

July 11, 2021

Dr. Craig Scratchley  
School of Engineering Science  
Simon Fraser University  
Burnaby, British Columbia  
V5A 1S6

Re: ENSC 405W Design Specification for the **Sheet Music Transcriber** by **HappyJam**

Dear Dr. Scratchley,

As per the ENSC 405W Capstone A course Instructions, please find attached to this letter the design specifications for the SMT (Sheet Music Transcriber) by HappyJam. The SMT takes in an audio sample of music played on any instrument or a small group of instruments and converts the music to a score ready to be played or further edited by a musician.

The document below covers the outline and specifications for our proposed product. The specifications are divided into sections and releases for a clear description of different subsystems and for the timescale these systems will be developed across. These sections include the algorithmic design, user interface design, program data management design, and other design decisions.

HappyJam is a multinational, diverse, and multidisciplinary team of passionate senior engineering students: Computer Engineers Matthew Marinets, Polina Bychkova, Haoran Hu, and Avital Vetschazer; System Engineer Jaskirat Arora; and Electronics Engineer Akaash Parajulee.

Thank you very much for your time and consideration. We truly appreciate your concern and time investment. Please let us know if you have any questions or concerns. You could contact our Chief Communications Officer Polina Bychkova anytime at [pbychkov@sfu.ca](mailto:pbychkov@sfu.ca).

Regards,



Matthew Marinets  
Chief Executive Officer  
HappyJam



# Design Specification for The Sheet Music Transcriber

The Music Transcriber by HappyJam



## Company 1

### Authors:

Matthew Marinets (301311930)  
Akaash Parajulee (301308798)  
Polina Bychkova (301269789)  
Avital Vetschaizer (301301019)  
Jaskirat Arora (301319559)  
Haoran Hu (301350932)



## **Abstract**

This document outlines the design specifications of the Sheet Music Transcriber (SMT) as designed by HappyJam. The purpose of the product is to provide an analysis and visualization tool for music recordings to convert them into piano roll and sheet music formats. Design possibilities and decisions are outlined and justified for algorithms, software architecture, user interface, and possible artificial intelligence components. Supporting test plans are also included to ensure implementation satisfies design requirements.



## Change Log

*Table 1: Change Log*

Date	Version	Notes
2021-07-09	1	Completed first version



## Table of Contents

<b>Abstract</b>	<b>2</b>
<b>Change Log</b>	<b>3</b>
<b>Table of Contents</b>	<b>4</b>
<b>List of Tables</b>	<b>7</b>
<b>List of Figures</b>	<b>8</b>
<b>Glossary</b>	<b>10</b>
<b>1 Introduction</b>	<b>12</b>
1.1 Scope	13
1.2 Design Item Identification	13
<b>2 System Overview</b>	<b>14</b>
<b>3 Algorithm Design</b>	<b>16</b>
3.1 Spectrogram Generation	16
3.1.1 Window Size	16
3.1.2 Window Function	17
3.1.3 Spectrogram Sharpening	20
3.2 POI Identification	21
3.2.1 Power Thresholding	22
3.2.2 Frequency-Axis Peakfinding	23
3.2.3 Onset Detection with Time-Axis Peakfinding	24
3.2.4 Wide-Peak Detection with a Modified Hough Transform	26
3.2.5 Full Spectrum Identification of Closed Hi-Hat Notes	28
3.3 Pitch Mapping	29
3.3.1 Logarithmic Mapping of POIs to Pitch	30
<b>4 Software Design — Program State Management</b>	<b>31</b>
4.1 Implementation Frameworks	31
4.1.1 Qt Framework	31
4.1.2 Project Layout Structure	32
4.1.3 Qt Event framework	33
4.2 Data management	34
4.2.1 User Adjustable Parameter Management	34
4.2.2 User Input Raw Music Data Management	35
4.2.3 Algorithm Post-Processed Data Management	36
4.2.4 Saving and Loading	36



