University of Dublin

TRINITY COLLEGE

Reducing Dissonance with Dynamic Tuning Algorithms for MIDI Synthesis

Kilian Kirsch
M.A.I. (Electronic & Computer Engineering)
Dissertation May 2021
Supervisor: Prof. David Gregg

School of Computer Science and Statistics

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

DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

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.

____ __________________________

Name Date

ABSTRACT

This dissertation investigates a software based tuning method as an alternative to the standard western, one-size-fits-all approach of tuning music pitches in the form of twelve-tone equal temperament (12-TET). It is based on the belief that software synthesized music does not have to follow the constraints of traditional instruments which cannot be tuned, with precision, in a practical timeframe. The aim is to provide a custom tuning system, that considers the melody of a song.

Four popular songs read from standard MIDI files were parsed and represented in a program. This representation was analysed for its structure, dissonance between notes, and run through a gradient descent algorithm. This algorithm optimizes the frequency of notes to reduce calculated dissonance between simultaneous notes, or chords. Results were then compared to standard tuning methods and the algorithm fine-tuned to maximize the reduction in dissonance.

The algorithm shows up to 12% reduction in total calculated dissonance. Successful dissonance reduction occurs in 3 out of 4 MIDI tunes of varying complexity. It can be concluded that reducing dissonance in comparison to 12-TET is possible using the methods outlined in this project. Possible adaptations to improve usability and widen the tuning parameters are discussed in later sections of the dissertation.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Professor David Gregg, for the regular meetings, insightful feedback, and guidance during the project.

Thank you to my girlfriend, Hannah, and my mother, Monika, for being cheerful and welcome distractions in a time where it is easy to fall into an unbalanced routine, for always being around to talk, and putting food on the table.

Thank you also to any friends who have supported me throughout the last year.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

NOMENCLATURE

12-TET – Twelve-tone equal temperament

MIDI – Musical Instrument Digital Interface

VST – Virtual Studio Technology

DAW – Digital Audio Workstation

SMF – Standard MIDI File

MPE – MIDI Polyphonic Expression

Pitch – The fundamental frequency of a music note

Melody/Tune – Multiple pitches in series

Chords – Multiple pitches played together

Harmony – How pitches played together interact

Timbre – The frequency spectrum that defines the unique sound of an instrument

1 INTRODUCTION

Musical tuning intervals are determined by the mathematical ratio between the frequencies of sine waves, representing notes. In general, the 'simpler' the ratio, the more harmonious it sounds to the human ear. For example, if the note A_4 is a 440 Hz sine wave, its 'octave'[1], A_5, is 2:1 the frequency, 880 Hz. As the fractional denominator for this ratio is 1, the smallest possible positive integer, this is known as the least dissonant or most consonant ratio. If a sound is pleasant to the ear, it is known to be *consonant*, with the opposite, an unpleasant sound, being *dissonant*. This relationship can be measured and will be explored. There are many other small integer ratios such as the 'major second' - 9:8, 'major third' – 5:4, 'perfect fifth' – 3:2 etc. Such frequency ratios are also known as *just intervals*. Groups of intervals that sound pleasant when played together are known as *scales*. A scale that implements just intervals is known as a *just intonation scale*. If a tune is played with intervals between notes that are close to just intervals it could sound superior to the listener over picking two frequencies at random.

However, it becomes difficult to incorporate these exact ratios on instruments, as there are not enough keys/strings/frets to represent the near infinite number of frequencies required if notes were to be tuned based on the notes played previous. It is also not possible to stick to true just intonation intervals when multiple notes are played at once, as the ratio between all notes is different and just intervals are not equally spaced. This is because instruments have fixed frequencies to represent notes. Instruments, such as the piano in Figure 6, tend to split an octave into 12 *semitones*, which is known as a *chromatic scale*. It is not possible to split an octave into 12 using evenly spaced small integer ratios.

The following describes the ratio issues with just intonation when it is applied to instruments with fixed tuning:

- Consider a note, 'x', with a pitch of 500 Hz. It has a major second just interval, 9:8 the frequency, 562.5 Hz. Let this pitch be note 'y'. A major second is 2 semitones above the original note.
- The major second for note y, is note 'z' with a pitch of 632.8 Hz – 9:8 the frequency of 562.5 Hz and another 2 semitones.
- Note z should be 4 semitones above the original note, x, which is a major third interval – a 5:4 frequency ratio in just intonation. However, the major third of 500 Hz is 625 Hz.
- Note z now has two different tunings, 632.8 Hz relative to y and 625 Hz relative to x. This effect compounds further when notes are tuned relative to the two frequencies of note z.

Figure 1 shows 12 semitones between C_4 and C_5, and how just intonation can vary due to inconsistent intervals. The frequencies change based on which note they are tuned in relation

[1] 'Octave' is the name given to the 2:1 frequency ratio of one note to another.

to. This note is the *unison* interval, which has a 1:1 frequency ratio. A_4 is at a constant 440 Hz, but its interval from the reference note changes.

In western modern music, instruments are tuned to a standard known as 12-tone-equal temperament (12-TET). This tuning approximates just intervals while still being applicable to instruments with a limit of keys/strings/holes to represent notes, such as pianos, guitars, and flutes, by being evenly spaced on a logarithmic scale. Figure 1 shows this approximation by how the equal temperament frequencies fall between the different just intonation scales' frequencies.

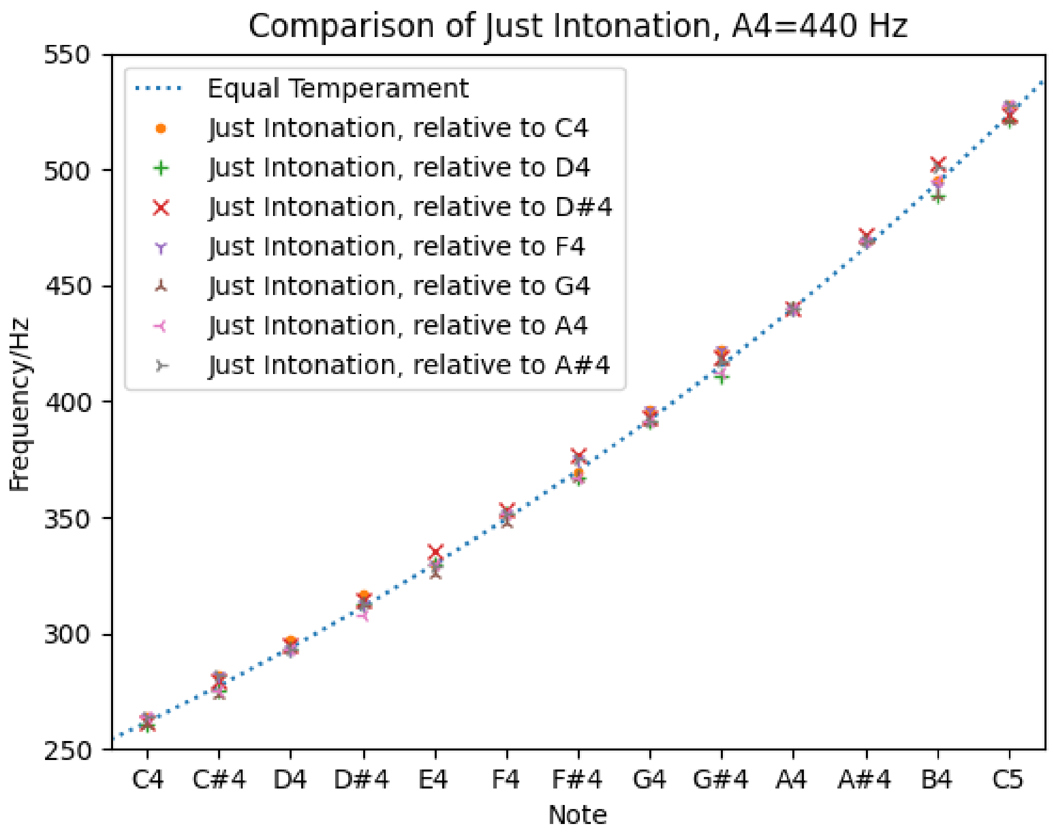


Figure 1 Comparison of just intonation frequencies when tuned relative to different notes.

As music can also be synthesized in software, where restrictions on the flexibility of tuning notes do not exist, a tuning system could therefore be created that adjusts based on the specific intervals in a musical piece. This system could be programmed to optimize these intervals to be less dissonant than 12-TET. It is the goal of this project to apply an algorithm that can find a unique tuning system for every tune it is applied to, that will improve on the generic equal temperament.

My project will examine how software synthesizers read and interpret music in the form of the *Musical Instrument Digital Interface* (MIDI) standard. The MIDI standard is used to store information on the notes played in a song, along with information such as timing, intensity of key presses and instrument names. MIDI is the prevalent standard for conveying this information to electronic instruments. The file structure of a standard MIDI file will be broken down and explained, as MIDI needs to be understood to extract information for the algorithm. The project will read a MIDI file of any tune to understand when and what notes are played, by which instruments.

Based on this, frequencies will be assigned to the notes played. These frequencies can be used to calculate the dissonance between notes played simultaneously in the tune. The project algorithm will use a calculation of dissonance between the frequencies of notes to quantify how pleasant a group of notes sound to the listener. When multiple notes are compared at once, there are minima on the multi-dimensional dissonance plane, where the dissonance is at its lowest. The aim of the project is to reduce this dissonance, by altering the frequencies, to find the intervals between note frequencies that converge in one of these minima. It would be possible to find the maximum dissonance reduction by changing all frequencies to be the same. However, the frequency change will be monitored to stay true to the original melody. To represent the new intervals between frequencies, a logarithmic ratio called 'cents' is used.

Chapter 2, Background, will explore the information, technology, and equations needed to understand the project. Chapter 3, Implementation, will discuss the application of this technology. It will also highlight how dissonance measurements are applied to the algorithm. Chapter 4, Results & Discussion, will discuss the results of the algorithm and how it compares to other possible tunings. It will also discuss the algorithm's weaknesses for application. Finally, Chapter 5, Conclusion, will summarize the project and make suggestions on the future work that could be added to the project. The section on future work will describe additional parameters to be added to the algorithm to improve tuning accuracy and usability.

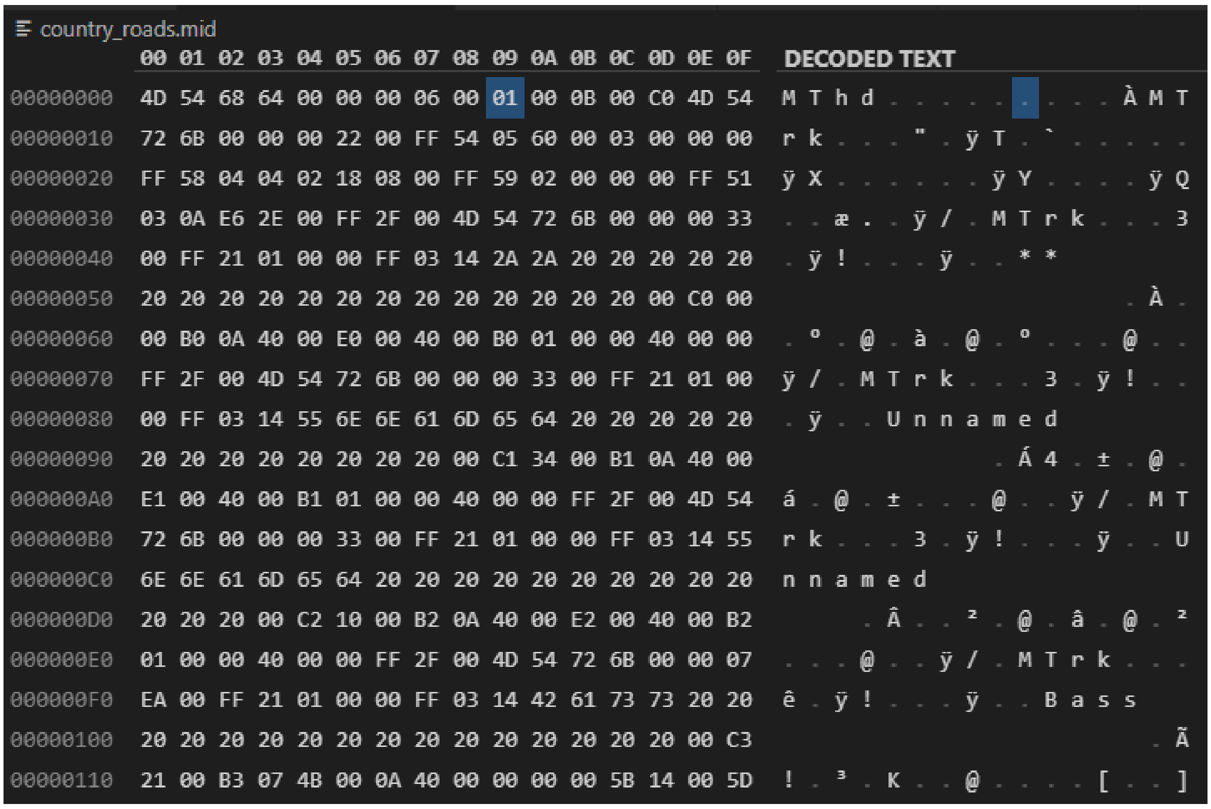
2 BACKGROUND

2.1 MIDI

2.1.1 What is MIDI?

Musical Instrument Digital Interface (MIDI) is a standard used by manufacturers to convey information about how a piece of music is played on an electronic instrument (Lehrman, 2017). These instruments can also be called synthesizers and often take the form of a keyboard. MIDI does not define the actual sound of the music, but the synthesizer produces it based on its specifications of waveforms, frequencies, and filters. Therefore, a MIDI signal or file will sound different based on what synthesizer it is played on. Synthesizers are not limited to hardware but can take the form of software synthesizers as Virtual Studio Technology (VST) plugins to Digital Audio Workstations (DAWs), used by electronic music producers to produce a desired sound. These 'softsynths' will be the focus of this project. Different softsynths come with different functionalities, the focus of this project will be those that allow the user to specify tunings for played notes, beyond the constraint of semitone intervals. Well known free open source examples of such are ZynAddSubFX/Zyn-Fusion[2] and Surge[3].

2.1.2 Standard File Structure



```
country_roads.mid
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  DECODED TEXT
00000000  4D 54 68 64 00 00 00 06 00 01 00 0B 00 C0 4D 54  MThd.........ÀMT
00000010  72 6B 00 00 00 22 00 FF 54 05 60 00 03 00 00 00  rk...".ÿT.`.....
00000020  FF 58 04 04 02 18 08 00 FF 59 02 00 00 00 FF 51  ÿX......ÿY....ÿQ
00000030  03 0A E6 2E 00 FF 2F 00 4D 54 72 6B 00 00 00 33  ..æ..ÿ/.MTrk...3
00000040  00 FF 21 01 00 00 FF 03 14 2A 2A 20 20 20 20 20  .ÿ!...ÿ..**     
00000050  20 20 20 20 20 20 20 20 20 20 20 20 20 00 C0 00               .À.
00000060  00 B0 0A 40 00 E0 00 40 00 B0 01 00 00 40 00 00  .°.@.à.@.°...@..
00000070  FF 2F 00 4D 54 72 6B 00 00 00 33 00 FF 21 01 00  ÿ/.MTrk...3.ÿ!..
00000080  00 FF 03 14 55 6E 6E 61 6D 65 64 20 20 20 20 20  .ÿ..Unnamed     
00000090  20 20 20 20 20 20 20 20 00 C1 34 00 B1 0A 40 00          .Á4.±.@.
000000A0  E1 00 40 00 B1 01 00 00 40 00 00 FF 2F 00 4D 54  á.@.±...@..ÿ/.MT
000000B0  72 6B 00 00 00 33 00 FF 21 01 00 00 FF 03 14 55  rk...3.ÿ!...ÿ..U
000000C0  6E 6E 61 6D 65 64 20 20 20 20 20 20 20 20 20 20  nnamed          
000000D0  20 20 20 00 C2 10 00 B2 0A 40 00 E2 00 40 00 B2     .Â..².@.â.@.²
000000E0  01 00 00 40 00 00 FF 2F 00 4D 54 72 6B 00 00 07  ...@..ÿ/.MTrk...
000000F0  EA 00 FF 21 01 00 00 FF 03 14 42 61 73 73 20 20  ê.ÿ!...ÿ..Bass  
00000100  20 20 20 20 20 20 20 20 20 20 20 20 20 20 00 C3                .Ã
00000110  21 00 B3 07 4B 00 0A 40 00 00 00 00 5B 14 00 5D  !.³.K..@.....[..]
```

Figure 2 Contents of a standard MIDI file

[2] https://zynaddsubfx.sourceforge.io/
[3] https://surge-synthesizer.github.io/

The following section is based on (MIDI Manufacturers Association, 1996).

MIDI can be both transmitted electrically by serial transmission over a cable or read from a Standard MIDI File (SMF). An SMF has the file extension (.mid) of which there are 2 common types. Type 0 uses a single 'track' for all information, whereas type 1 allows for multiple tracks, which can be useful for dividing out information for different instruments. As type 1 is the most relevant format for this project, this will be examined further. Figure 2 shows the contents of a SMF, opened with the Visual Studio Code Hex Editor extension.

SMFs are split into chunks, that have a 4-byte identifier string, 4-bytes for number of data bytes, and then the chunk data. The identifier strings can be the header string, "MThd", or the track string, "MTrk". The layout of this is shown in Figure 3. The header chunk of a MIDI file defines some important information such as the number of track chunks in the file. Each track chunk contains information about how one instrument is supposed to play, or synthesize, a tune.

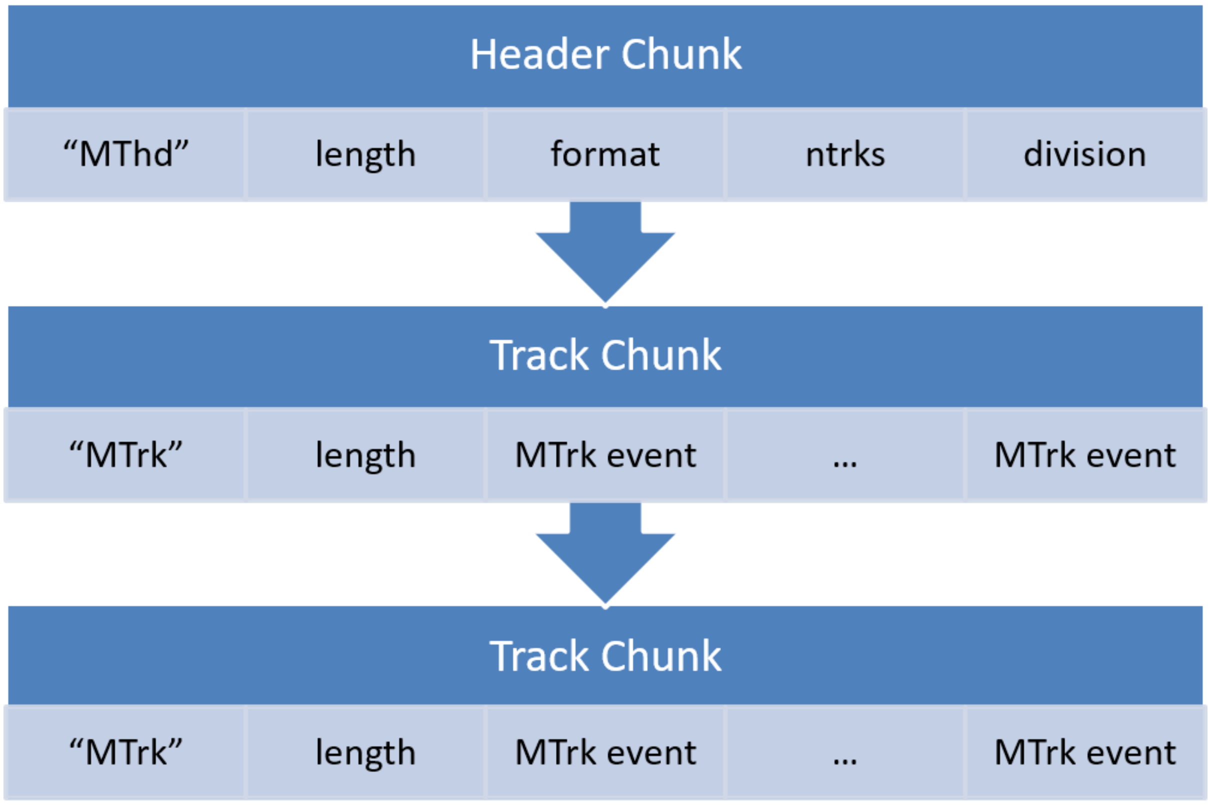


Figure 3 General layout of MIDI chunks

2.1.2.1 Header Chunk

All type 1 MIDI files, which contain multiple tracks, must start with the 4-byte string "MThd", or 0x4D546864. This symbolizes the start of a MIDI file and the beginning of a header 'chunk'. This is followed by 4 bytes defining the length of data in the header chunk in bytes. The header data length will always be 6 bytes, which is made up of 2 bytes for the MIDI file type, 2 bytes for the number of tracks, and 2 bytes for the timing definition. This is shown in Table 1.

