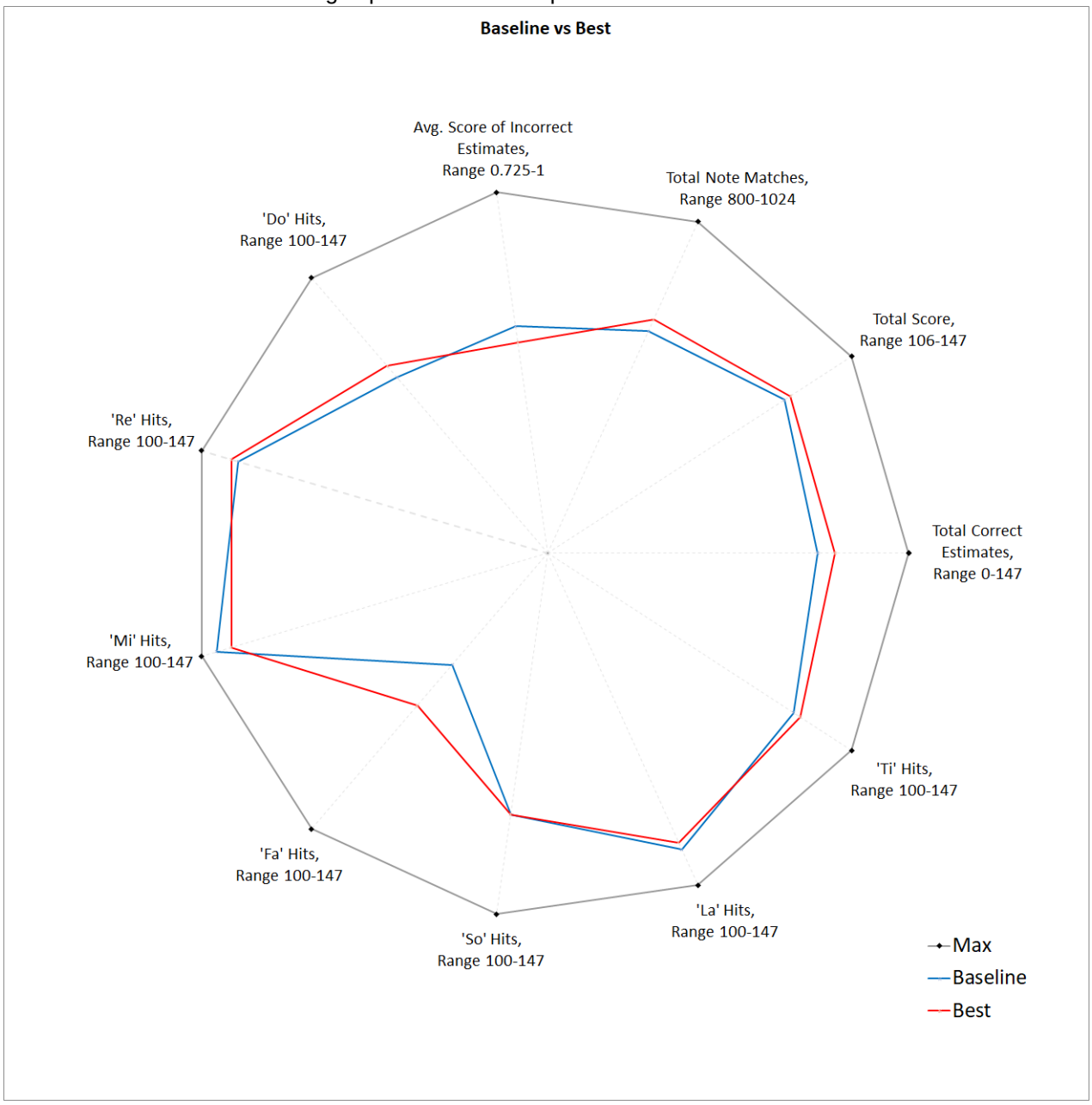
This is similar to the previous visualisation. The size of the bar can be misleading, as it can make some incorrect estimates appear better than other, correct ones due to being larger, when the colour is the more important factor.

- Pedal Experiments:



This visualisation is more easily understandable than those made for the mode experiment, as all of the estimates are correct.

- Baseline versus Best-Performing implementation comparison:



This radar chart compares all measured metrics in a single graph. Through this it can be observed that the baseline implementation is outperformed by the best-performing implementation over a number of metrics. Note that the tone-specific misses have been changed to hits for clearer visualisation, by subtracting the misses from the total number of tones to be detected in the dataset.

A1.3 Additional Data, Images & Results

All images, plots, graphs, results and dataset information generated throughout this project have been supplied in an additional .zip file submitted along with this report to be viewed if desired.

References:

[1]: Nattiez JJ. Music and discourse: Toward a semiology of music. Princeton University Press; 1990 Nov 21.

[2]: Forte A. Tonal harmony in concept and practice. Holt, Rinehart and Winston; 1974.

[3]: Langner G, Ochse M. The neural basis of pitch and harmony in the auditory system. Musicae Scientiae 2006; 10:185–208. doi:10.1177/102986490601000109.