<b>5 Software Design — User Interface</b>	<b>38</b>
5.1 Use Cases	39
5.2 Theme	41
5.3 Structure	42
<b>6 Conclusion</b>	<b>45</b>
<b>7 References</b>	<b>46</b>
<b>Appendix A - Test Plan</b>	<b>48</b>
A.1 Algorithm Testing	48
A.2 Project Manager Testing	50
<b>Appendix B - Supporting Design Options</b>	<b>52</b>
B.1 Algorithm Design Options	52
B.1.1 Window Function Options	52
B.1.2 POI Identification Alternatives	52
B.1.3 Onset Detection Alternatives	54
B.1.4 Note Mapping Enhancements	54
B.2 Technical Stack Selection	57
B.2.1 Implicit Requirements	57
B.2.1.1 Performance	57
B.2.1.2 Syntax complexity	57
B.2.1.3 Third-party and community support	57
B.2.1.4 Integrated development environment support and dev tooling	58
B.2.1.5 Cross Platform Compatibility	58
B.2.1.6 Software Distribution and Packaging	58
B.2.2 UI frameworks overview	58
B.2.3 Stack Comparison Matrix	60
<b>Appendix C - User Interface and Appearance</b>	<b>63</b>
C.1 Introduction	63
C.1.1 Purpose	64
C.1.2 Scope	64
C.2 Audience and User Analysis	64
C.3 User Interface Analysis	64
C.3.1 Visibility	65
C.3.2 Feedback	65
C.3.3 Constraints	65
C.3.4 Mapping	66
C.3.5 Consistency	66



C.3.6 Affordance	66
C.4 Engineering Standards	67
C.5 Audience Research	68
C.5.1 User Profile 1	68
C.5.2 User Profile 2	69
C.5.3 User Profile 3	70
C.6 Testing	72
C.7 Graphical Representation	73
C.8 Conclusion	75
C.9 References	76



## List of Tables

<b>Table 1: Change Log</b>	<b>3</b>
Table 3.1.1 — Design Items Governing the Spectrogram Window Size	17
Table 3.1.2 — Design Items Governing the Spectrogram Window Function	20
Table 3.1.3 — Design Items Governing Spectrogram Sharpening	21
Table 3.2.1 — Design Items Governing the Power Thresholding Algorithm	23
Table 3.2.2 — Design Items Governing Frequency-Axis Peakfinding	24
Table 3.2.3 — Design Items Governing Onset Detection Hints	26
Table 3.2.4 — Design Items Governing Identification of Drum Hits	28
Table 3.2.5 — Design Items Governing Identification of Hi-Hat Hits	29
Table 3.3.1 — Design Items Governing Mapping POIs to Pitch	30
Table 4.1.1 — Design Items Governing the UI Framework	31
Table 4.1.2 — Design Items Governing project layout structure	33
Table 4.2.1 — Design Items Governing User Adjustable Parameters	35
Table 4.2.2 — Design Items Governing User Input Music Data	35
Table 4.2.3 — Design Items Governing Post-Processed Data	36
Table 4.2.4 — Design Items Governing Project Saving and Loading	36
Table 5.1 — Design Items Governing UI Use Cases	39
Table A.1.1 — General SMT Algorithmic Functionality Testing Plan	48
Table A.1.2 — Single Instrument Testing Plan	48
Table A.1.3 — Dual Instrument Testing Plan	49
Table A.1.4 — Multi-Instrument Testing Plan	50
Table A.2.1 — Project Manager Testing Plan	50
Table B.2.2.1 — Base Design Specification Table for the UI Framework	59
Table B.2.2.2 — Summary of Design Specification of the above frameworks	60
Table B.2.3.1 — Design Requirements that are fulfilled with the stack choice	61
Table B.2.3.2 — Stack Design Options	62
Table B.2.3.3 — Comparison Summary	63
Table C.1 — The Music Transcriber UI Goals	64
Table C.4.1 — Engineering Standards	68
Table C.5.1 — Michael's Q&A	69