The timing definition, or division, can have two formats. If it starts with 0, the other bits will define the timing in ticks per quarter-note/beat. If it starts with 1, the other bits will define frames per second and ticks per frame, however this format is rare.

A common header chunk will therefore be:

Table 1 Examination of a SMF header chunk

chunk type	**length**	**format**	**ntrks**	**division**
0x4D546864	0x00000006	0x0001	0x000B	0x00C0
"MThd" chunk type	6 header data bytes to follow	Type-1 Standard MIDI file	11 track chunks to follow	192 ticks per beat

2.1.2.2 Track Chunks

Following the header chunk are the track chunks. Track chunks start off like a header chunk, defining the chunk type as "MTrk", or 0x4D54726B, and 4-bytes to define the chunk length. The structure of the track data is different to the header data.

The track data is separated into MTrk, or track, events, which are made up of delta time and a command/event. Figure 3 shows where these events are in relation to the beginning of a chunk.

Delta time describes the time that has passed since the previous event, given in ticks, the resolution of which was described in the header chunk. Delta time is given as a 'variable-length quantity'. This allows 4-byte numbers up to 0x0FFFFFFF to be represented, while smaller values such as 0x7F are still represented in 1 byte, saving space. When reading variable length numbers, 7 bits are used per byte, i.e. (value & 0x7f), a mask of the value and the first 7 bits of a byte. The most significant bit in every byte, signals whether more bytes follow, or not. If the value masked with the MSB of the byte, is true, i.e. (value & 0x80), a further byte is included in the variable-length number, until it is false. For example, the maximum representable variable-length quantity 0x0FFFFFFF is represented by 0xFFFFFF7F, as shown in Table 2.

The event related to delta time can take one of three forms:

- Meta-event
- MIDI event
- Sysex event

The format of these events can vary, however, each one will start with a 'status byte', which defines the type of messages. The status byte is usually followed by one, or multiple, data byte(s). Status bytes are in the range of 0x80 – 0xFF and data bytes are 0x00 - 0x7F. This way they can be distinguished by whether a byte's MSB is high or low. Unrecognized status bytes and related data are ignored, as opposed to throwing an error.

Table 2 Variable-length quantity number vs. representation

Step 1: Original Number								
Hex number by byte	0x0F		0xFF		0xFF		0xFF	
Binary number	0000_1111		1111_1111		1111_1111		1111_1111	
Step 2: Separate 7 bits per byte, add continuation bit "1" as MSB for all but last byte								
7 bits per byte	1111_111		1_1111_11		11_1111_1		111_1111	
Continuation bit when numbers follow	Cont.	Value	Cont.	Value	Cont.	Value	Cont.	Value
	1	1111_111	1	1_1111_11	1	11_1111_1	0	111_1111
Step 3: Variable-length representation								
Binary repr.	1 + 1111_111		1 + 1_1111_11		1 + 11_1111_1		0 + 111_1111	
Hex repr.	0xFF		0xFF		0xFF		0x7F	

2.1.2.2.1 Meta-Events

Meta-events specify song-related information, such as key and time signatures, track names, etc. They begin with 0xFF, followed by the type of meta message, and the number of data bytes as a variable-length quantity.

The meta-events important for this project are:

- Track Name (0x03): Used to label a track, often with the instrument intended for the track. After the length, string text is used to define the name. For example, if the instrument name is "Bass", this will be encoded as ASCII characters. As track name is a meta-event, the message starts with 0xFF; followed by 0x03, for the type track name; then 0x04 to represent the data length - 4 bytes, as there are 4 ASCII characters; then finally, the ASCII string "Bass".
- Instrument Name (0x04): As the track name event exists, this meta-event is not used much. The use is the same.
- End of Track (0x2F): Non-optional meta event to signal the end of a track. It does not have any data bytes: 0xFF2F00
- Set Tempo (0x51): This defines the tempo in microseconds per quarter-note and can be used to calculate a song's beats per minute (bpm). This event will always have 3 bytes of data, for example: 0xFF03 + 0x07A120. This specifies a tempo of 500,000 microseconds per beat, which is equivalent to 120 beats per minute: $BPM = \frac{1}{tempo} \times 10^6 \times 60$. If this message is missing, a default tempo of 500,000 µs/quarter-note is assumed.
- Time Signature (0x58): Defines the time signature, or beats per bar, of a song. The length of this message is always 4 bytes, made up of 2 bytes for the numerator of the signature, 2 bytes for the denominator of the signature, 2 bytes for the number of MIDI clocks in a metronome click, and the number of notated 32nd notes per beat/quarter note. The denominator is given as a negative power of 2, i.e. 2^{-d}. To define a time signature of 4 quarter(1/4) notes per bar, with 24 MIDI clocks per quarter

note, and naturally 8 32nd notes per quarter note, the following byte code is used: 0xFF5804 + 0x04 + 0x02 + 0x18 + 0x08, where 0x02 results in the denominator of 4.

- Key Signature (0x59): Always a length of 2 bytes, with the number following the length specifying the number of flats if negative and the number of sharps if positive. 0 indicates a key of C. The second data byte specifies a major key if 0, and minor key if 1, i.e. 0xFF5902 + 0x01 + 0x00 translates to a G major key.

Meta-events such as tempo and time signature are stored in a MIDI file's first track, while subsequent tracks contain the instrumental performances with the track name and end of track meta messages.

2.1.2.2.2 MIDI Events

MIDI events are specified to be on a channel, from 1 to 16 (which is coded as N-1, where N is channel number). The two most common MIDI events are *Note Off* and *Note On*. Both have a similar structure, with two data bytes that specify note number and key velocity. Note numbers range from 0 – 127, with a convention of Note 0 being C_{-1} and Note 127 being G_9, where all notes are 1 semitone apart, this distribution is shown in Table 4. The subscript denotes octaves, -1 being a much lower frequency than 9, as seen in Figure 4. Velocity, also ranging from 0-127 defines the speed at which a key is pressed/released. This relates to the volume of a key press, where a Note On message with a high velocity is perceived as louder. If a Note On message has a velocity of 0, this is equivalent to a Note Off message. Example MIDI events are shown in Table 3.

Table 3 Example usage of 'Note On' and 'Note Off' Channel Voice Messages

	Status Byte	**Data Bytes**	
		Note Number	**Key velocity**
Note On	0x90	0x3C	0x40
	Note On, Channel 1	Note 60 = C_4	Average-level velocity
Note Off	0x80	0x3C	0x40
	Note Off, Channel 1	Note 60 = C_4	Average velocity
Note Off (alternative)	0x90	0x3C	0x00
	Note On, Channel 1	Note 60 = C_4	Off velocity

There are many other messages that are MIDI events, n in the status byte is the channel number i.e. channel 1:

- Aftertouch (0xAn for Polyphonic Key; 0xDn for Channel): Pressure on a key while being held. The polyphonic status byte refers to pressure on an individual key. Channel aftertouch defines pressure for a whole channel.
- Pitch Bend (0xEn): Varying the pitch of the note played.
- Program Change (0xCn): Specify the type of instrument to be used for the given channel.
- Control Change (0xBn): Control various functions in a synthesizer, such as pedals and effects.

- Channel Mode Messages: An extension of Control Change. Functions such as turning all sound off and resetting all controllers on a channel.

2.1.2.2.3 System Exclusive (Sysex) Events

Sysex events, or system exclusive messages, allow a manufacturer to define additional events specific to their synthesizer, which may not exist already in the MIDI format. They begin with 0xF0, followed by a variable length number, like delta time, to define the number of bytes to be transmitted, and end with 0xF7. The MIDI Association has made addenda to the original MIDI standard since its release, such as MIDI Tuning messages, which take on the sysex format (MIDI Manufacturers Association, 1999). However, the implementation of this is optional, and many software synthesizers do not recognize them as seen on a list of DAW plugins for microtonal tuning on the Xenharmonic Wiki (Xenharmonic Wiki, 2021), where other tuning methods than MIDI Tuning Standard are common.

2.1.3 MIDI Summary

Now that the standard MIDI file structure has been introduced, the next section will explore different interpretations of the MIDI notes. Each MIDI note read from 'Note On' and Note Off' events corresponds to a frequency. However, this frequency can vary based on which tuning system is used.

2.2 Tuning

2.2.1 Frequency/Cents

The pitch of a note in music is determined by the frequency of its wave. To define how music is played, one note is defined to be a set pitch, and others are tuned in relation to it. In modern, western, music, it is common to tune A_4, the A above middle C to 440Hz. This is standardized in (International Organization for Standardization, 1975). To describe the difference in pitch, or interval, between two notes, frequency ratios can be used. A 2:1 ratio, describes an octave. This is the same note, played at a higher pitch. For example, with the standard of A_4 = 440Hz, one octave higher, A_5, is 880Hz. Every octave is divided into twelve notes, the interval between each is known as a semitone. Simple integer ratios such as an octave are known as just intervals. However, it is impossible to define all possible intervals as a ratio of small integers.

Cents are a logarithmic measurement of intervals. An octave is measured to be 1200 cents. An interval of n cents, between two frequencies f_1 and f_2 is calculated as shown in Equation 1. In equal temperament, the interval of a semitone is constant, so that notes represented by a semitone are 100 cents apart.

Equation 1 Calculating an interval between two frequencies, in cents

$$n = 1200 \times \log_2\left(\frac{f_2}{f_1}\right)$$

2.2.2 Musical Systems

Instruments such as pianos are limited in the number of notes it can play, by the number of keys it has. This is the same for a MIDI synthesizer. The MIDI standard only allows for the use of 128 notes. By default, these notes are defined to be spaced 1 semitone apart. Table 4 shows what notes the MIDI notes correspond to by default. The interval of a semitone can vary based on the tuning system used.

Table 4 Default distribution of MIDI notes

Note/Octave	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
C	0	12	24	36	48	60	72	84	96	108	120
C♯/D♭	1	13	25	37	49	61	73	85	97	109	121
D	2	14	26	38	50	62	74	86	98	110	122
D♯/E♭	3	15	27	39	51	63	75	87	99	111	123
E	4	16	28	40	52	64	76	88	100	112	124
F	5	17	29	41	53	65	77	89	101	113	125
F♯/G♭	6	18	30	42	54	66	78	90	102	114	126
G	7	19	31	43	55	67	79	91	103	115	127
G♯/A♭	8	20	32	44	56	68	80	92	104	116	128
A	9	21	33	45	57	69	81	93	105	117	-
A♯/B♭	10	22	34	46	58	70	82	94	106	118	-
B	11	23	35	47	59	71	83	95	107	119	-

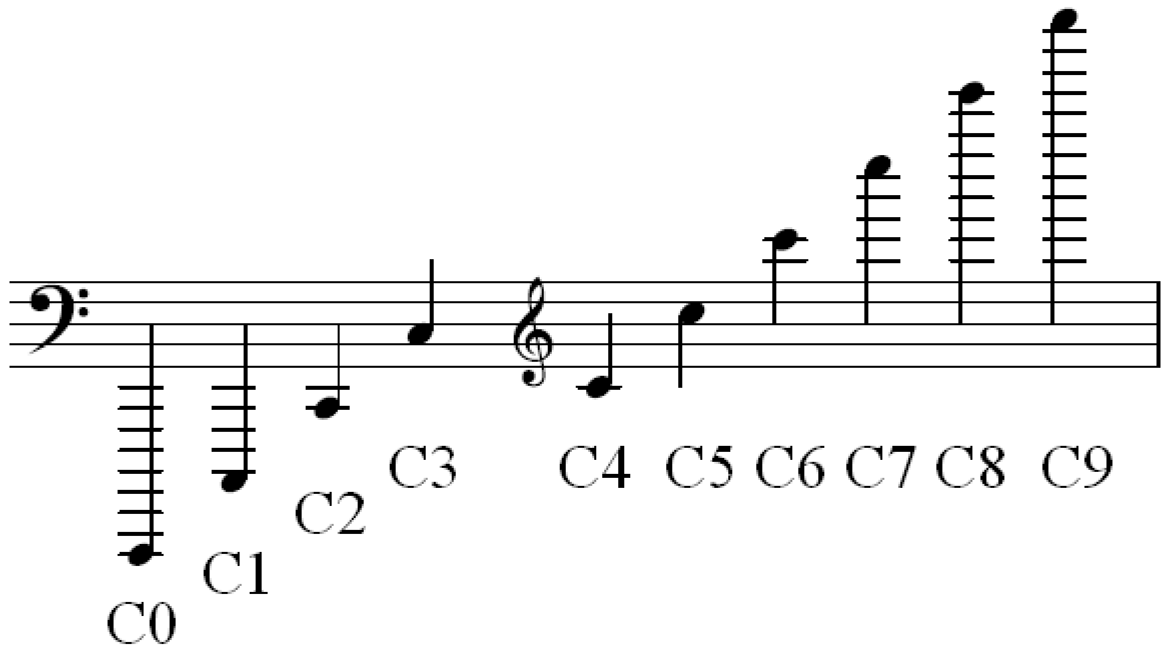


Figure 4 Comparison of a C over multiple octaves (User:Angr, 2021).

2.2.2.1 Just Intonation

In just intonation, the intervals between note frequencies are based on small-integer ratios. Table 6 and Figure 5 show the intervals that make up a twelve-tone scale. Each of these intervals are 1 semitone apart. Just intonation intervals are completely in tune with the note they are played with. However, due to the non-uniform intervals, the tuning cannot be applied to fit different combinations of notes. For example, the major third interval from a C, is an E, with a ratio of 5:4. The major third is 4 semitones above unison. If the C is tuned to a frequency of 300Hz, the E has a frequency of 375Hz. The major third of an E would then be G♯/A♭, at a frequency of 468.75Hz. That is another 4 semitones, so that the new note is 8 semitones above the original. However, this is also the minor sixth interval from the original C, which should be tuned to a ratio of 8:5, which is 480Hz. This shows that it is impossible to have just intervals between all notes of the song, and a compromise needs to be made to keep the intervals approximate to their original but be applicable to all played notes. This altering of intervals is known as temperament.

Table 5 Example comparing intervals and semitones.

Major third (4 semitones)				Major third (4 semitones)				
C	C♯/D♭	D	D♯/E♭	E	F	F♯/G♭	G	G♯/A♭
Minor sixth (8 semitones)								

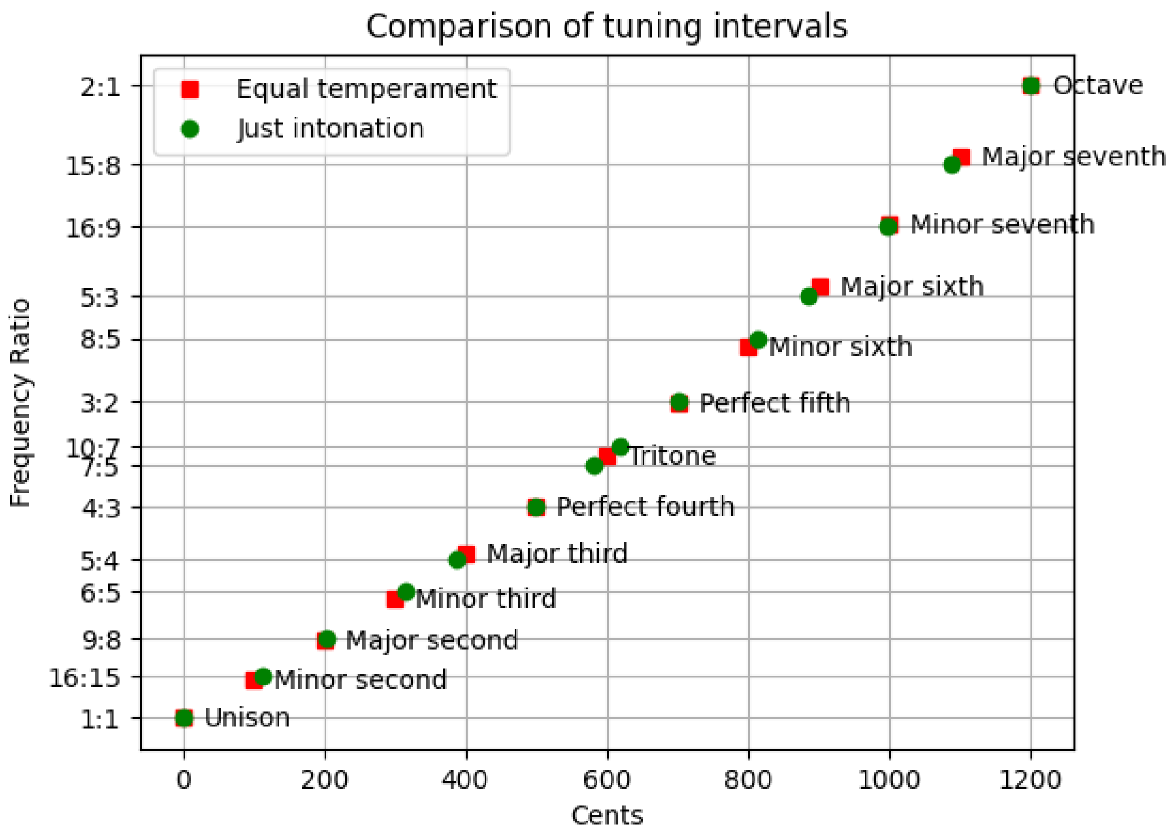


Figure 5 Equal temperament and just intonation compared by intervals.

2.2.2.2 Equal Temperament

Twelve-tone equal temperament (12-TET) is the most common tuning system used in western music. It splits an octave into 12 equal intervals, 100 cents, or $2^{1/12}$ apart. 12-TET is meant to approximate just intervals, while also being compatible with any key and scale. As the intervals are equal, there are no issues with the melody of the music creating pitch drift or uneven intervals.

2.2.2.3 Trade-Offs Between Traditional Scales

Figure 5 and Table 6 show the differences between equal temperament and just intervals in a twelve-tone scale. Both have benefits and drawbacks. While equal temperament is never in perfect tune, just intonation also has this issue when music is not played in one single key. (van Steenhoven, 2010) gives many examples of different tuning systems, which are all limited due to the constraints of physical instruments.