[4]: Prout E. Harmony: its theory and practice. Cambridge University Press; 2011 Nov 8.

[5]: Definition of Semitone [Internet]. Merriam-webster.com. 2020 [cited 24 April 2020]. Available from: https://www.merriam-webster.com/dictionary/semitone.

[6]: Cooper P. Perspectives in music theory: an historical-analytical approach. Harpercollins College Div; 1981.

[7]: Definition of Whole Step [Internet]. Merriam-webster.com. 2020 [cited 24 April 2020]. Available from: https://www.merriam-webster.com/dictionary/whole%20step.

[8]: Definition of Half Step [Internet]. Merriam-webster.com. 2020 [cited 24 April 2020]. Available from: https://www.merriam-webster.com/dictionary/half%20step.

[9]: Benward B, Saker MN. Music in theory and practice. McGraw-Hill; 1997 Jan.

[10]: Kennedy M, Bourne J. The concise Oxford dictionary of music. OUP Oxford; 2004 Apr 22.

[11]: Latham A, editor. The Oxford companion to music. Oxford University Press; 2002.

[12]: Benward B, Saker M. Introduction. The materials of music: Sound and time. Music in theory and practice, pp. xii–25). NY, NY: McGraw-Hill. 2003.

[13]: Definition of API [Internet]. Merriam-webster.com. 2020 [cited 24 April 2020]. Available from: https://www.merriam-webster.com/dictionary/API.

[14]: International Organization for Standardization. ISO/IEC 13818-3:1998 Information technology — Generic coding of moving pictures and associated audio information — Part 3: Audio: ISO; 1998.

[15]: IBM Corporation and Microsoft Corporation, Multimedia Programming Interface and Data Specifications 1.0, August 1991

[16]: Y. Shafranovich, Common Format and MIME Type for Comma-Separated Values (CSV) Files, RFC Editor, October 2005

[17]: https://www.python.org/about/ [accessed 24 April 2020]

[18]: https://essentia.upf.edu/ [accessed 24 April 2020]

[19]: https://numpy.org/, https://www.scipy.org/scipylib/index.html [accessed 24 April 2020]

[20]: https://pandas.pydata.org/ [accessed 24 April 2020]

[21]: https://pypi.org/project/youtube_dl/ [accessed 24 April 2020]

[22]: https://matplotlib.org/ [accessed 24 April 2020]

[23]: https://www.ffmpeg.org/ [accessed 24 April 2020]

[24]: https://pypi.org/project/natsort/ [accessed 24 April 2020]

[25]: https://pypi.org/project/Pillow/ [accessed 24 April 2020]

[26]: https://www.audacityteam.org/ [accessed 24 April 2020]

[27]: Krumhansl CL. Cognitive foundations of musical pitch. Oxford University Press; 2001 Nov 15.

[28]: https://github.com/GiantSteps/giantsteps-key-dataset [accessed 24 April 2020]

[29]: https://ddmal.music.mcgill.ca/research/The_McGill_Billboard_Project_(Chord_Analysis_Dataset)/ [accessed 24 April 2020]

[30]: https://tunebat.com [accessed 24 April 2020]

[31]: http://www.songkeyfinder.com/ [accessed 24 April 2020]

[32]: Cunningham, Stuart & Grout, Vic & Bergen, Harry. Mozart to Metallica: A Comparison of Musical Sequences and Similarities. 332-339, 2005.

[33]: Tagg P. Everyday tonality. Towards A Tonal Theory Of What Most People Hear. New York & Montreal: The Mass Media Scholars' Press Inc. 2009.

[34]: Ward Silver H. Filter Basics: Stop, Block, and Roll(off) [Internet]. Nuts and Volts Magazine. 2018 Available from: https://www.nutsvolts.com/magazine/article/filter-basics-stop-block-and-rolloff

[35]: https://www.mtx.com/library-clipping [accessed 24 April 2020]

[36]: Devine, K. Imperfect Sound Forever: Loudness Wars, Listening Formations and the History of Sound Reproduction. Popular Music, 32(2), pp. 159-176. 2013 doi: 10.1017/S0261143013000032

[37]: Randel, DM. The Harvard Dictionary of Music. Harvard University Press; 2003.

[38]: O'Donnell M. Perceptual Foundations of Sound [Internet]. People.cs.uchicago.edu. [cited 24 April 2020]. Available from: http://people.cs.uchicago.edu/~odonnell/Scholar/Work_in_progress/Digital_Sound_Modelling/lectnotes/node4.html [accessed 24 April 2020]

[39]: Forster C. Musical Mathematics: On the Art and Science of Acoustic Instruments. Chronicle Books; 2010.

[40]: James Hardy, "The Beats to Beat: A History of Guitar Hero", History Cooperative, May 15, 2014, https://historycooperative.org/the-beats-to-beat-a-history-of-guitar-hero/. [accessed April 25, 2020]

[41]: https://www.basicmusictheory.com/c-major-scale [accessed 24 April 2020]