Table C.5.2 — Jane’s Q&A	71
Table C.5.3 — Alex’s Q&A	72
Table C.6.1 — UI Testing Plan	72



## List of Figures

Figure 1.1 — An information flow diagram of what the SMT should accomplish	13
Figure 2.1 — A hierarchical model of the software, showing subsystem interactions	16
Figure 3.1 — Spectrogram of a recording playing middle C on a piano.	17
Figure 3.1.2.1 - The Rectangular window and its Fourier Transform, the sinc function	19
Figure 3.1.2.2 — A frequency spectrum plot of a piece of music created with a rectangular window function.	19
Figure 3.1.2.3 — A frequency spectrum of the same audio sample created with a Hann window function.	20
Figure 3.1.2.4 — The Hann window and its Fourier Transform	20
Figure 3.1.3.1 — A prominence kernel.	22
Figure 3.2.1.1 — A close up of a spectrogram generated from Top Gear Track 1, with pixels above a threshold highlighted in purple.	23
Figure 3.2.2.1 — A close up of a spectrogram generated from Top Gear Track 1, with frequency-axis peaks highlighted in green.	24
Figure 3.2.3.1 — A close-up of the spectrogram of the Skill Tree Song from the Rogue Legacy OST with frequency-axis peaks highlighted in green. Note the chord at frequency bucket 42 has no break when it is played again at time-sample 145.	26
Figure 3.2.3.2 — Close-up of the same spectrogram as Figure 3.2.3.2, this time with points where the second derivative of power with respect to time is less than -2 highlighted in cyan.	27
Figure 3.2.4.1 Frequency responses of a drum, cymbal, and closed hi-hat	28
Figure 4.1.2: Layout Structure Example	33
Figure 4.1.3 Comparison Summary	35
Figure 5.1 User Interface Framework	39
Figure 5.2.1. Color Palette for UI theme	43
Figure 5.3.1 A scaled graph of the structure of user interface, which can be use as a reference to Figure 5.1	44
Figure 5.3.2 scheme for menu bar	44
Figure 5.3.3 main windows including sidebar and Central Window, which displays sheet music, audio waveform, audio spectrum and piano roll	45
Figure B.1.2.1 — An unwanted POI ridge that is part of the same connected region of interest as the main ridge. Using a per-region maximum detection algorithm would keep this unwanted ridge from being highlighted as a POI.	54
Figure B.1.3.1 — A thresholded spectrogram with peaks along the time-axis highlighted in cyan.	55



Figure B.1.4.1 — A spectrogram of an acoustic bass playing a note around 73.4 Hz, showing off the time-smearing with a normal spectrogram. Taken from A Unified Theory of Time Frequency Reassignment [2] 56

Figure B.1.4.2 — Reassigned spectrogram of an acoustic bass playing a note around 73.4 Hz. Taken from A Unified Theory of Time Frequency Reassignment [2] 57



## Glossary

### 12TET

Twelve-tone equal temperament. The standard tuning system in modern Western music, organizing pitch into twelve equally-spaced tones in every octave. An octave has a frequency difference of a factor of 2, and hence every tone in a 12TET system will be  $\sqrt[12]{2}$  times higher than the next-lowest tone.

### Chord

A group of notes played at the same time.

### FFT

Fast Fourier Transform — a family of algorithms that quickly calculate a Discrete Fourier Transform by taking advantage of properties of the Fourier Transform and reusing intermediate values.

### Harmonic

A component of a periodic signal or waveform at a particular integer multiple of the fundamental frequency. The sinusoidal component with the same frequency as the overall signal is the first harmonic; the component at twice the frequency of the signal is the second harmonic; and so on.