Table 6 Comparing 12-TET with just intonation (Wikipedia, 2021)

Name	**Exact value in 12-TET**	**Decimal value in 12-TET**	**Cents**	**Just intonation interval**	**Cents in just intonation**	**Difference**
Unison (C)	$2^{\frac{0}{12}} = 1$	1	0	1⁄1 = 1	0	0
Minor second (C♯/D♭)	$2^{\frac{1}{12}} = \sqrt[12]{2}$	1.059463	100	16⁄15 = 1.06666...	111.73	-11.73
Major second (D)	$2^{\frac{2}{12}} = \sqrt[6]{2}$	1.122462	200	9⁄8 = 1.125	203.91	-3.91
Minor third (D♯/E♭)	$2^{\frac{3}{12}} = \sqrt[4]{2}$	1.189207	300	6⁄5 = 1.2	315.64	-15.64
Major third (E)	$2^{\frac{4}{12}} = \sqrt[3]{2}$	1.259921	400	5⁄4 = 1.25	386.31	+13.69
Perfect fourth (F)	$2^{\frac{5}{12}} = \sqrt[12]{32}$	1.33484	500	4⁄3 = 1.33333...	498.04	+1.96
Tritone (F♯/G♭)	$2^{\frac{6}{12}} = \sqrt{2}$	1.414214	600	7⁄5 = 1.4 10⁄7 = 1.42857...	582.51 617.49	+17.49 -17.49
Perfect fifth (G)	$2^{\frac{7}{12}} = \sqrt[12]{128}$	1.498307	700	3⁄2 = 1.5	701.96	-1.96
Minor sixth (G♯/A♭)	$2^{\frac{8}{12}} = \sqrt[3]{4}$	1.587401	800	8⁄5 = 1.6	813.69	-13.69
Major sixth (A)	$2^{\frac{9}{12}} = \sqrt[4]{8}$	1.681793	900	5⁄3 = 1.66666...	884.36	+15.64
Minor seventh (A♯/B♭)	$2^{\frac{10}{12}} = \sqrt[6]{32}$	1.781797	1000	16⁄9 = 1.77777...	996.09	+3.91
Major seventh (B)	$2^{\frac{11}{12}} = \sqrt[12]{2048}$	1.887749	1100	15⁄8 = 1.875	1088.27	+11.73
Octave (C)	$2^{\frac{12}{12}} = 2$	2	1200	2⁄1 = 2	1200.00	0

2.2.3 Soft Synths and Tuning (scl/kbm)

Most synthesizers apply equal temperament tuning by default and need to be configured to change at what frequency tones are played. A list of DAW plugins that allow microtonal tuning can be found on the Xenharmonic Wiki (Xenharmonic Wiki, 2021), the most common tuning methods are scl/kbm files and tun file. This project will use the former, the Scala scale file format and keyboard mappings, which are documented at (de Coul, 2021) and (Scala help: Mappings, 2021).

It will use Scala scale files (scl) and keyboard mapping (kbm) files to inform the synthesizer what frequency to tune a note to. In both scl and kbm, an exclamation mark (!) specifies a comment. The first non-comment line in the scale file format is a description of the scale. The second line contains the number of note tunings specified in the file. All further lines describe pitch intervals. When an interval is given in cents, it needs to contain a period, if an interval is given as a frequency ratio it must be given as an integer, or fraction separated by a slash. The unison interval, 1/1 or 0.0 cents, is left out in a scale file. For example, a .scl file describing 12-TET is shown in Listing 1. The program of the project will provide cents values for all 128 MIDI notes.

```
! D:\12 Tone Equal Temperament.scl
!
12 Tone Equal Temperament | ED2-12 - Equal division of harmonic 2 into 12 parts
 12
!
 100.00000
 200.00000
 300.00000
 400.00000
 500.00000
 600.00000
 700.00000
 800.00000
 900.00000
 1000.00000
 1100.00000
 2/1
```

Listing 1 Example of the contents of a scale format (scl) file.

The kbm, or keyboard mapping file, define how the intervals provided in a scale file map to a MIDI instrument. For example, the 12-TET scale above, could correspond to the twelve-tone C scale, starting at middle C, or MIDI note 60. An example kbm file for when the scale is tuned relative to standard A440Hz, is given in Listing 2.

```
! Template for a keyboard mapping
!
! Size of map. The pattern repeats every so many keys:
12
! First MIDI note number to retune:
0
! Last MIDI note number to retune:
127
! Middle note where the first entry of the mapping is mapped to:
60
! Reference note for which frequency is given:
69
! Frequency to tune the above note to (floating point e.g. 440.0):
440.0
! Scale degree to consider as formal octave (determines difference in pitch
! between adjacent mapping patterns):
12
! Mapping.
! The numbers represent scale degrees mapped to keys. The first entry is for
! the given middle note, the next for subsequent higher keys.
! For an unmapped key, put in an "x". At the end, unmapped keys may be left out.
0
1
2
3
4
5
6
7
8
9
10
11
```

Listing 2 Example of the contents of a keyboard mapping (kbm) file.

2.2.4 Tuning Summary

The tuning section examined how frequencies are allocated to notes for music, and a format that a software synthesizer can use to apply customized tuning. The next section will talk about how the frequencies of multiple music notes interact, and how they can be measured to be more, or less, pleasant than another.

2.3 CONSONANCE/DISSONANCE

2.3.1 What is Consonance/Dissonance?

In relation to an interval in music, consonance relates to how pleasant or agreeable two sounds are together. Dissonance is the opposite and can sound imperfect (Lahdelma & Eerola, 2020). For reasons not fully understood humans simple, small integer, ratios between frequencies sound the most consonant (Schellenberg & Trehub, 1994). As the frequency ratio moves away from these small integer ratios, humans perceive the sound to be more dissonant.

2.3.2 Graph and Formulae Behind Sensory Dissonance vs. Frequency Ratio

Sethares (Sethares W. , 1993) introduces Equation 1, a formula to calculate dissonance between two sinusoidal frequencies:

Equation 2 Dissonance between two sinusoids.

$$d(f_1, f_2, v_1, v_2) = v_1 v_2 \left[e^{-as|f_2 - f_1|} - e^{-bs|f_2 - f_1|}\right]$$

Where f_1 and f_2 are the sinusoids' frequencies, and v_1 and v_2 are their respective amplitudes; $a = 3.5, b = 5.75; s = \frac{d^*}{s_1 \min(f_1, f_2) + s_2}$, where $d^* = 0.24$, $s_1 = 0.021$, and $s_2 = 19$.

When a tone is played, it is made up of the fundamental frequency, or pitch, and partials. The partials are higher frequencies than the pitch frequency, with smaller amplitudes. Figure 6 shows the difference between a pitch and the timbre of a note. How these partials are distributed in the frequency spectrum depends on what instrument, or synthesizer, they are played on.

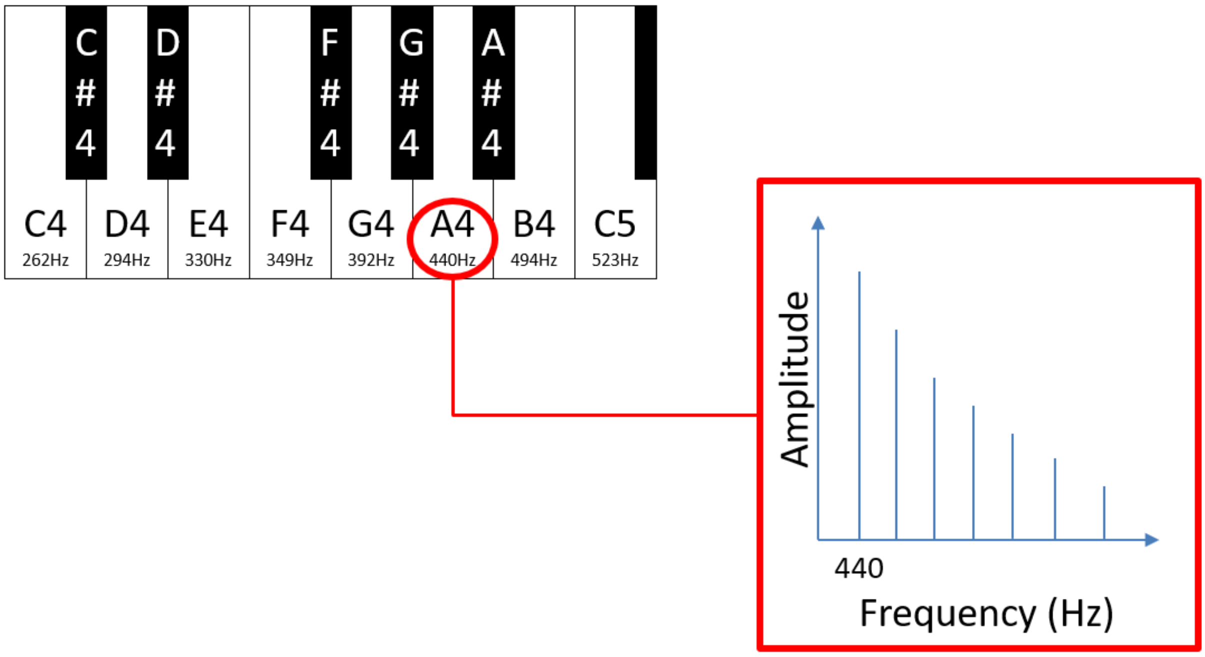


Figure 6 Example of difference between pitch and timbre.

To calculate the dissonance of a timbre, and any other notes played at the same time. The dissonances can be summed up as shown in Equation 3.

Equation 3 Sum of the dissonances of timbre and simultaneous notes played.

$$D = \frac{1}{2} \sum_{l=1}^{m} \sum_{k=1}^{m} \sum_{p=1}^{n} \sum_{q=1}^{n} d(a_p f_l, a_q f_k, v_p, v_q)$$

, where m is the number of fundamental frequencies played together i.e. pitches of separate notes, and n is the number of partials of each timbre. a_p and a_q represent the ratio between a partial frequency and the fundamental frequency, f_l and f_k, respectively. For example, if a timbre is made up of 3 sinusoids, or partials, of the frequencies 500Hz, 1000Hz and 1500Hz, where 500Hz is the pitch, or fundamental frequency, f, then a would be [1, 2, 3], so that a*f is [500, 1000, 1500]. The amplitudes v_p and v_q include the smaller amplitudes of the partials.

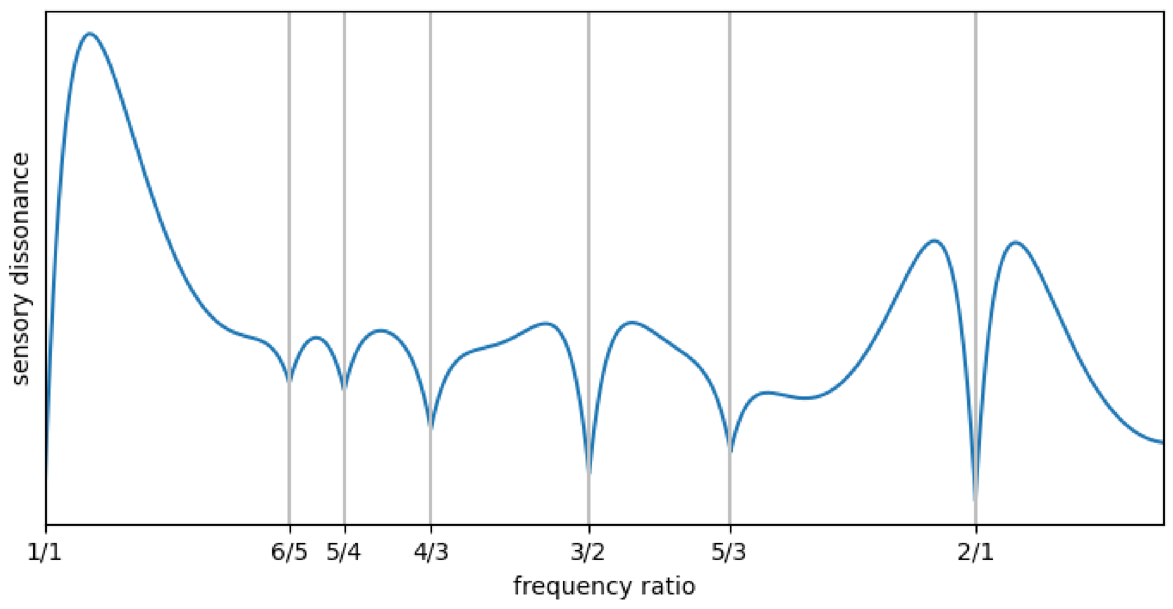


Figure 7 Example of a dissonance curve.

Figure 7 shows the dissonance between a note with 500Hz pitch, with 6 partials and a range of higher pitch frequencies with 6 partials up to 1150Hz, using Equation 3, so that $m = 2$, and $n = 6$. Example values for one calculation, between pitches f_1, 500Hz, and f_2, 1000Hz, are given in Table 7. The figure shows the trend that small integer ratio intervals have smaller dissonance.

Table 7 Values corresponding to an example dissonance calculation, between 2 notes, with 6 partials each.

a	**f**	**v**	**a*f_1**	**a*f_2**
1	500	1.00	500	1000
2	1000	0.88	1000	2000
3	-	0.77	1500	3000
4	-	0.68	2000	4000
5	-	0.60	2500	5000
6	-	0.53	3000	6000

2.3.3 Consonance/Dissonance Summary

This section covered how music can be quantified to be pleasant or unpleasant. The next section will use this information inside an optimization algorithm to reduce dissonance in a tune.

2.4 Dynamic Tuning

2.4.1 Iteration Algorithm

Equation 3 can be used throughout a tune, to reduce the dissonance of simultaneous notes, or chords, in a MIDI file. Sethares (Sethares W. , 1994) proposes the use of a gradient descent algorithm as a solution. Equation 4 is an iteration which updates a played frequency descending the steepest gradient.

Equation 4 Iteration of a gradient descent algorithm to reduce dissonance.

$$f_i(k+1) = f_i(k) - \mu \frac{dD}{df_i(k)}$$

Where f_i is one of m frequencies played at a given time; µ is the step size, and the gradient approximates the partial derivative of the sum of dissonances, with respect to f_i.

µ is chosen so that the gradient, changes the frequency at an appropriate rate, so that dissonances decrease without changing the identity of the tune.

2.4.1.1 Gradient Calculation

The calculation of the gradient proposed by Sethares (Sethares W. , 1994) is shown in Equation 5.

Equation 5 Gradient / Partial derivative of dissonance sum.

$$\frac{dD}{df_i} = \sum_{k=1}^{m} \sum_{p=1}^{n} \sum_{q=1}^{n} \frac{d}{df_i} d\big(a_p f_i, a_q f_k, v_p, v_q\big)$$

Where m is the number of notes/pitches, n is the number of partials. Here f_i is constant so that the fundamental frequency of the note adapted in the algorithm is 'f_1'. The derivative of the elements within the triple sum are shown in Equation 6.

Equation 6 Derivative of the equation for dissonance between two sinusoids.

$$\frac{d}{df} d(f,g,v,w) = \begin{cases} vw\left[-\frac{ad^*}{(fs_1+s_2)}\exp\left(\frac{ad^*(f-g)}{fs_1+s_2}\right) + \frac{bd^*}{(fs_1+s_2)}\exp\left(\frac{bd^*(f-g)}{fs_1+s_2}\right)\right], & f > g, \\ vw\left[\frac{ad^*(gs_1+s_2)}{(fs_1+s_2)^2}\exp\left(\frac{ad^*(f-g)}{fs_1+s_2}\right) - \frac{bd^*(gs_1+s_2)}{(fs_1+s_2)^2}\exp\left(\frac{bd^*(f-g)}{fs_1+s_2}\right)\right], & f < g, \\ 0, \quad f = g, & \end{cases}$$

2.4.2 Dynamic Tuning Summary

This section explored an algorithm to reduce dissonance in a combination of notes. The next chapter will discuss how information from this section and the rest of the chapter can be used in practice.

3 Implementation

3.1 Reading and Preparing MIDI Notes

3.1.1 Parsing

The custom program for this project is written in the Python programming language. Python was chosen due to Mido[4], which is a convenient library to deal with messages in a MIDI file. The program takes in a MIDI file, which can be found on websites such as BitMidi[5].

To store information on the played notes, my custom 'NoteMessage' class is used. The 'NoteMessage' class has the following attributes:

- Note number.
- When the note was turned on.
- The 'Note On' velocity.
- When the note was turned off.
- What channel the event was on.
- What track the note is played on.

When parsing through the MIDI file, a list of 'NoteMessage' objects is created for every track chunk. When a 'Note On' MIDI event is read, a new 'NoteMessage' instance is created. The data within the 'Note On' message for channel, note number, and velocity are saved to the attributes as part of the object's constructor, along with the track number it was played on, and the total delta time that has elapsed before the event. When a 'Note Off' message is read, the program looks for the last 'NoteMessage' that was played with the same note number, on the same track and channel. The note off time attribute of that 'NoteMessage' instance is then set to the total delta time passed before the 'Note Off' message was read.

My program keeps track of the total delta time that has passed, by summing the delta time in every MTrk event for every track chunk. This total delta time is reset to 0 for every new track. However, the maximum delta time is stored to a new variable, so that the length of the tune can be calculated for debugging purposes. $Time\ (in\ ms.) = (Number\ of\ Ticks) * (Tempo\ (\mu s/qn)\ /\ Div\ (ticks/qn))\ /\ 1000$, where tempo is read from the 'Set Tempo' meta-event, as described in Meta-Events, and division is read from the header chunk, as described in Header Chunk.

The program also reads 'Track Name' or 'Instrument Name' events, as discussed in Meta-Events. If either event contains a substring of 'drum', the 'NoteMessage' list it is contained in is filtered out of the final list passed to the algorithm. This is because drum sounds do not follow the pattern of most instruments of being described as notes that have a pitch, and they are not tuned to a specific frequency. While they are represented by note numbers in MIDI files, these note numbers do not correspond to the notes in Table 4. The synthesizer would assign the notes to sounds such as the 'snare' or 'hi-hat'. As the MIDI tracks relating to the

[4] Source Code: https://github.com/mido/mido/ Documentation: https://mido.readthedocs.io/
[5] https://bitmidi.com/

playing of drums carry different information to the other tracks, they would skew the general key of the song, without adding valuable information, in relation to tuning of individual instruments.

Figures 8, 9, 10, and 11 show examples of plots generated to show the MIDI note spread across different tracks in various MIDI files. These graphs will be used as a reference for the range of notes in tracks by comparisons between tuning systems. Some songs, such as in Figure 11 are more chaotic, with tracks that span a large variety of notes. The song in Figure 8 has much less tracks, and the notes tend to have a smaller range.