### Note

The smallest unit of musical analysis. A note is a pulse of sound with a few properties:

- Timing: when the note is played
- Duration: How long the note lasts
- Pitch: the fundamental frequency of the sound
- Volume: the amplitude of the sound
- Timbre: waveform of the sound; timbre varies with instrument

### Pitch

Pitch is the name musicians give to the fundamental frequency of a note. Pitches have names that loop through [A, A#, B, C, C#, D, D#, E, F, F#, G, G#], sometimes notated with a number following the pitch symbol to indicate which octave it falls in. Pitch names are spaced logarithmically, so A4 (440 Hz) is twice as high as A3 an octave below (220 Hz).

### POI

Point of Interest. Refers to locations in the spectrogram that may represent part of a note or a harmonic of a note.



### Score

A score is a document written in sheet music that describes how to play a song.

### Sheet Music

Sheet music is the primary notation used in Western music to describe how to play a particular part or song.

### Spectrogram

A two-dimensional image acquired from an audio sample representing the relative power carried by across time and frequency axes.

### Transcription

The process of writing down music in sheet music notation.

### Timbre

Timbre is the quality of a note that differs between instruments, manifesting physically as a different waveform. As the wave is periodic, timbre may be described as the relative power of a note's harmonics — that is, the relative power present in frequencies around integer multiples of the fundamental frequency.

### Tempo

The speed by which a section of the music is played, expressed in beats per minute (bpm).



## 1 Introduction

Musicians are trained to create music with their instruments, and often sheet music as a written notation to record and communicate musical information. Performing a piece of music from its score is a generally straightforward task and can be done in real-time by a skilled performer. Converting information the other way, transcribing music into a score, is much more difficult. This process requires repeated listening, trial and error, and specialized training.

The Sheet Music Transcriber (SMT) is a program that eases the transcription process by automating the transcription process, generating scores from audio recordings and providing visualizations for the different stages of analysis.

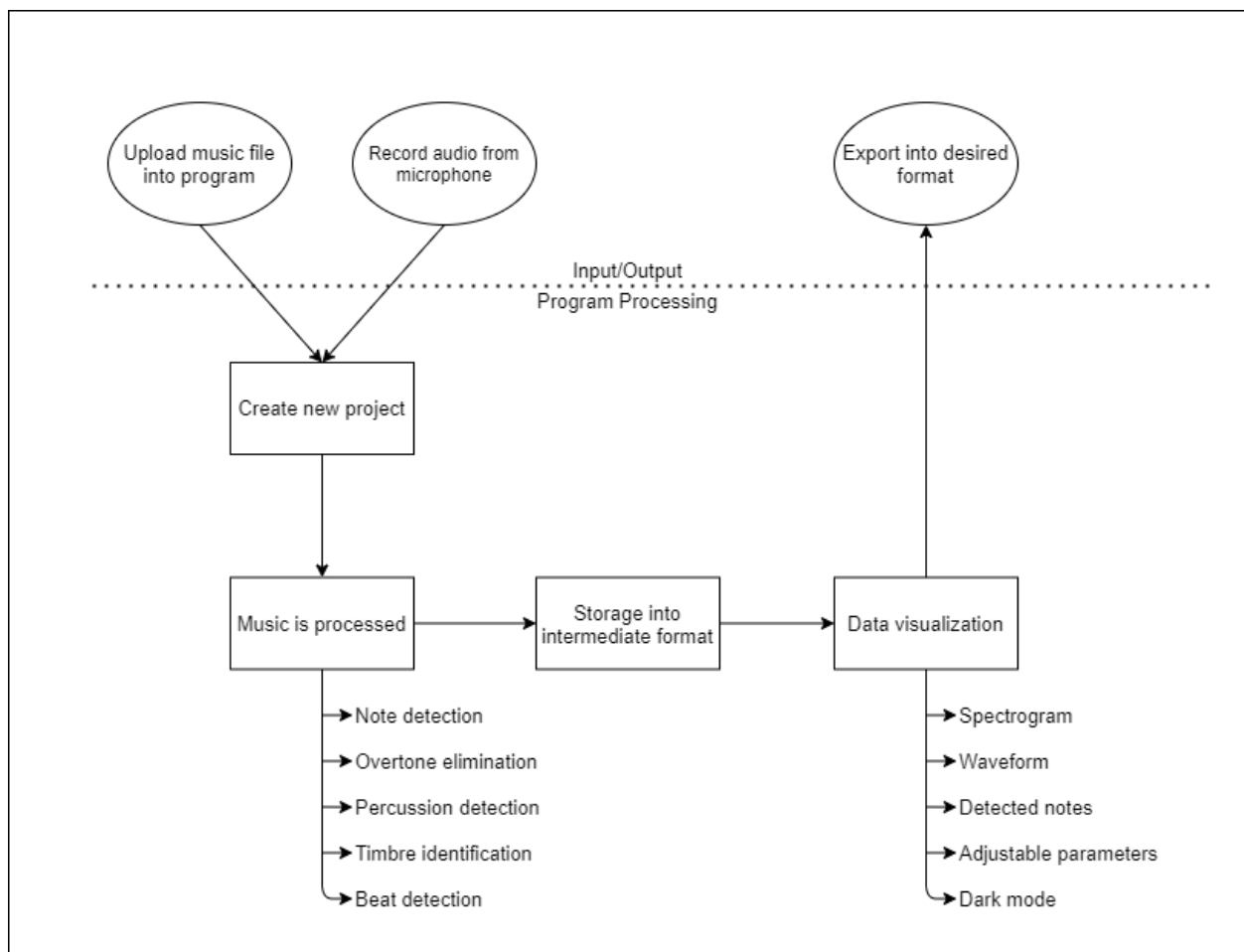


Figure 1.1 — An information flow diagram of what the SMT should accomplish



### 1.1 Scope

This document specifies design specification items for each subsystem of the Sheet Music Transcriber, with a preamble including justification and relevant background information. Design items are stated for the prototype version of SMT currently under development.

Appendix A additionally includes a testing plan for all design items. Appendix B includes information on additional design options, including paths not followed and justification for their exclusion. Appendix C includes design further decisions and justifications describing the UI.

### 1.2 Design Item Identification

Each design item is given an identification code in the following format:

D-[Section].[Design item – Elaboration ID]-[Phase]

- [Section] is the section of this document the the design item falls under
- [Design item – Elaboration ID] is a dot-separated list of numbers providing a unique identification for each design item. Following design items will increment the first number, with secondary numbers used to elaborate on parent design items
- [Phase] refers to which release phase the design item is a part of
  - A means the feature is required for alpha
  - B means the feature is required for beta
  - V1 means the feature is required for the first full release



## **2 System Overview**

The SMT has three main subsystems to be designed:

- The algorithmic system
- The state manager
- The user interface

The algorithmic system identifies notes and musical properties such as tempo and time signature. It includes several input parameters which the user may adjust to fine-tune note detection and identification.

The state manager governs how data is organized and passed between the algorithmic system and the user interface, maintains state, and saves project setting and progress to disk. To allow for easier algorithm development and user modification, the system architecture must be able to dynamically load different algorithms and input parameter sets, and update the UI and intermediate data in storage to match.

The user interface includes all data input and visualization methods in the SMT, including audio spectrogram display, data display, and algorithm parameter input methods.