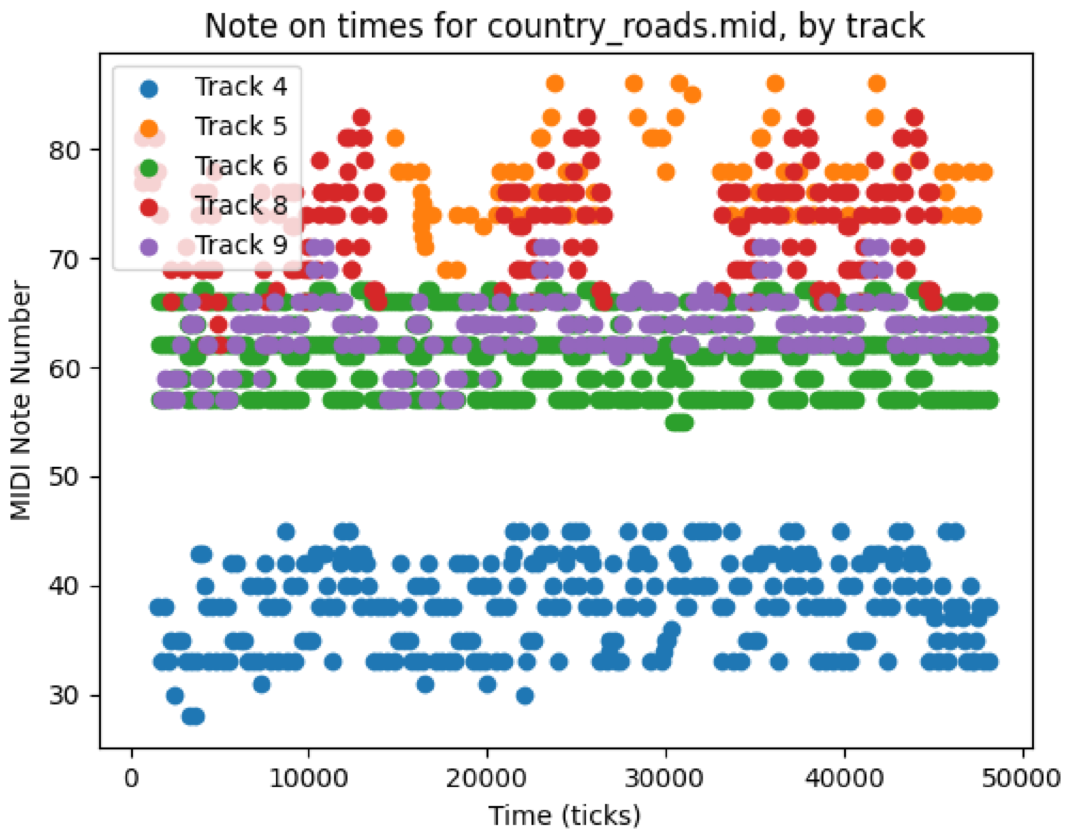


Figure 8 Notes from a MIDI adaptation of John Denver - Take Me Home, Country Roads.

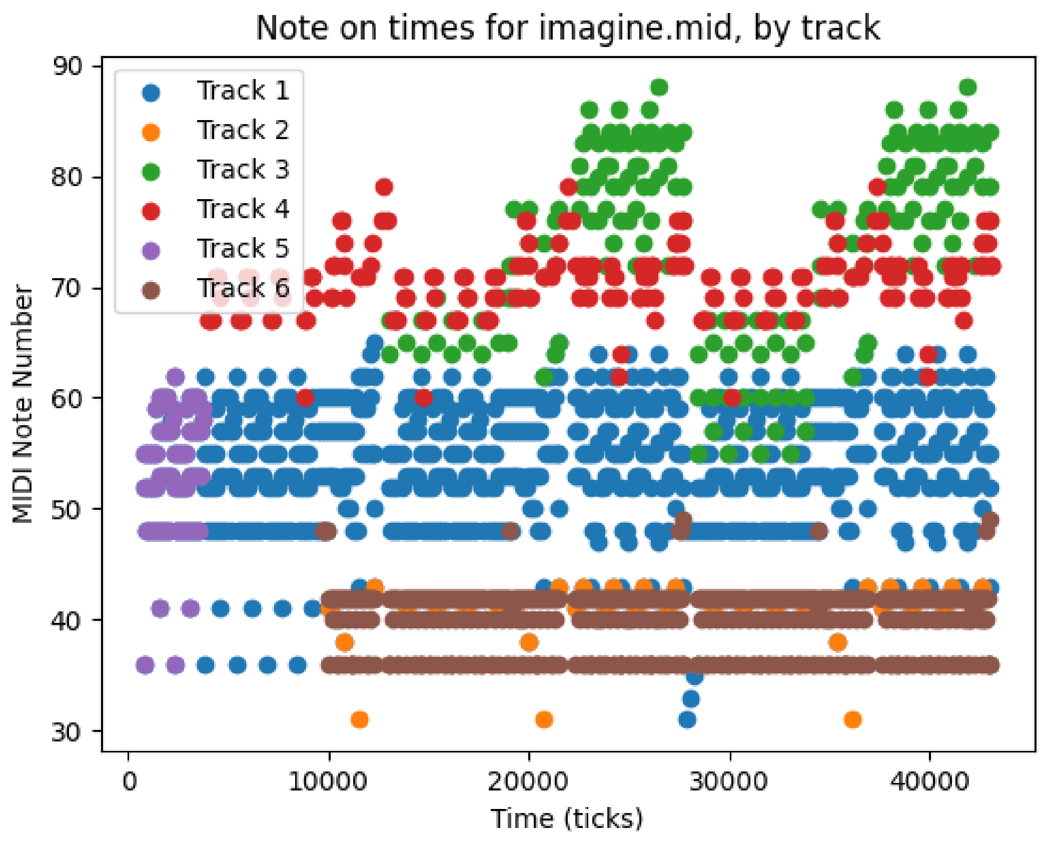


Figure 9 Notes from a MIDI adaptation of John Lennon – Imagine.

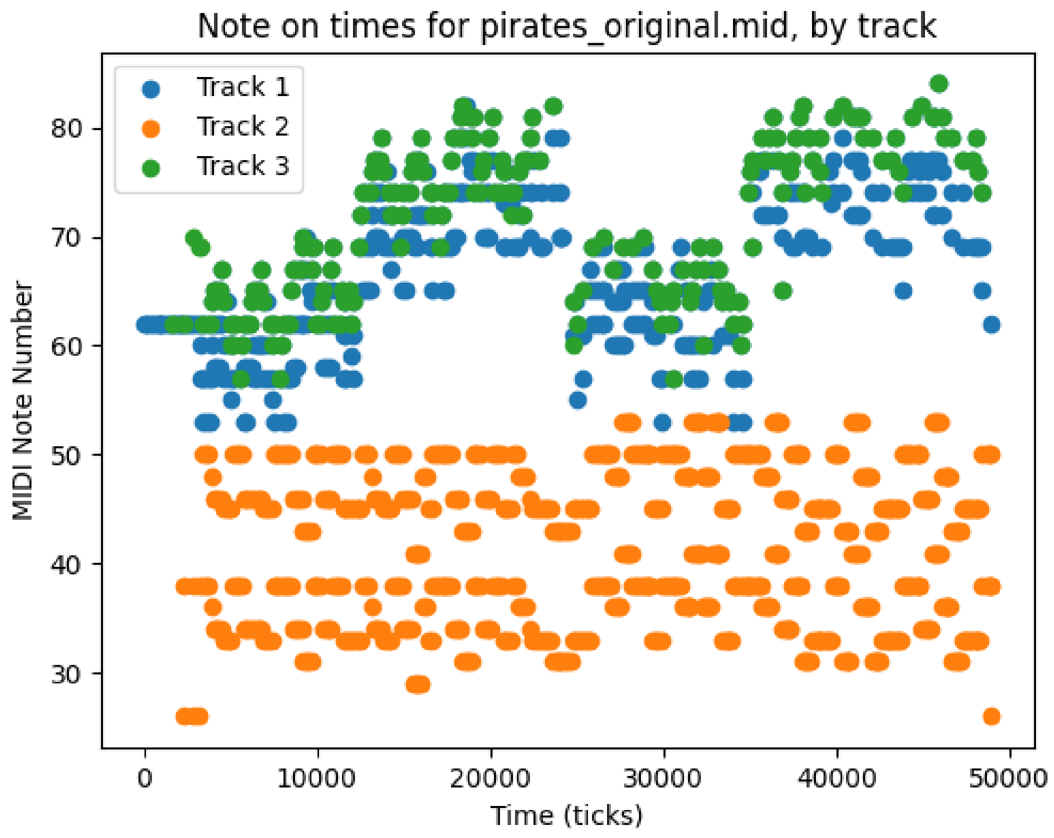


Figure 10 Notes from a MIDI adaptation of Klaus Badelt - He's A Pirate.

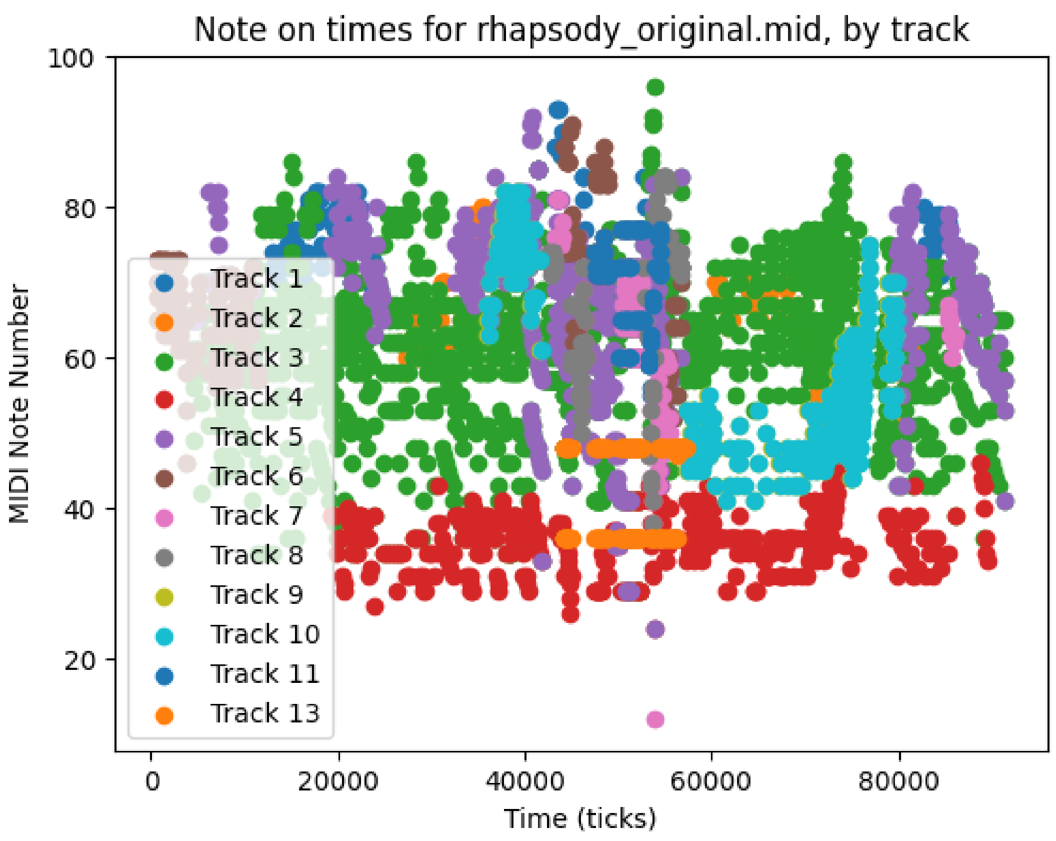


Figure 11 Notes from a MIDI adaptation of Queen - Bohemian Rhapsody.

3.1.2 Simultaneous Notes

When the MIDI file has been read to the finish, and all 'Note On' and 'Note Off' events have been assigned to a 'NoteMessage', the list of objects has to be sorted so that notes played at the same time can be tuned to be more harmonious. This is the case when the note_on_time attribute of a 'NoteMessage' is equal to another played on the same instrument track. Simultaneous notes are grouped by this attribute, within the list of notes in a track, and then sorted again so that the notes are in ascending order within the groups.

Figures 12, 13, 14, and 15 show examples of plots generated to show simultaneous notes played in MIDI files. This showcases the program's ability to group notes. They are also useful in determining how notes are related to each other and for spotting trends for how many notes are grouped and over which range in a song. The number and range of simultaneous notes in Figure 15 are much larger than in Figure 12. This indicates that the latter may see more of a dissonance reduction.

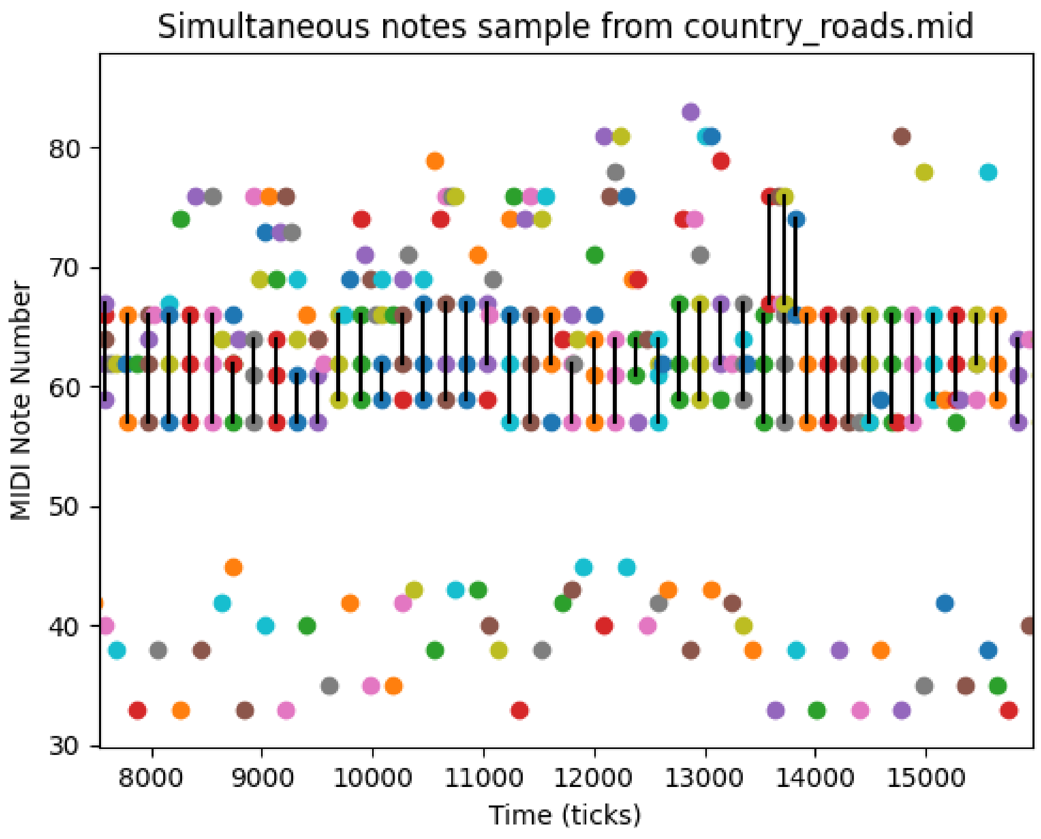


Figure 12 Sample of simultaneous notes highlighted from a MIDI adaptation of John Denver - Take Me Home, Country Roads.

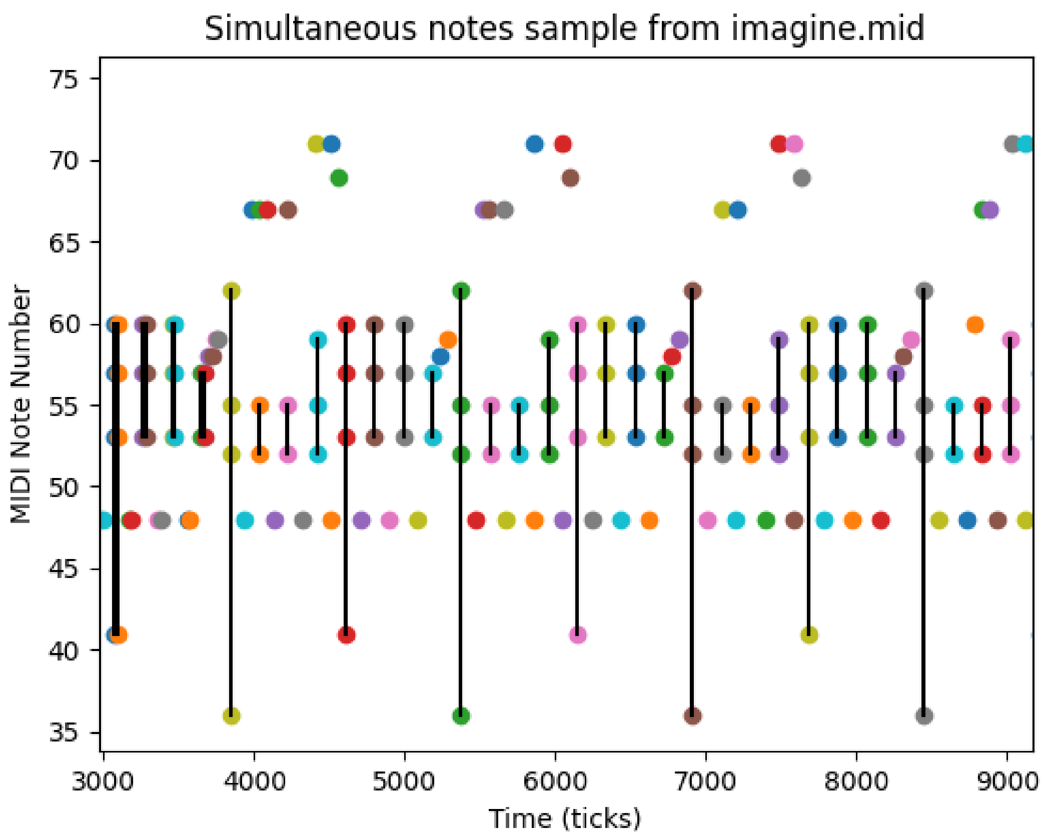


Figure 13 Sample of simultaneous notes highlighted from a MIDI adaptation of John Lennon – Imagine.

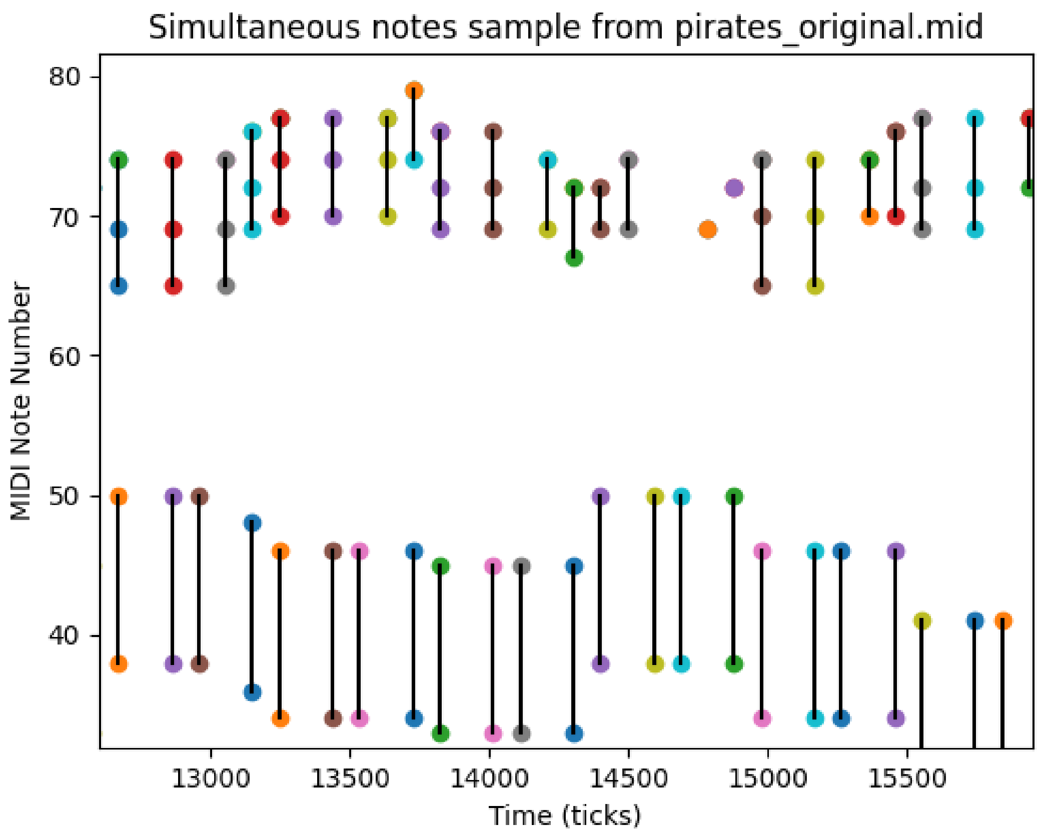


Figure 14 Sample of simultaneous notes highlighted from a MIDI adaptation of Klaus Badelt - He's A Pirate.