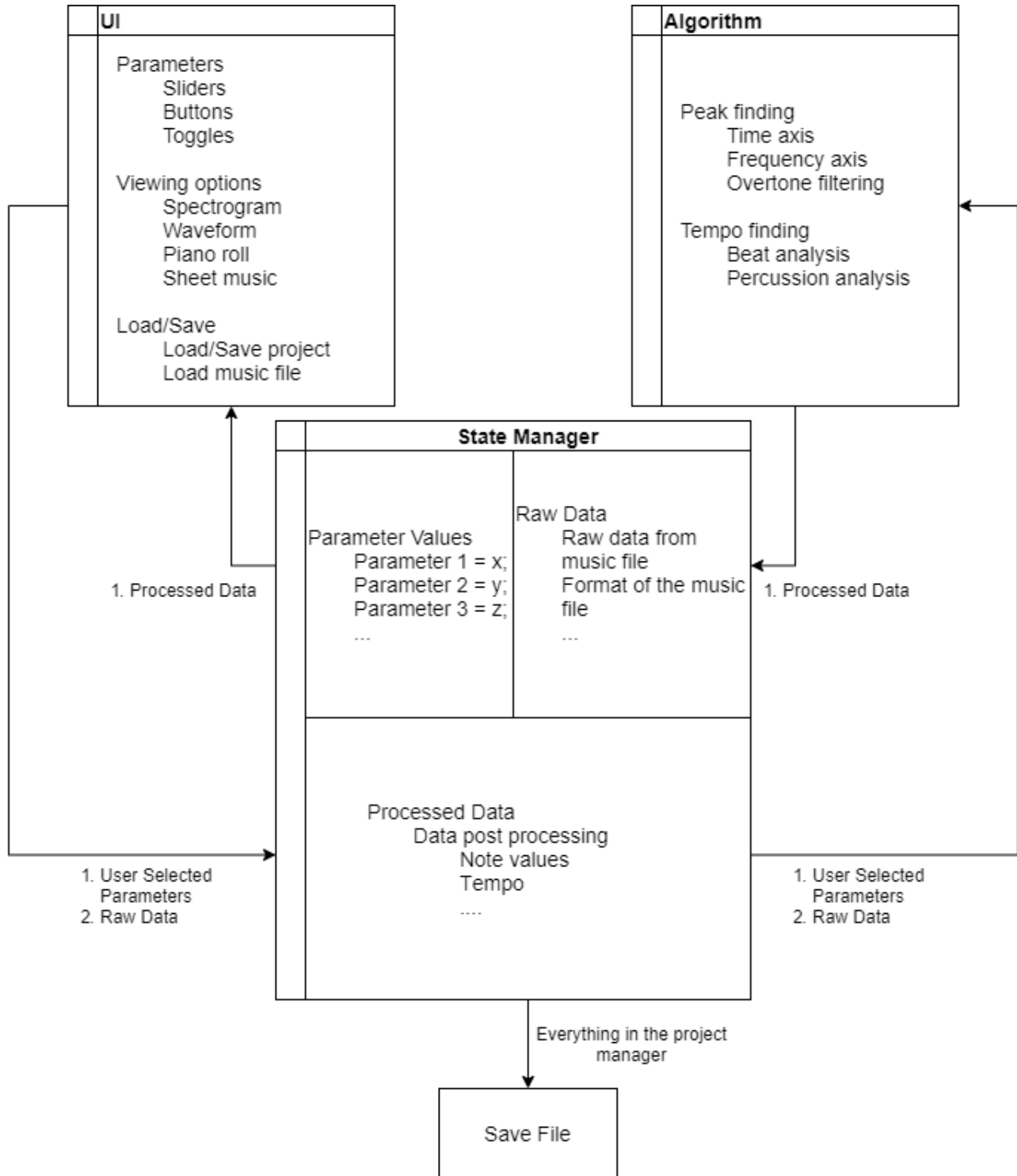


Figure 2.1 — A hierarchical model of the software, showing subsystem interactions



### 3 Algorithm Design

Automated assistance in identifying notes is one of the key components of the SMT. The general workflow of the assistance algorithms is to generate a spectrogram from the audio sample, identify patterns and points of interest within the spectrogram, and join or divide these points of interest into notes with properties pulled from the spectrogram.