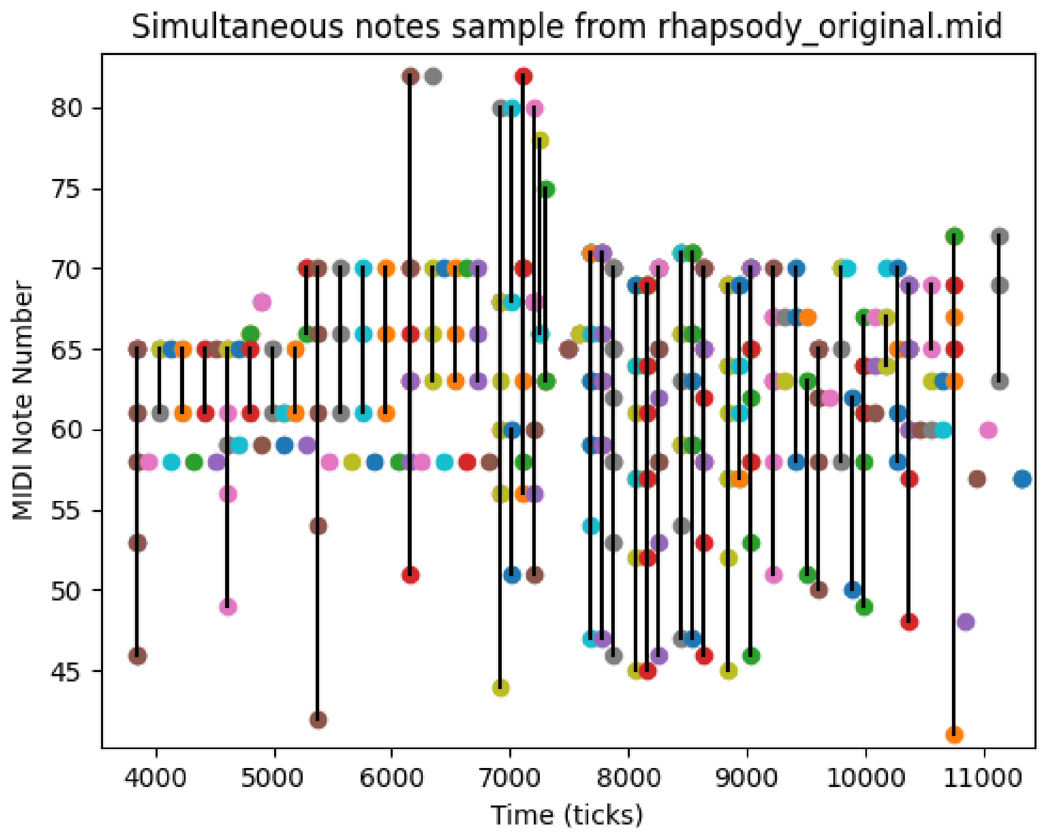


Figure 15 Sample of simultaneous notes highlighted from a MIDI adaptation of Queen - Bohemian Rhapsody.

3.2 ALGORITHM

3.2.1 Baseline Frequencies and Cents

My algorithm works off the baseline frequencies of equal temperament. These frequencies are shown in Figure 16. This plot also showcases the exponential scaling of frequencies as the notes increase, which is the reason for the logarithmic scaling of cents. As the MIDI note number increases, the frequencies scale more. The adjustments to frequency tunings need to adjust according to this.

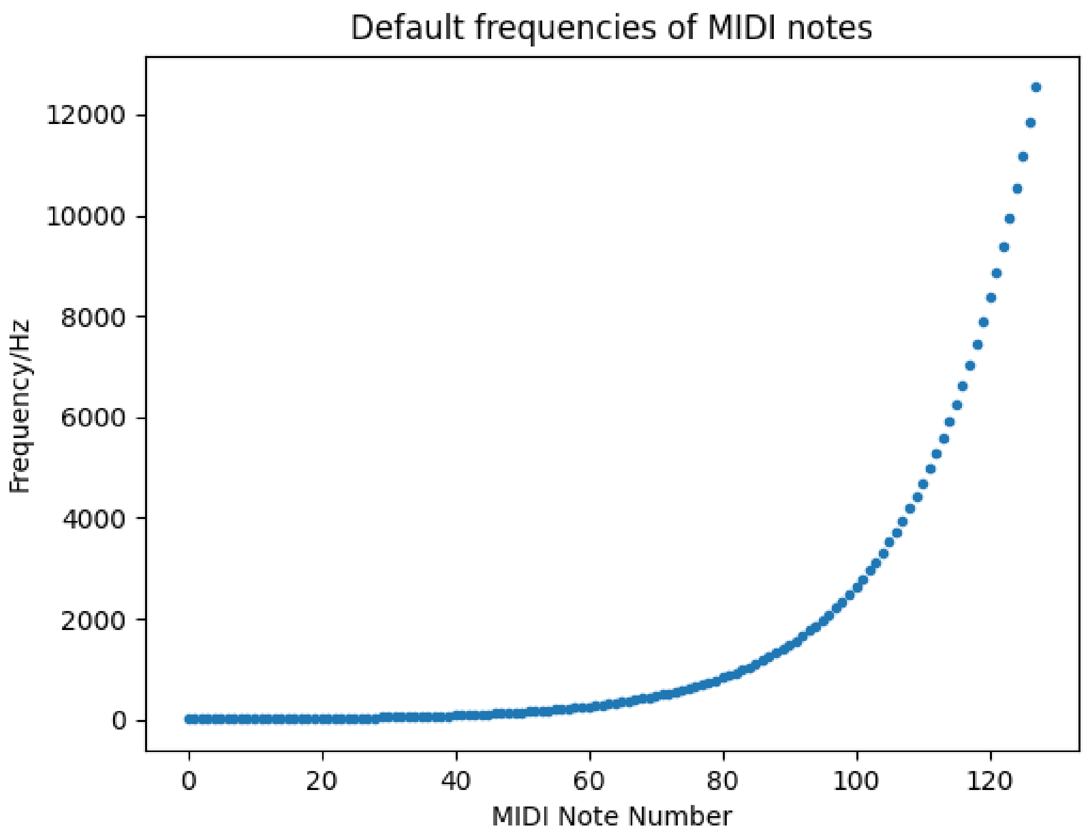


Figure 16 Baseline frequencies used before tuning.

3.2.2 Tuning track by track

The tuning algorithm of Equation 4 is applied track by track. The step size is adjusted based on the distribution of the simultaneous notes played within a track. The gradient calculation will be larger for higher dimension, large variance gradient calculations, e.g. if there are 4 notes played at once, ranging over more than an octave. To limit this from happening, a maximum change of 4 cents is enforced. This way the wide variation in the frequencies cannot be blown out of proportion by one or two large gradients, as the change will be reduced. Cents are used so that the change is proportional to the exponential frequency scaling seen in Figure 16.

Some experimental tweaks were made to fine tune the algorithm:

- If the change of a note frequency exceeds a threshold, the rate of change is limited e.g. a total change over multiple iterations that would result in a frequency varying by over 50 cents will be limited. This tweak was kept, to keep the songs' melodies.
- If a change in frequency results in higher dissonance, the change will first be inversed, so that the negative change to a frequency will instead be positive. If this change results in improved dissonance the change is kept, otherwise no change is made. This tweak proved useful to decrease dissonance in a tune and was kept.
- As the partials of the final synthesizer are unknown, the same values as in Table 7 are used to create the timbre of the tuning frequencies. This tweak was kept, to reflect

the application to a synthesizer more truly than just comparing dissonance between pitches.

- I tried to fix a random note for every chord of simultaneous notes with the intent that if a frequency is fixed and others adapt relative to it, the algorithm will execute quicker. However, this did not yield much improvement, and I removed it.
- I used a logarithmic step size scaling before the implementation of a maximum cents change. My program indexed an array of step sizes based on the current note played, so that larger frequencies see proportional changes. This became obsolete with the implementation of limiting change to 4 cents and was removed.

My algorithm stops iterating over a note pitch, when all note pitches at one point in time change less than 0.1 Hz from their previous iteration. When all simultaneous notes in all tracks meet this criteria, the scale file and keyboard mapping files are generated, as discussed in section 2.2.3. Unique pairs of these files are generated for every track, so that they can be applied on an instrument-by-instrument basis.

As these tuning files only allow one frequency for every note per file, a lot of my tuning algorithm's potential is wasted. It has the potential to provide the lowest dissonance frequency combinations for notes at any point of a song, however the final implementation used by a synthesizer only uses one of these frequency combinations. The algorithm operates on the basis that a note frequency is retuned in real time. This issue arises due to a lack of options for MIDI note retuning. The new MIDI Polyphonic Expression (MIDI Manufacturers Association, 2018) proves to be a promising option to improve the algorithm's effectiveness when applied to a synthesizer and exploring the application of this could be the next step for future work.

3.2.3 Comparison Metrics

In order to quantify the success of the algorithm, the dissonance measure of Equation 3 is calculated for notes played at all points of the song, before any adjustments to frequency are made, and again after. The dissonance sum across all simultaneous notes can then be compared between tuning systems to find which gives the smallest overall dissonance.

As an experiment, the frequencies resulting from dynamic tuning are compared track by track to the dissonance of a universal equal temperament tuning such as in Figure 16, and a just intonation tuning, tuned relative to C. Figure 17 shows the just intonation frequencies used for comparison to the dynamic tuning. The intervals used are those in Table 6, and tuned so that C is the unison/octave interval. The comparison will act as a control to show if my implementation of dynamic tuning will also have reduced dissonance compared to other tunings than its baseline.

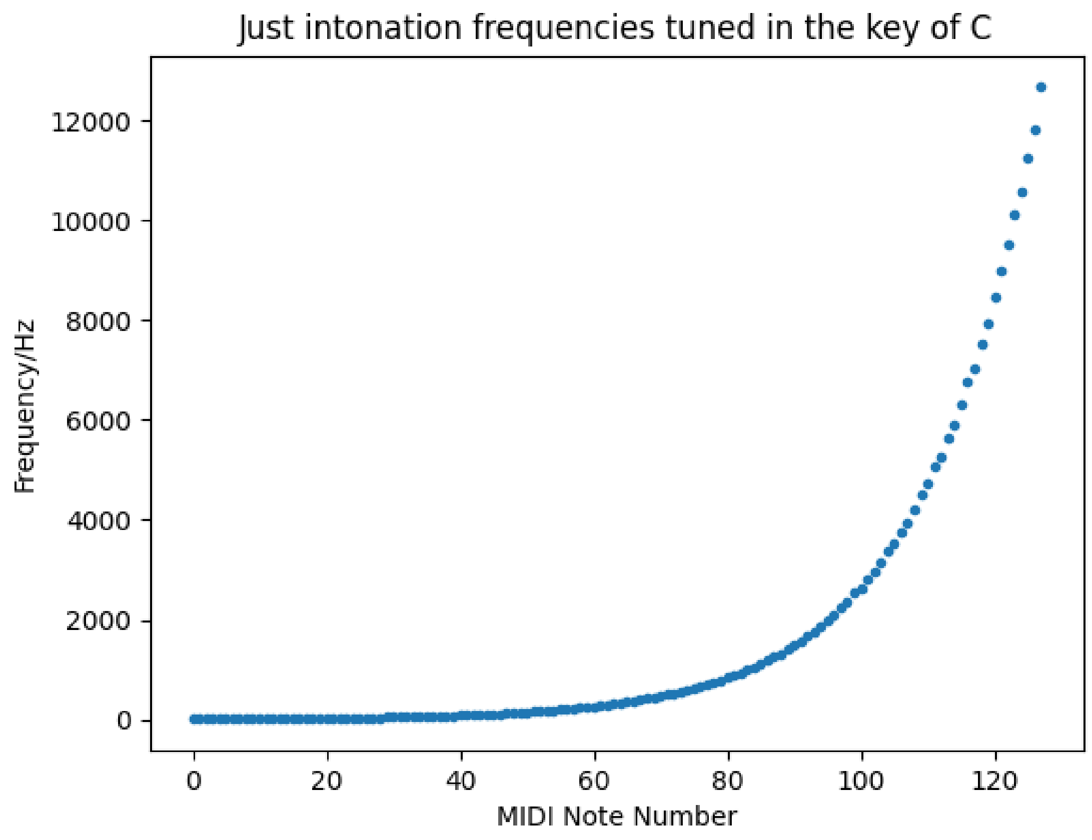


Figure 17 Just intonation frequencies used for comparison.

3.2.4 Hyperparameter Tuning

Parameters such as the value of maximum change in cents per iteration can be evaluated with the dissonance measure. The reason cents are used, is because of the exponential scaling of frequency for the notes; a 1 Hertz frequency change is more significant at lower frequencies than higher frequencies. Figure 18 and Table 8 show the difference between dissonances when changing the maximum cent change value. It is shown that 4 cents per iteration have the lowest resulting dissonance for this tune, and this result is similar in other tunes. 4 cents are close to 0.5Hz, when the fundamental frequency is 200 Hz; around 1 Hz when it is in the 400s; over 2 Hz in the 900s etc. Limiting the frequency change to 4 cents allows the algorithm to find a local dissonance minimum, which may not be found when large changes are made at once. It also stops the gradient from exploding after multiple iterations of large changes.

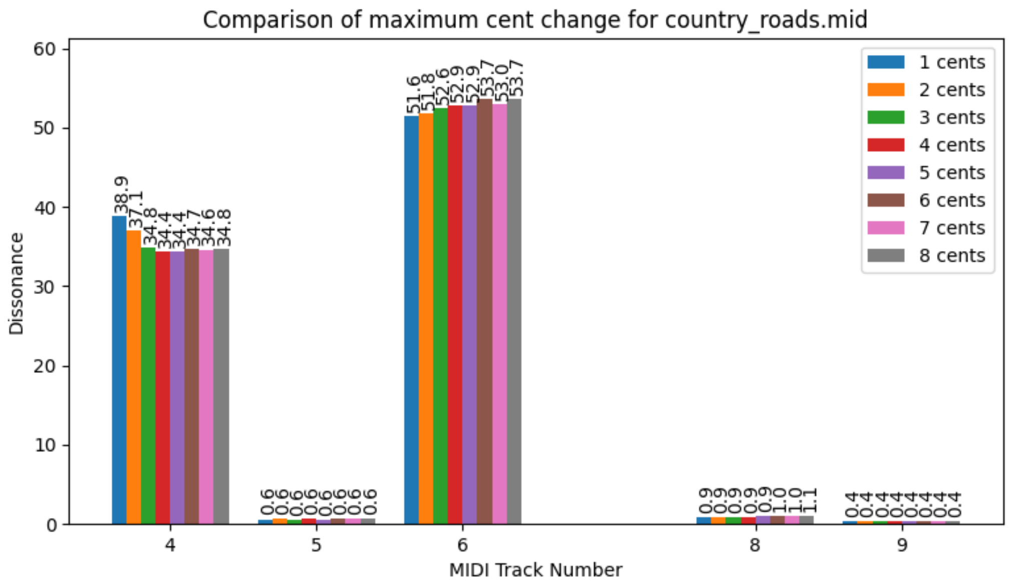


Figure 18 Comparison of impact from changes in the maximum cent change hyperparameter to total dissonance.

Table 8 Comparison of impact from changes in the maximum cent change hyperparameter to total dissonance.

Track	**Maximum cent change**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
4	38.88	37.10	34.85	34.41	34.42	34.69	34.64	34.80
5	0.61	0.61	0.60	0.62	0.60	0.65	0.62	0.63
6	51.55	51.76	52.56	52.86	52.87	53.66	53.03	53.66
8	0.89	0.89	0.93	0.93	0.95	1.03	1.02	1.06
9	0.42	0.42	0.42	0.42	0.42	0.43	0.42	0.43
	92.36	**90.79**	**89.35**	**89.25**	**89.26**	**90.47**	**89.73**	**90.58**

3.3 MIDI in Practice

It is now possible to take the original MIDI file, along with the generated scale file (scl) and keyboard mapping (kbm) files, and apply them to a software synthesizer, such as those mentioned in section 2.1.1. I tested this with success using the Surge plugin inside the REAPER[6] digital audio workstation.

3.4 Implementation Summary

Now that the method of my dynamic tuning algorithm was explained, the next chapter will explore and discuss the results of its application. It will be compared to the baseline, twelve-tone equal temperament, and a control, just intonation, tuned relative to C.

[6] https://www.reaper.fm/

4 RESULTS & DISCUSSION

4.1 DISSONANCE COMPARISON

4.1.1 Results

The following figures and tables showcase the dissonances calculated between all simultaneous notes on a track by track basis. Observations will be made after each figure and table pair, with a more general and detailed discussion following.

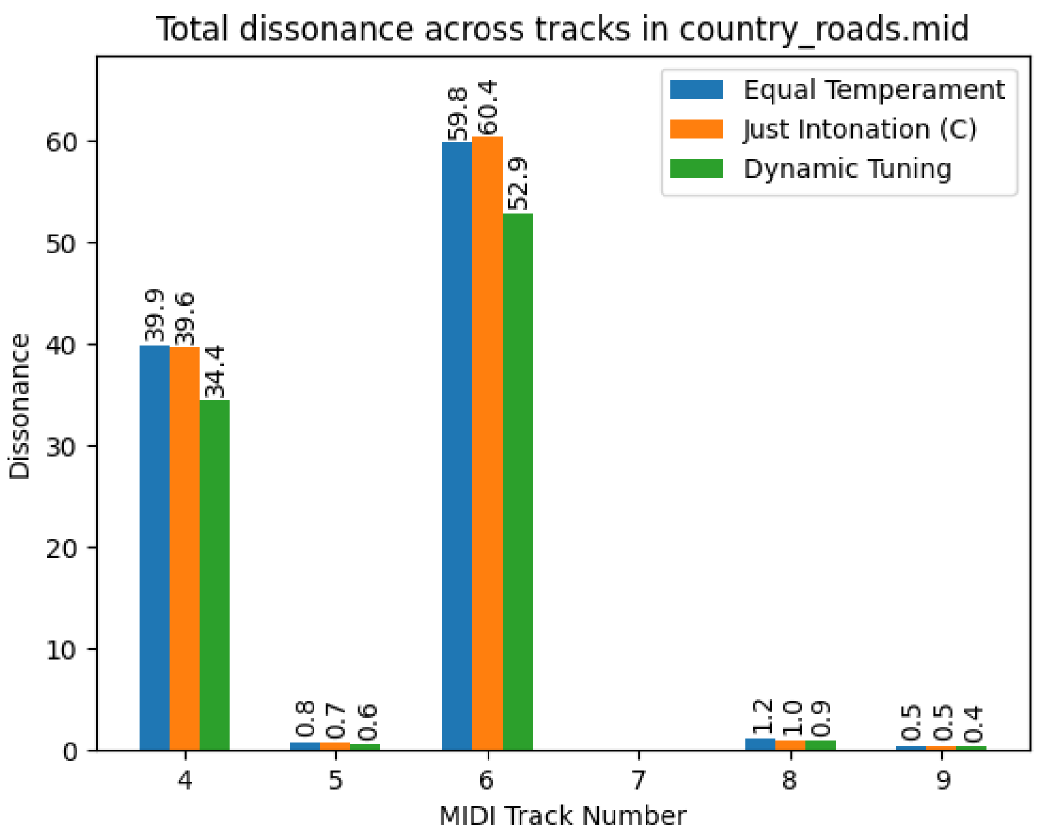


Figure 19 Dissonance comparison for a MIDI adaptation of John Denver - Take Me Home, Country Roads.