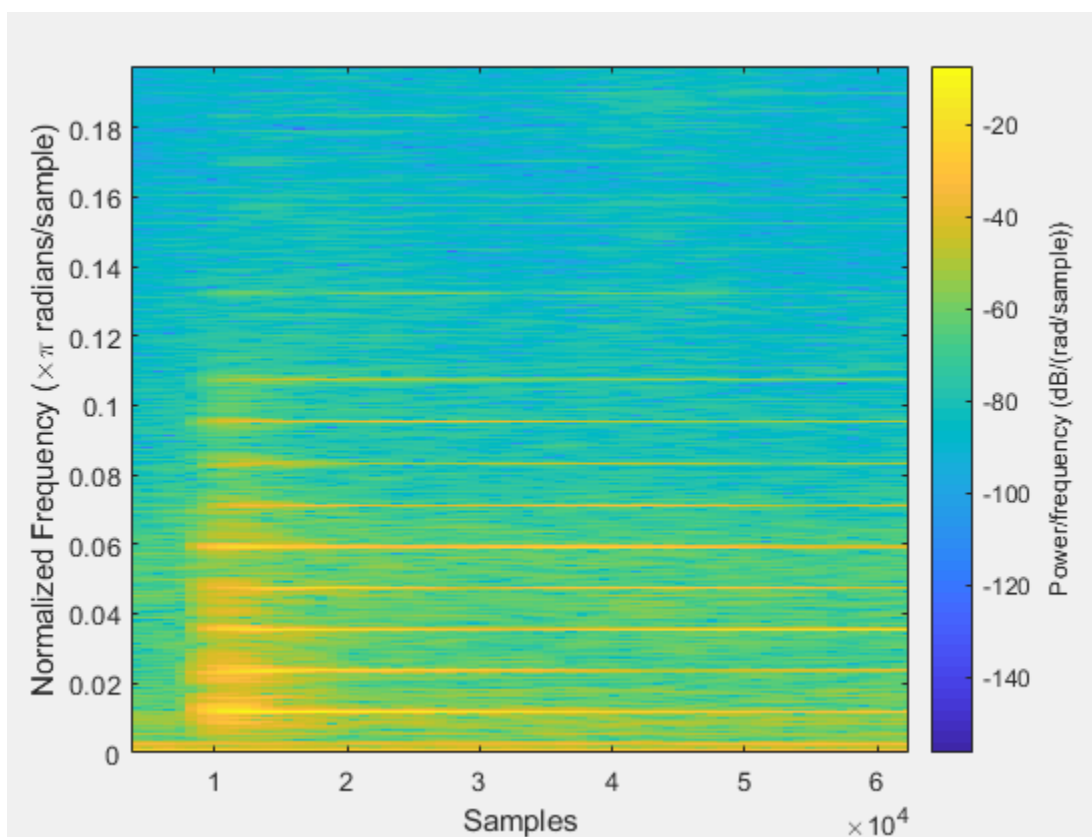


Figure 3.1 — Spectrogram of a recording playing middle C on a piano.

#### 3.1 Spectrogram Generation

A spectrogram is generated by breaking an audio sample into smaller subsections and taking the Fourier Transform of those sections with the Fast Fourier Transform. A two-dimensional image is thus generated, with one axis representing the time at the center of a window, and the other axis representing the frequency component of that window. The magnitude at each pixel thus represents the power density at the pixel's time and frequency.

##### 3.1.1 Window Size

Window size refers to the number of points used for each subsection's Fourier Transform. This generally presents a tradeoff — more points gives better frequency



resolution, but consumes more samples and hence gives poorer time resolution. While there are ways to work around this tradeoff [1] [2], many of our other algorithms run before these later cleanups.

A core problem in this tradeoff is that our frequency resolution requirements are not linear — we need better resolution to differentiate bass note pitches compared with treble note pitches. Note how in figure 3.1, the fundamental representing middle C — the 40th note on the piano, is only about 50 pixels from the bottom edge of the spectrogram. This comes about because musical pitches are logarithmically spaced; lower pitches have much smaller differences in frequency compared with higher pitches.