Table 9 Dissonance comparison for a MIDI adaptation of John Denver - Take Me Home, Country Roads.

Track	Equal Temperament	Dynamic Tuning	Just Intonation (C)
4	39.89	34.41	39.59
5	0.82	0.62	0.74
6	59.78	52.86	60.44
8	1.24	0.93	1.01
9	0.49	0.42	0.48
	102.20	**89.25 (-12.67%)**	**102.26 (+0.05%)**

Figure 19 and Table 9 show large relative decreases in the dissonance when using my tuning approach – Dynamic Tuning. It saw a total 12.67% decrease compared to equal temperament, with track 4 and track 6 having a high proportion of the decrease. Track 4 and track 6 are the biggest tracks in the MIDI file. Therefore, they make up 12.4 out of the 12.95 dissonance decrease, or 95.75%. The control, just intonation, has a similar total dissonance than equal temperament. Track 4 is labelled as a 'Bass' track; track 5 as 'Guitar'; track 6 also as 'Guitar'; track 8 as 'Violin'; track 9 as 'Melody'.

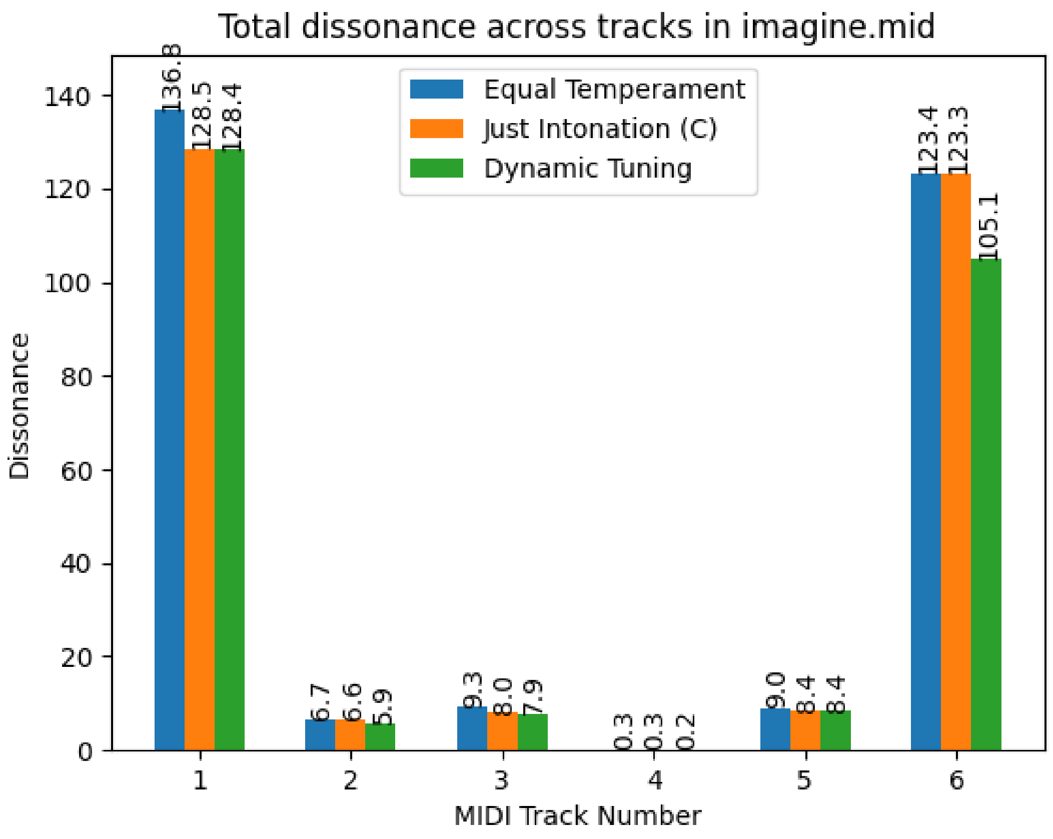


Figure 20 Dissonance comparison for a MIDI adaptation of John Lennon – Imagine.

Table 10 Dissonance comparison for a MIDI adaptation of John Lennon – Imagine.

Track	Equal Temperament	Dynamic Tuning	Just Intonation (C)
1	136.85	130.11	128.52
2	6.70	5.90	6.59
3	9.30	8.03	8.04
4	0.26	0.23	0.26
5	9.02	8.59	8.39
6	123.36	105.07	123.29
	285.49	**257.92 (-9.66%)**	**275.09 (-3.64%)**

Figure 20 and Table 10 show a MIDI file containing 6 tracks. The larger changes in dissonance happen in Track 1, 2, 3, 5, and 6. Dynamic tuning has a 9.66% reduction in dissonance over equal temperament. Of this change, track 1 and track 6 make up 6.74 and 18.29, respectively, out of the 27.57 total change. This is a 90.8% dissonance reduction from 2 of the 6 tracks. Here, just intonation also saw a dissonance reduction of over 3% compared to equal temperament, which is smaller than my dynamic tuning implementation by 6%.

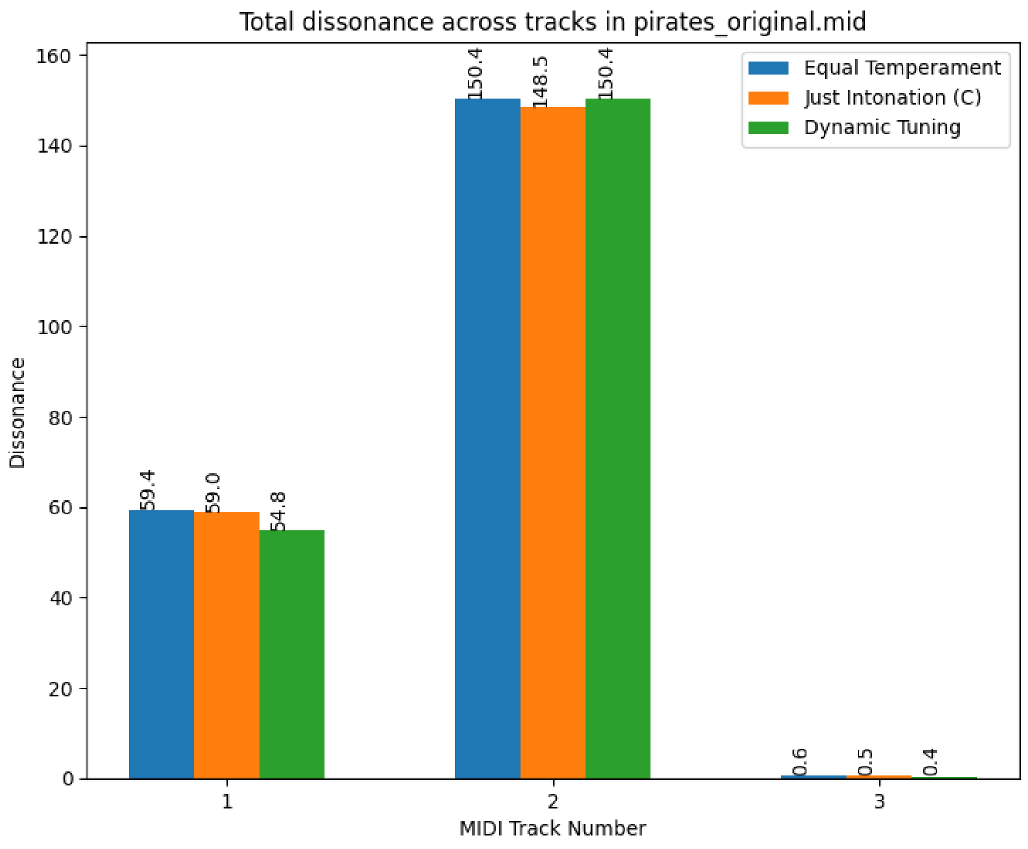


Figure 21 Dissonance comparison for of Klaus Badelt - He's A Pirate.

Table 11 Dissonance comparison for of Klaus Badelt - He's A Pirate.

Track	Equal Temperament	Dynamic Tuning	Just Intonation (C)
1	59.40	54.82	58.95
2	150.42	150.42	148.50
3	0.57	0.39	0.54
	210.39	**205.62 (-2.67%)**	**208.00 (-1.14%)**

In Figure 21 and Table 11, a song with 3 instrument tracks is examined. The tracks with significant changes to dissonance, track 1 and track 2, refer to the right hand and left hand of a piano, respectively, according to MIDI meta-events. For this song, dynamic tuning saw a small reduction in dissonance, 2.67%, compared to equal temperament. The just intonation scale used as a control had almost half of that reduction. In track 2, no reduction in dissonance is seen in dynamic tuning, so that all reduction is because of reducing dissonance from track 1. For comparison, in just intonation, the reduction is more evenly distributed.

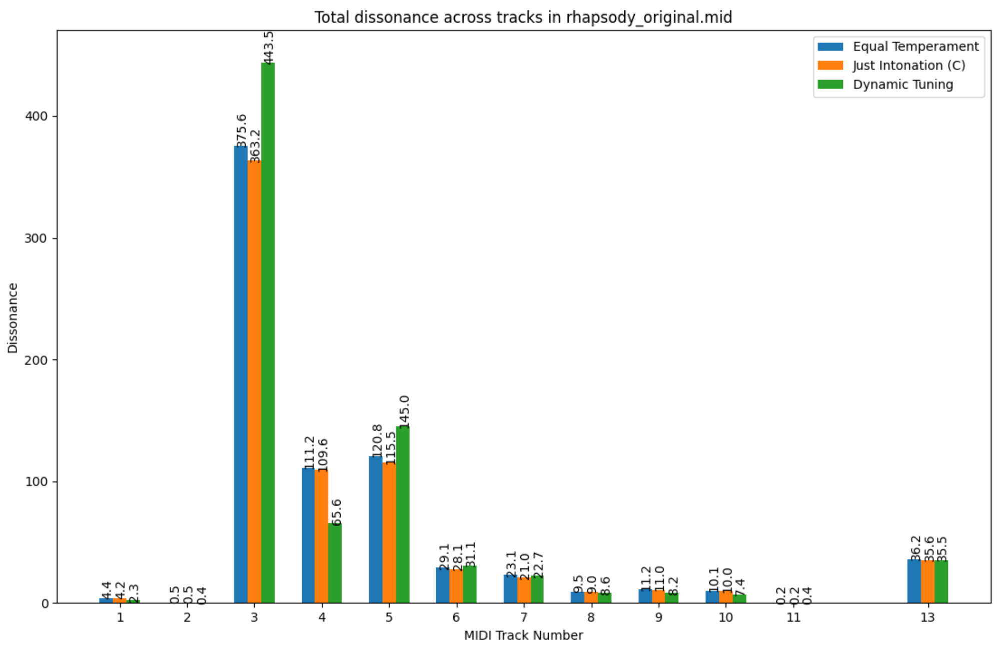


Figure 22 Dissonance comparison for a MIDI adaptation of Queen - Bohemian Rhapsody.

Table 12 Dissonance comparison for a MIDI adaptation of Queen - Bohemian Rhapsody.

Track	Equal Temperament	Dynamic Tuning	Just Intonation (C)
1	4.40	2.25	4.18
2	0.51	0.42	0.50
3	375.59	443.50	363.24
4	111.19	65.55	109.55
5	120.85	144.97	115.53
6	29.07	31.09	28.14
7	23.08	22.69	21.04
8	9.54	8.62	9.01
9	11.23	8.17	11.03
10	10.14	7.38	9.97
11	0.20	0.37	0.19
13	36.23	35.48	35.60
	732.02	**770.49 (+5.26%)**	**707.98 (-3.28%)**

Figure 22 and Table 12, show the comparison of dissonance change of a complex song, with 12 different tracks. These tracks, in order, are labelled as ‘Lead Vocalist’, ‘Lead Vocalist 2’, ‘Piano’, ‘Bass’, ‘Strings’, ‘Choir’, ‘Brass’, ‘Horn’, ‘Lead Guitar’, ‘Lead Guitar Eko’, ‘Orchestra Hit’, ‘Timpani Drum’. Of these 12 tracks, 9 tracks have a dissonance over 3. Here dynamic tuning sees an increase in dissonance over equal temperament. While there is a decrease in most tracks, track 3, track 5, and track 6 see increases large enough to shift the total into positive numbers. Without a change in these 3 mentioned tracks, dynamic tuning would have reduced dissonance by 55.48, or 7.58%. Just intonation sees a decrease in dissonance compared to equal temperament, however, in tracks other than track 3, 5, and 6, dynamic tuning has less dissonance.

4.1.2 Discussion

In general, the algorithm manages to fulfil its purpose of reducing dissonance across all tracks. The largest relative dissonance decrease is seen in Figure 19 and Table 9. Here dissonance decreased by over 12% over 5 tracks. The biggest changes in dissonance are seen in Track 4 and Track 6, which are labelled as bass and guitar tracks respectively. As can be seen in Figure 8, they are both tracks that span about an octave. My dynamic tuning system seems to work well for tracks with a small variety and range of notes. It could therefore be suited to tuning guitar and bass tracks, as these instruments span less octaves in the real world than, for example, pianos.

Otherwise, the smallest dissonance decrease is shown in Figure 21 and Table 11. This could be due to the file being simpler, leaving little room for improvement. There is no change shown in track 2, which corresponds to the 'left hand' of a piano according to the meta-messages. This track only plays octaves, i.e. notes 12 semitones apart. Figure 7 shows that octaves have the lowest possible dissonance already, and the equal temperament tuning already uses a just interval octave, as seen in Table 6. Therefore, no improvement can be made to track 2.

On average, my dynamic tuning solution results in about a ~5% decrease in dissonance. This improves to an over 8% decrease in dissonance when the 3, out of 26, outlier tracks that saw an increase are left unchanged.

The only file that resulted in dissonance after dynamic tuning, is in Table 12 or Figure 22. This could be down to the song being quite chaotic when compared to the others used for testing. The largest discrepancy can be found in Track 3, which is the piano track, according to the file's meta messages. Figure 11 shows that this is an expansive track, ranging over 40 – 50 notes. It seems that the algorithm does not perform well under these circumstances.

The dissonance comparisons show that the algorithm is better at decreasing dissonance in a song when the range of notes is small. For tracks where equal temperament tuning is already close to the just interval, for example the octave and major fifth, the total dissonance of equal temperament and dynamic tuning will be similar.

An issue with using the dissonance measurement method for comparing between equal temperament, just intonation and the dynamic tuning is that it only considers the vertical relationship between notes. The dissonance is only calculated between notes that are played at the same time, but ignores adjacent notes, that are played close in time and may still be in the listener's ear. It is possible that when dissonance between notes adjacent in time is considered, the results could differ.

Another caveat of this method is the use of estimated partials, as in Table 7. For a more exact measurement of dissonance, on a synthesizer by synthesizer basis, the timbre of notes of each should be examined. It is possible that the outcome of the pitch adjustments will differ if the partials used by the instrument are different.

4.2 Example Changes

4.2.1 Results

This section will inspect frequency changes of different tracks of MIDI files after they were dynamically tuned by my program. These tracks were also explored in section 4.1.1. To visualize the changes, a small range of the most frequent notes, within the tracks with the biggest changes in dissonance, of each tune, will be examined. Notes without change will be omitted from the tables, as they were not used in the tracks at all. Observations will be made after each figure and table pair, with a more general and detailed discussion following.

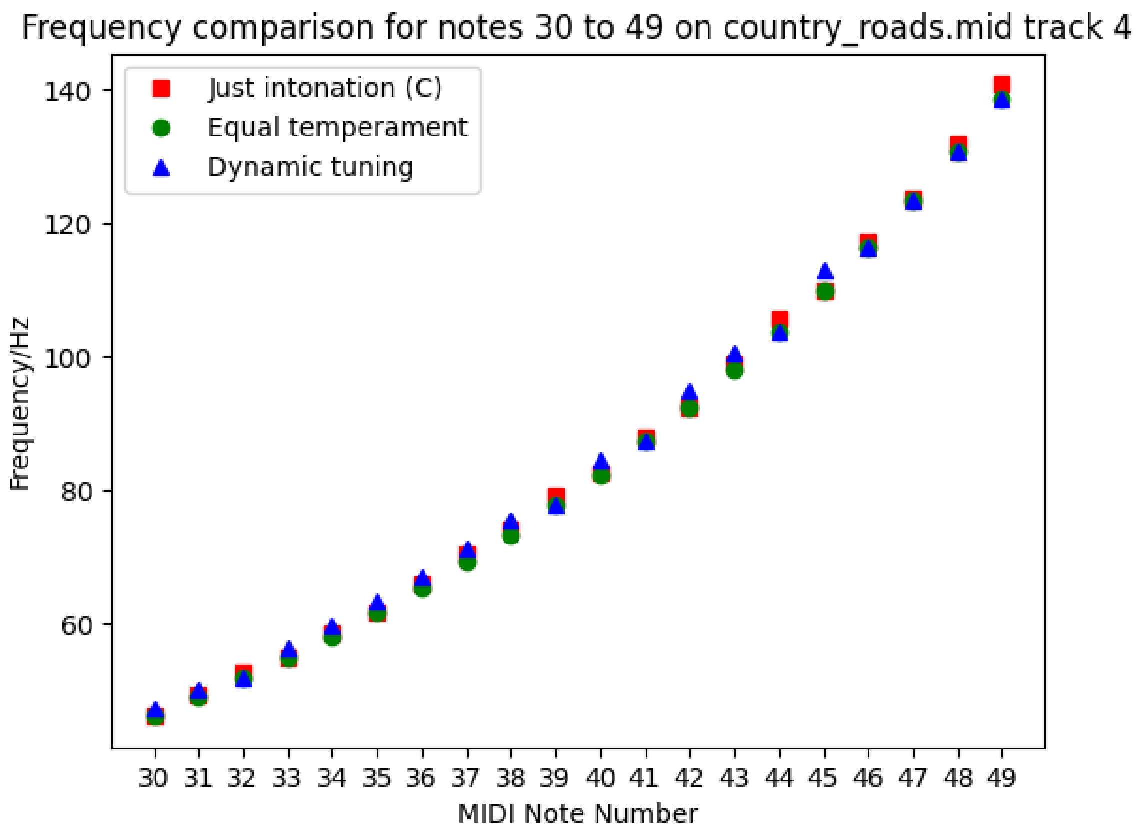


Figure 23 Frequency comparison in track 4 of John Denver - Take Me Home, Country Roads.

Table 13 Frequency comparison in track 4 of John Denver - Take Me Home, Country Roads.

Note	**Frequency/Hz**	**Difference compared to baseline (12-TET)**
30	47.55	1.30
31	50.38	1.38
33	56.55	1.55
34	59.91	1.64
35	63.47	1.74
36	67.25	1.84
37	71.24	1.95
38	75.48	2.06
40	84.72	2.32
42	95.10	2.60
43	100.75	2.76
45	113.09	3.09

Figure 23 and Table 13 show notes between note 30 (F♯1) and 45 (A2). To reduce dissonance, my dynamic tuning algorithm, has increased the frequencies of the notes compared to twelve-tone equal temperament. The increase in frequencies, is in proportion to the magnitude of frequencies. Note 30 saw an increase of 1.3 Hz, to a frequency of 47.55. Note 45 is over twice the frequency, and saw over twice the frequency change as well, a change of 3.09 Hz.

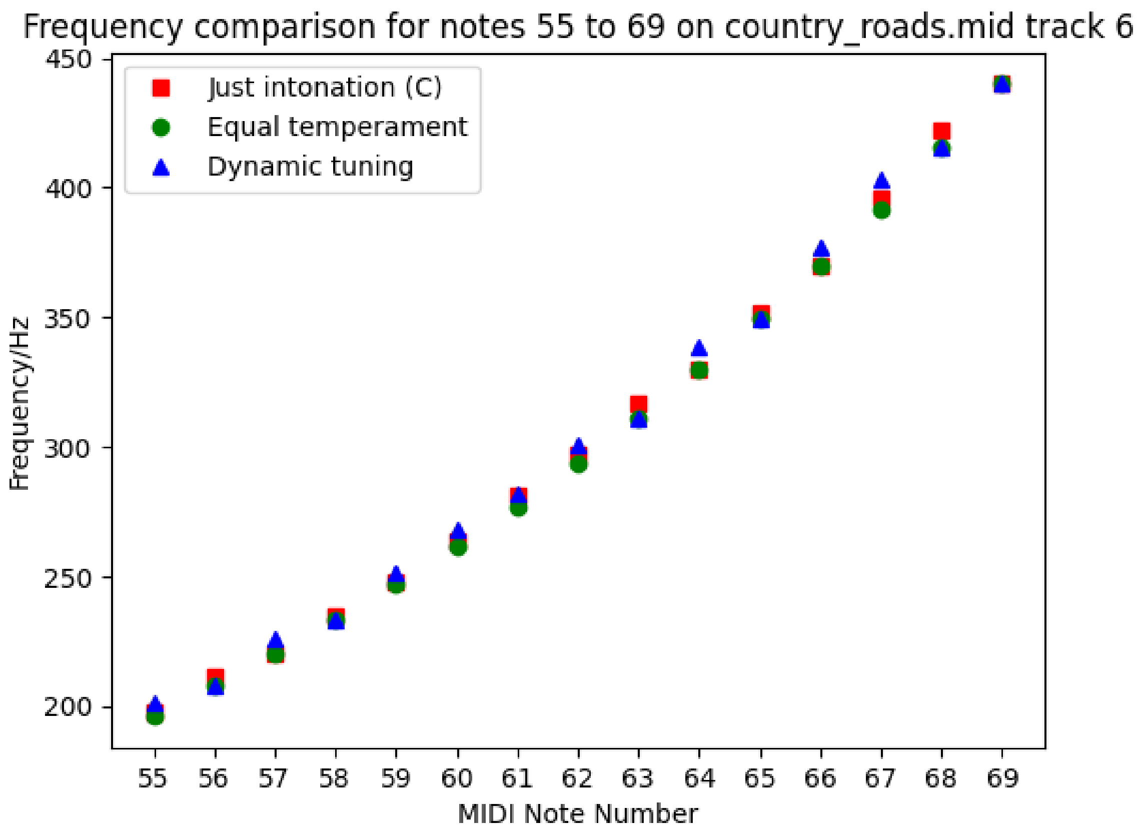


Figure 24 Frequency comparison in track 6 of John Denver - Take Me Home, Country Roads.

Table 14 Frequency comparison in track 6 of John Denver - Take Me Home, Country Roads.

Note	**Frequency**	**Difference compared to baseline (12-TET)**
55	201.51	5.51
57	226.14	6.14
59	251.43	4.48
60	268.36	6.73
61	282.29	5.11
62	301.14	7.47
64	338.86	9.23
66	376.75	6.76
67	402.96	10.96

The examined notes in Figure 24 and Table 14 range from MIDI note 55 (G3) and note 67 (G4). These notes saw a variation in increases to reduce dissonance. The smallest increase was note 59, with 4.48 Hz. The largest increase was 10.96 Hz for note 67. Note 59 is played more in the track than note 67, as seen in Figure 8. As the sample size for note 59 is larger, it could be converging to a frequency closer to its original.

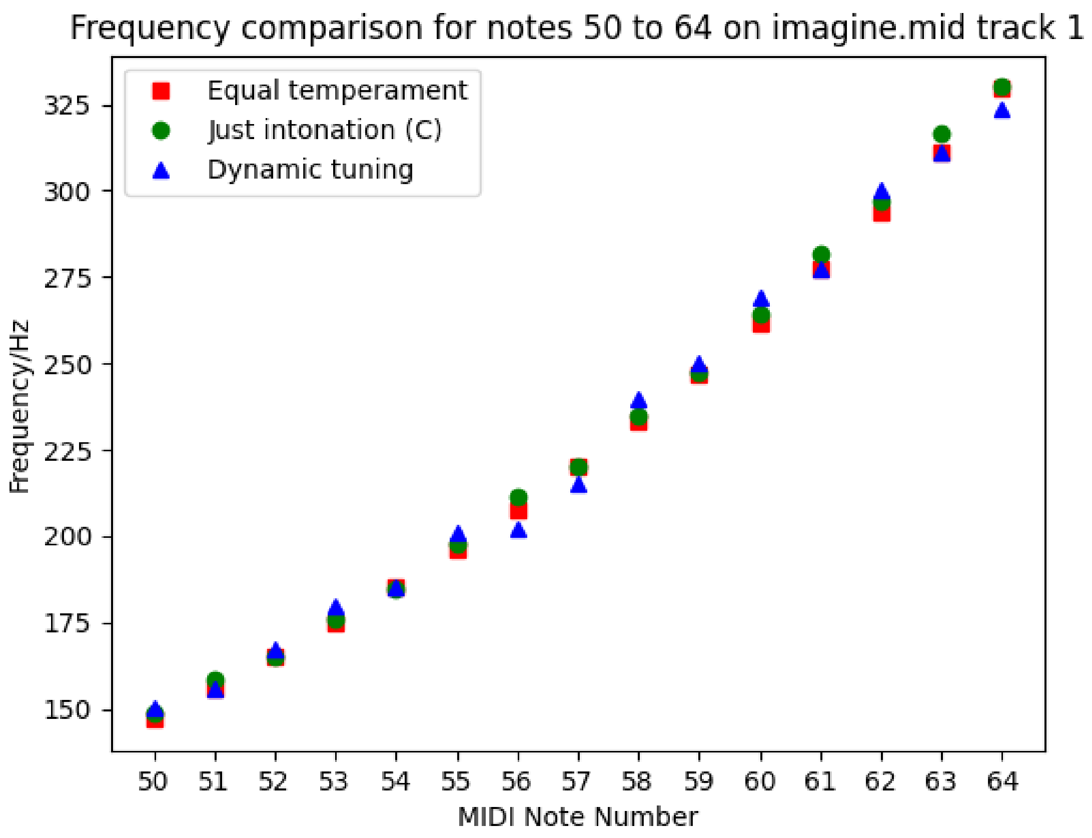


Figure 25 Frequency comparison in track 1 of John Lennon – Imagine.

Table 15 Frequency comparison in track 1 of John Lennon – Imagine.

Note	**Frequency**	**Difference compared to baseline (12-TET)**
50	150.25	3.41
52	167.41	2.59
53	179.49	4.88
55	200.93	4.94
56	201.92	-5.73
57	214.93	-5.07
58	239.63	6.55
59	250.26	3.31
60	268.91	7.28
62	300.42	6.75
64	323.53	-6.09
65	358.17	8.95

Figure 25 and Table 15, shows notes 50 (D3) to 65 (F4). To reduce dissonance, my tuning system has both increased and decreased frequencies. The changes range from -6.09 Hz to +8.95 Hz, to the equal temperament baseline. Notes with larger changes, such as note 64 and note 60, seem to be more regularly played in groups of 3 – 5 than notes with smaller changes such as note 59 and 52. The latter two notes are also often played alone.

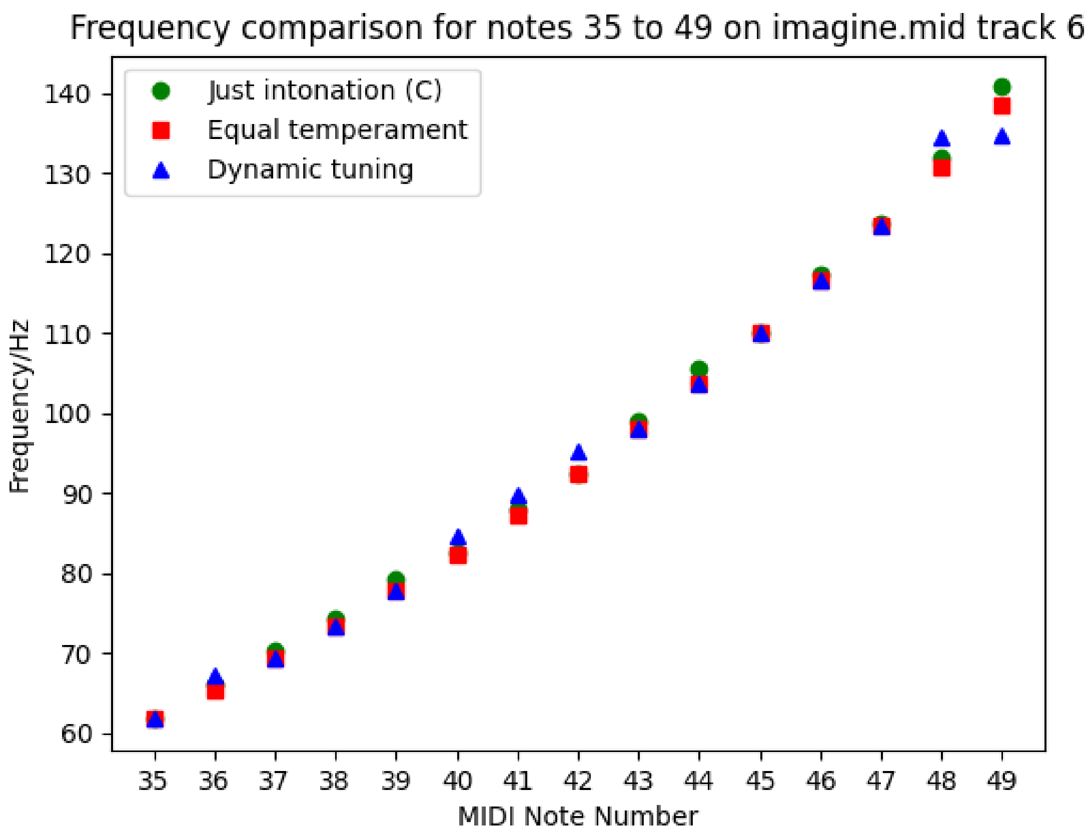


Figure 26 Frequency comparison in track 6 of John Lennon – Imagine.

Table 16 Frequency comparison in track 6 of John Lennon – Imagine.

Note	**Frequency**	**Difference compared to baseline (12-TET)**
36	67.34	1.94
40	84.72	2.32
41	89.76	2.45
42	95.25	2.75
48	134.49	3.68
49	134.79	-3.80

In Figure 26 and Table 16, most notes show increasing frequency. However, note 49 decreases to a similar frequency to note 48. These two notes are never played together, so the algorithm is not trying to create a unison interval. In fact, note 49 is only played twice, both times with note 36. Note 36 and 49 are 13 semitones apart. It seems that my algorithm is tuning their frequencies closer to being 12 semitones apart, by tuning 36 up and 49 down. This would be a just interval octave, the lowest dissonance interval, apart from unison.

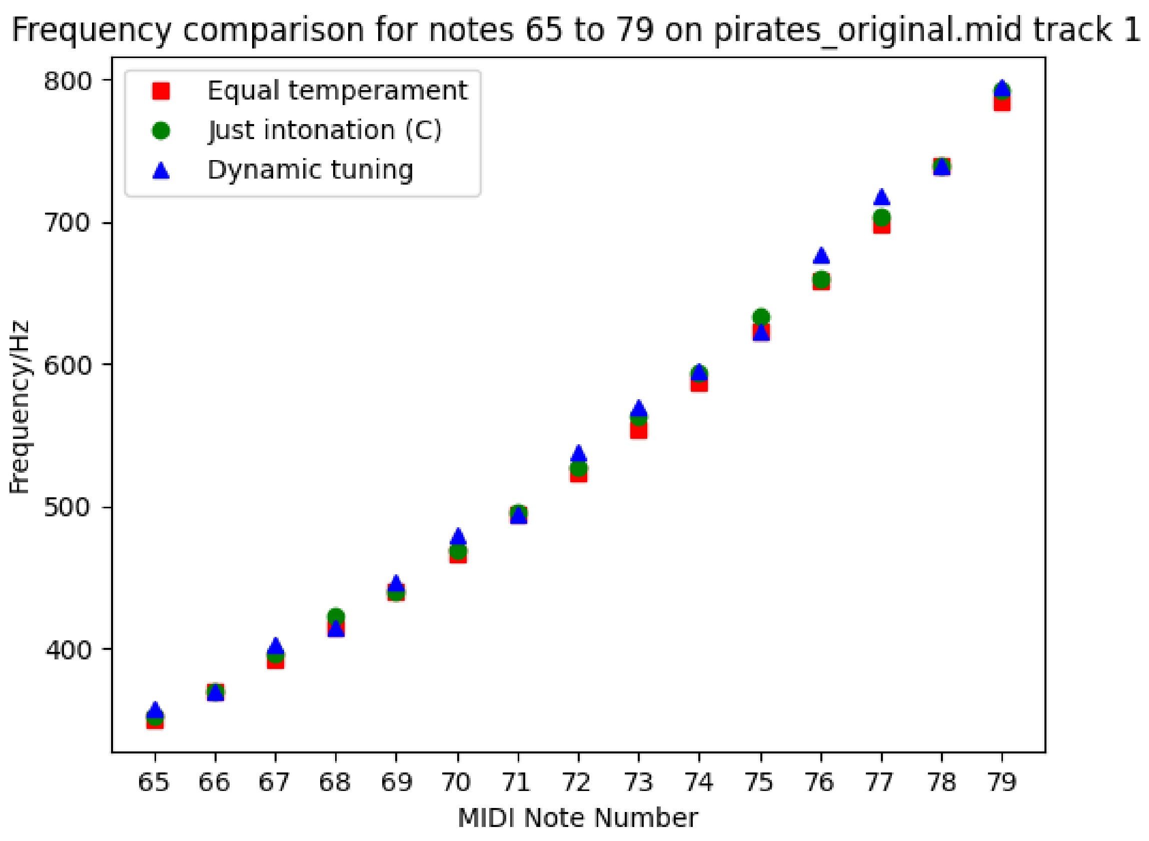


Figure 27 Frequency comparison in track 1 of Klaus Badelt – He's A Pirate.

Table 17 Frequency comparison in track 1 of Klaus Badelt – He's A Pirate.

Note	**Frequency**	**Difference compared to baseline (12-TET)**
65	357.33	8.10
67	403.00	11.01
69	446.83	6.83
70	479.21	13.04
72	537.93	14.68
73	569.95	15.58
74	595.25	7.92
76	677.73	18.48
77	717.97	19.51
79	794.69	10.70

Figure 27 and Table 17 frequencies from note 65 to 79 are shown. There is a pattern of the magnitude of frequency increase being proportional to frequency magnitude. However, notes 69 (A4), 74 (D4), and 79 (G5), do see smaller relative increases. These notes are close to their just intonation tuning, relative to C. The MIDI key signature meta message (see section on Meta-Events) mentions that this song is in the key of F major[7], however C major also contains

[7] Major scales contain notes 0, 2, 4, 5, 7, 9, 11 semitones above the key, i.e. all 'major' and 'perfect' intervals in Table 6.

the mentioned note intervals. I would propose that is a result of the frequencies falling into the just interval minima.

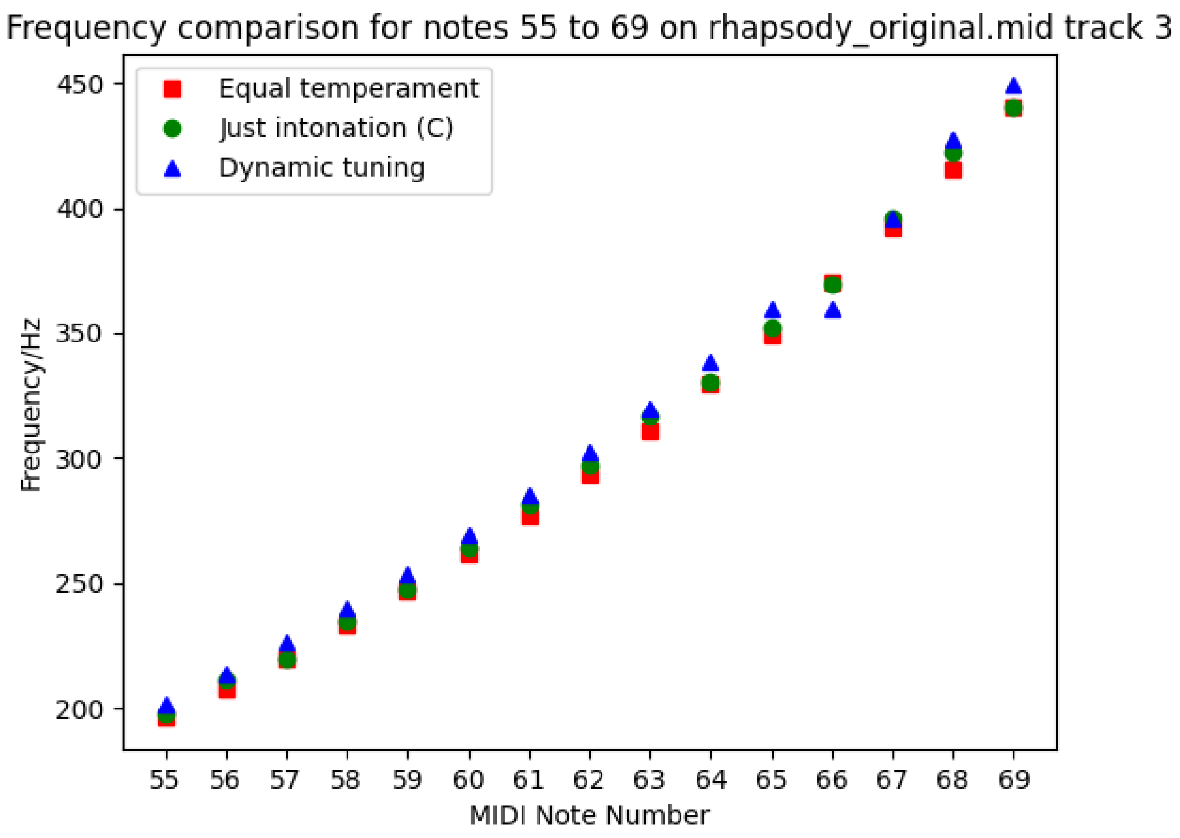


Figure 28 Frequency comparison in track 3 of Queen – Bohemian Rhapsody.

Table 18 Frequency comparison in track 3 of Queen – Bohemian Rhapsody.

Note	Frequency	Difference compared to baseline (12-TET)
54	179.92	-5.08
55	201.83	5.83
56	213.43	5.77
57	226.11	6.11
58	239.92	6.83
59	253.77	6.83
60	269.30	7.67
61	284.91	7.73
62	302.24	8.57
63	320.13	9.00
64	338.73	9.11
65	359.35	10.12
66	359.74	-10.25
67	395.86	3.87
68	427.62	12.32
69	449.03	9.03

Figure 28 and Table 18 show frequencies for notes in the range of note 54 to note 69. Most of these notes follow a pattern of being tuned up in frequency by my algorithm. The difference in frequency compared to 12-TET also scales relative to frequency magnitude. Notes 54, 66, 67, and 69 do not follow this trend. Notes 65 and 66 have close frequencies due to note 66 seeing a frequency decrease. These notes are never played simultaneously. The last instance of note 66 (F♯4), in this track is when it is played with note 54 (F♯3), 69 (A4), and 72 (C5) at delta time 87168. There are -12, 3, 6 semitones between note 66 and the others respectively. Out of these, the least dissonant interval is the octave between note 54 and 66. The final tuning of note 54 was 179.92 Hz, which has a lower frequency relative to equal temperament, like note 66. It is possible that my dynamic tuning algorithm adjusted the notes to maintain the just octave interval, 2:1.

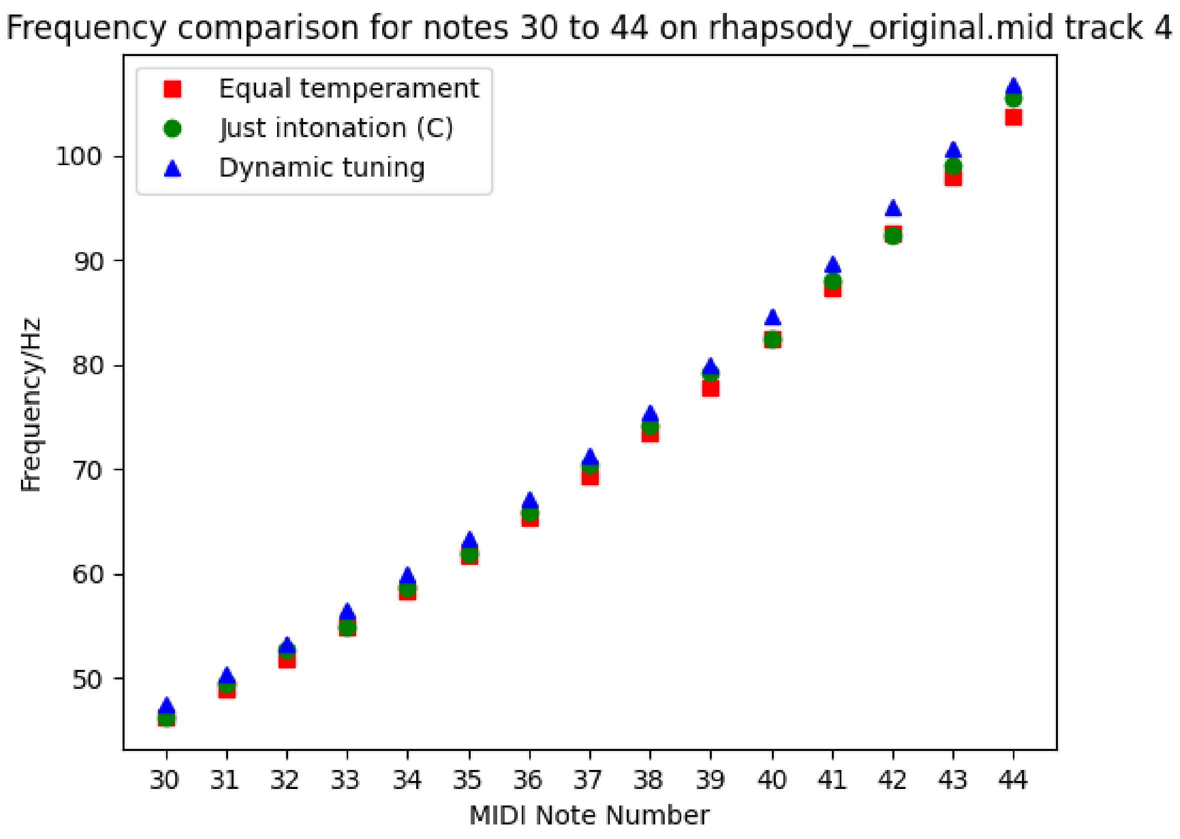


Figure 29 Frequency comparison in track 4 of Queen – Bohemian Rhapsody.

Table 19 Frequency comparison in track 4 of Queen – Bohemian Rhapsody.

Note	Frequency	Difference compared to baseline (12-TET)
30	47.55	1.30
31	50.38	1.38
32	53.37	1.46
33	56.55	1.55
34	59.91	1.64
35	63.47	1.74
36	67.25	1.84
37	71.24	1.95
38	75.47	2.05
39	79.97	2.18
40	84.72	2.32
41	89.76	2.45
42	95.10	2.60
43	100.75	2.76
44	106.75	2.92

The final examined frequency changes are in Figure 29 and Table 19. Here, the notes follow a trend of having higher frequencies, compared to 12-TET, and the magnitude of the difference scaling in proportion to the magnitude of the frequency. The examined notes are all played alone, without accompanying notes.

4.2.2 Discussion

In Figure 25 and Table 15, a mix of frequency increases and decreases are shown. Notes 55 and 56 have become close in frequency, however the other notes do not follow this trend. Notes 55 and 56 are never played simultaneously, therefore it could be a result of their frequencies becoming stuck in a local minimum of dissonance, when calculated with the other notes they are played with. These notes are often combined with notes with intervals between them, that are not represented on the dissonance plot, as seen in Figure 7, such as the 'minor seventh', which has an almost flat minimum between 5/2 and 2/1. This can be resolved by using more partials for every fundamental frequency, as more partials create more minima between two frequencies.

The tuning change of note 68 from Figure 28 and Table 18 is examined closer in Table 20. The note's frequency changes, and those of the simultaneous notes it was played with, are shown. The mean of note 68's frequency over time is 413.8 Hz; the standard deviation is 9.7 Hz; the median is 408.5 Hz; the mode is 407.4 Hz; and the range is [403.7 Hz, 427.6 Hz]. It is interesting that the mean is close to the baseline 12-TET frequency of 415.3 Hz. It would be possible to use the mean or the median frequency as the final tuning value for the note. This would avoid overfitting of the tuning to a particular combination of notes, which would also address the concerns about tuning notes adjacent in time, as changes are less drastic. In a scenario where the tuning is only set once this could sound better throughout the song. However, in a scenario where tuning is adjusted in real time, every time notes are played, taking an approach of being specific and retuning the notes as in Table 20, could still be a viable option.

The algorithm tends to increase frequencies rather than decrease them, with exceptions. This could be due to a ripple effect, resulting from one note increasing, and others following to maintain a previous interval. However, this effect was observed in notes that are frequently played alone, for example those in Figure 29 and Table 19. It is possible that there is a limitation in the system resulting in single notes drifting upwards in frequency, limited only by the frequency bound. This effect does not change the overall melody of the song, as the change is proportional throughout.

Table 20 Frequency changes of note 68, in track 3, of a MIDI adaptation of Queen - Bohemian Rhapsody.

Total delta time/Ticks	Notes Played	Frequency/Hz
6912	[44, 56, 60, 63, **68**]	[103.3, 206.7, 258.0, 309.7, **413.4**]
7200	[51, 60, **68**]	[159.9, 269.0, **404.0**]
20736	[41, 53, 65, **68**]	[85.3, 170.6, 341.1, **409.5**]
20928	[65, **68**, 72]	[339.5, **403.9**, 538.0]
21120	[39, 51, 63, **68**]	[80.0, 159.9, 319.7, **403.9**]
21312	[38, 50, 62, **68**, 72]	[75.5, 151.0, 301.8, **403.9**, 508.9]
24192	[**68**, 72]	[**406.7**, 508.9]
24768	[65, **68**]	[347.4, **417.2**]
34176	[41, 53, 65, **68**]	[87.3, 174.6, 349.0, **419.1**]
34368	[65, **68**, 72]	[353.1, **423.9**, 529.2]
34560	[39, 51, 63, **68**]	[80.0, 159.9, 319.6, **425.9**]
34752	[38, 50, 62, **68**, 72]	[74.4, 148.9, 285.4, **426.9**, 532.9]
37248	[41, 65, **68**]	[85.5, 355.5, **426.9**]
37632	[39, 63, **68**, 72]	[80.0, 320.4, **426.9**, 534.1]
[…]	[…]	[…]
73408	[**68**, 80]	[**403.7**, 807.8]
73600	[**68**, 80]	[**403.7**, 807.8]
74880	[44, 56, 63, **68**, 72, 75]	[103.3, 206.66, 310.1, **413.1**, 516.8, 619.3]
81984	[56, 60, 63, **68**]	[203.8, 254.8, 305.7, **407.4**]
84480	[44, 56, 59, 63, **68**]	[101.4, 202.9, 243.4, 304.3, **405.5**]
85248	[46, 60, 63, **68**]	[119.8, 254.8, 305.7, **407.4**]
85440	[63, **68**]	[305.7, **407.4**]
85632	[60, 63, **68**]	[254.8, 305.7, **407.4**]
86400	[56, 63, **68**, 72]	[202.4, 303.6, **404.6**, 538.6]
86592	[**68**, 72]	[**404.6**, 538.6]
88032	[**68**]	[**427.6**]
88128	[**68**]	[**427.6**]
88800	[**68**]	[**427.6**]
88896	[**68**]	[**427.6**]

4.3 Results & Discussion Summary

This chapter discussed the results of the experiments comparing my dynamic tuning approach with other tuning systems. The next chapter will wrap up the dissertation, concluding on the effectiveness of this algorithm and making suggestions for further work and improvements to be made.

5 Conclusion

5.1 Conclusions

The aim of the project was to improve the tuning of notes in a MIDI file. To find a solution to this problem, I had to explore many possible options. It was difficult to find a measurable way to quantify improvement. I explored options such as:

- Measuring improvements to the 'simplicity' of frequency ratios.
- Basing tuning on the calculated key, or base note, of a tune to apply unique just intervals.
- Swapping different variations of just intervals.

Ultimately, I decided to use dissonance between the timbre of notes played at a point in time.

My project manages to decrease the dissonance in 75% of the four MIDI songs it read. If it is possible to tune notes in real time, dissonance is decreased maximally throughout the song. Otherwise, it is possible to use the information of tuning throughout the track and take an average of changes to create a new temperament, like equal temperament but specific to the song. Dissonance is successfully decreased in most tested songs, with improvements up to 12%, and an average improvement of 5%. This average is skewed due to 3 tracks of one song, and it is improved to an 8% reduction if these outliers are returned to their original tuning.

During the project many problems were encountered. One such example was the gradient calculation being too large. This resulted in me trying to vary the step size to accommodate large changes in frequency. When the step size was decreased, I found that where the gradient was large, notes were adapting at an appropriate rate, but notes with smaller gradients would see no tuning changes. When the step size was increased, I found that the notes with large gradients would explode and exceed floating point number boundaries. My solutions to this problem were to tune the song by track/instrument; to limit the pitch change between each iteration; to remove irrelevant MIDI tracks e.g. drum/percussion instrument related tracks. Furthermore, the project at first lacked metrics such as those in the Implementation and Results & Discussion sections. The only metric was a comparison of frequencies, before and after the tuning. This made it difficult to evaluate if my project was improving over the baseline, as evaluating frequency change is more subjective than evaluating the changes in dissonance over different tracks. Once I discovered the dissonance solution my project could be quantifiably improved, resulting in a tangible improvement to sensory dissonance of frequency combinations.

5.2 Future Work

5.2.1 Explore Alternative Gradient Calculations

The gradient descent method used in the project is one of many optimization algorithms. There are others which may be more suited to the problem, such as *simultaneous perturbation stochastic approximation* (SPSA) as discussed in (Spall, 1997). These methods could be applied in the program and then compared to the original gradient descent method in terms of execution time, dissonance reduction, and subjectively when listening in a synthesizer.

5.2.2 Retune Unused MIDI Notes

The algorithm in its current state has the issue that the tuning format used only allows for a single retuning of a MIDI note per track. Although my algorithm has the capability of providing tuning for notes at every point of the song, it is limited by the free options available to apply it. This problem could be solved by using notes in the highest and lowest 2 octaves i.e. notes 0 to 24 and 103 to 127 or based on what notes are not being used in the track. These notes can be tuned to an alternative frequency for one of the used notes. The MIDI file would then be refactored by swapping the unused note numbers with its current notes. This could work well for examples such as that in Table 20, where a note has a large range of frequencies that it is retuned to. Frequencies could vary based on the context a note is played in.

5.2.3 Real-time MIDI Polyphonic Expression

In (MIDI Manufacturers Association, 2018), an addendum to the original MIDI specification, a new way of varying the pitch and timbre is discussed, MIDI Polyphonic Expression (MPE). Some synthesizers, such as Surge, and more in the 'Tuner plugins' section of the 'List of Microtonal Software Plugins' on the 'Xenharmonic Wiki' (Xenharmonic Wiki, 2021), have already adopted this addendum. Using MPE could be a good way of implementing real time tuning, inside a MIDI file. This would allow notes to be retuned dynamically at any point of the song, without third party standards such as scl/kbm being applied manually.

5.2.4 Horizontal Tuning and Comparison

A valuable next step for the project would be to examine the impact of tuning a note based on the notes that are adjacent in time. Currently, the algorithm only considers dissonance at a vertical level i.e. when notes are simultaneous. A solution to this problem could be to implement a moving window, which gives weight to notes around the group of current simultaneous notes, depending on how close they are in time. If a note is played within 100 milliseconds of the group, it could be looked at as an extra parameter to adjust the tuning to. This parameter could be weighted, to have less impact on the tuning depending on how close in time the note is played.

5.2.5 Comparison of Mean/Median Final Frequency

Dissonance comparisons could be made for when the mean of all pitches of a note, calculated throughout the song, is used, as discussed in section 4.2.2. This tuning of a note would be applied after the algorithm has completed, so that the frequency of the note is constant throughout the tune. It is possible that the dissonance would be higher in total, but as the notes aren't overfitted to each group of simultaneous notes, the tuning could be truer to the melody of the original tune.

6 BIBLIOGRAPHY

Bowling, D. L., & Purves, D. (2015). A biological rationale for musical consonance. *Proceedings of the National Academy of Sciences, 112*(36), 11155-11160. doi:10.1073/pnas.1505768112

Crocker, R. L. (1963). Pythagorean Mathematics and Music. *The Journal of Aesthetics and Art Criticism, 22*(2), 189-198.

Curry, H. B. (1944). The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 258-261.

de Coul, M. O. (2021, February 8). *Scala scale file format*. Retrieved from Scala: http://www.huygens-fokker.org/scala/scl_format.html

Horner, A., Beauchamp, J., & Haken, L. (1993). Methods for Multiple Wavetable Synthesis of Musical Instrument Tones. *Journal of the Audio Engineering Society, 41*(5), 336-356.

International Organization for Standardization. (1975). *ISO 16:1975 Acoustics – Standard tuning frequency (Standard musical pitch).* Retrieved from https://www.iso.org/standard/3601.html

Kleczkowski, P. (1989). Group Additive Synthesis. *Computer Music Journal, 13*(1), 12-20. doi:10.2307/3679851

Lahdelma, I., & Eerola, T. (2020, May 26). Cultural familiarity and musical expertise impact the pleasantness of consonance/dissonance but not its perceived tension. *Scientific Reports, 10*(1), 8693. doi:10.1038/s41598-020-65615-8

Lazzarini, V., & Timoney, J. (2010). Theory and Practice of Modified Frequency Modulation Synthesis. *Journal of the Audio Engineering Society, 58*(6), 459-471.

Lazzarini, V., Timoney, J., Pekonen, J., & Välimäki, V. (2009). Adaptive Phase Distortion Synthesis. *DAFx 09 proceedings of the 12th International Conference on Digital Audio Effects, Politecnico di Milano, Como Campus, Sept. 1 - 4, Como, Italy*, 1-8.

Lehrman, P. D. (2017). *What is MIDI?* Medford, MA: MMA.

Lindemann, E. (2007). Music Synthesis with Reconstructive Phrase Modeling. *IEEE Signal Processing Magazine, 24*(2), 80-91. doi:10.1109/MSP.2007.323267

MIDI Manufacturers Association. (1996). *The Complete MIDI 1.0 Detailed Specification.* Los Angeles, CA: MIDI Manufacturers Association.

MIDI Manufacturers Association. (1999). *MIDI Tuning Messages.* Los Angeles, CA: MIDI Manufacturers Association Incorporated.

MIDI Manufacturers Association. (2018, April 26). *MIDI Polyphonic Expression.* Retrieved from Official MIDI Specifications: https://www.midi.org/specifications/midi1-specifications/mpe-midi-polyphonic-expression

Moore, F. R. (1988). The Dysfunctions of MIDI. *Computer Music Journal, 12*(1), 19-28. doi:10.2307/3679834

Plomp, R., & Levelt, W. J. (1965). Tonal Consonance and Critical Bandwidth. *The Journal of the Acoustical Society of America, 38*(4), 548-560. doi:10.1121/1.1909741

Rabenstein, R., & Trautmann, L. (2001). Digital sound synthesis by physical modelling. *Symposium on Image and Signal Processing and Analysis (ISPA'01)* (pp. 12-23). Pula, Croatia: IEEE. doi:10.1109/ISPA.2001.938598

Roads, C. (1988). Introduction to Granular Synthesis. *Computer Music Journal, 12*(2), 11-13. doi:10.2307/3679937

Scala help: Mappings. (2021, April 1). Retrieved from Huygens Fokker: http://www.huygens-fokker.org/scala/help.htm#mappings

Schellenberg, E., & Trehub, S. (1994). Frequency ratios and the perception of tone patterns. *Psychonomic Bulletin & Review, 1*, 191-201. doi:10.3758/BF03200773

Sethares, A. W. (2002). Real-Time Adaptive Tunings Using Max. *Journal of New Music Research, 31*(4), 347-355. doi:10.1076/jnmr.31.4.347.14163

Sethares, W. (1993). Local consonance and the relationship between timbre and scale. *The Journal of the Acoustical Society of America, 94*, 1218-1228. doi:10.1121/1.408175

Sethares, W. (1994). Adaptive tunings for musical scales. *The Journal of the Acoustical Society of America, 96*, 10-18. doi:10.1121/1.410471

Sethares, W. A. (2005). *Tuning, Timbre, Spectrum, Scale.* London: Springer-Verlag.

Spall, J. C. (1997). A one-measurement form of simultaneous perturbation stochastic approximation. *Automatica, 33*(1), 109-112. doi:https://doi.org/10.1016/S0005-1098(96)00149-5

Terhardt, E. (1984). The Concept of Musical Consonance: A Link between Music and Psychoacoustics. *Music Perception, 1*(3), 276-295. doi:10.2307/40285261

Trulla, L. L., Di Stefano, N., & Giuliani, A. (2018). Computational Approach to Musical Consonance and Dissonance. *Frontiers in Psychology, 9*(381), 381-381. doi:10.3389/fpsyg.2018.00381

User:Angr. (2021, April 4). *Public domain.* Retrieved from Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Scientific_pitch_notation_octaves_of_C.png

Valimaki, V., & Huovilainen, A. (2007). Antialiasing Oscillators in Subtractive Synthesis. *IEEE Signal Processing Magazine, 24*(2), 116-125. doi:10.1109/MSP.2007.323276

van Steenhoven, K. (2010). *A Practical Guide to Intonation and Tuning.* Amsterdam: Karel van Steenhoven.

Wikipedia. (2021, February 8). *Equal temperament: Comparison with just intonation*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Equal_temperament#Comparison_with_just_intonation

Xenharmonic Wiki. (2021, April 1). *List of Microtonal Software Plugins.* Retrieved from Xenharmonic Wiki: https://en.xen.wiki/w/List_of_Microtonal_Software_Plugins

Zentner, M. R., & Kagan, J. (1998). Infants' perception of consonance and dissonance in music. *Infant Behavior and Development, 21*(3), 483-492. doi:10.1016/S0163-6383(98)90021-2