The two lowest notes on a piano are A0 (27.5 Hz) and A#0 (29.135), with a frequency difference a little over 1.5 Hz. With a standard sampling rate of 44.1 kHz, it would take

$$\frac{2(44100)}{(1.5)} = 58800 \text{ samples}$$

or about 1.3 seconds' worth of recording to detect that difference. That would completely eliminate any rhythmic information faster than that at all frequency levels, which would render most note duration detection impossible.

With some experimentation, we've found that a window size of  $2^{13} = 8192$  provides reasonable frequency resolution for most bass notes. To stand with the secondary purpose of the SMT — giving users visibility into the music and what the algorithms are doing — we have also decided to give the user the option to adjust this window size for the algorithm and for the spectrogram display.

Table 3.1.1 — Design Items Governing the Spectrogram Window Size

Design Item ID	Description	Requirement ID
D-3.1.1.1-A	The SMT algorithm will generate a spectrogram from the input audio sample	R4.5.4-B
D-3.1.1.2-A	The spectrogram generator will take input from the user governing the window size	R4.6-B
D-3.1.1.3-A	The spectrogram window size parameter will default to 8192 samples	-



### 3.1.2 Window Function

A spectrogram is generated by taking the Fourier Transform of only small sections of an audio sample, but this has a major visibility drawback. Namely, taking these samples unscaled and ignoring the others is the same as multiplying by a rectangular pulse in the time domain. By the multiplication – convolution duality property of the Fourier Transform, this is the same as convolving with  $\text{sinc}(f) = \frac{\sin(\pi f)}{\pi f}$  in the frequency domain. These functions are shown in Figure 3.1.2.1 below.

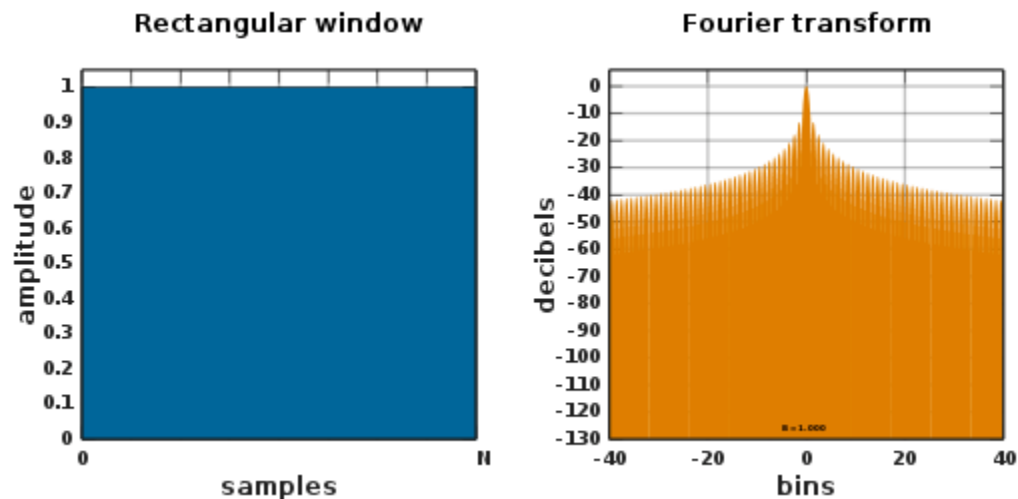


Figure 3.1.2.1 — The Rectangular window and its Fourier Transform, the sinc function

Convolution with such a spread-out function serves to spread all peaks in the spectrogram along the frequency axis, and creates a high effective noise baseline. Notice how in Figure 3.1.2.2, there is a linear fall-off past about 1500 Hz that obscures many harmonics in that frequency region. Compare that with Figure 3.1.2.3, which shows the Fourier Transform of the same audio sample but using a window function to block off that effect.

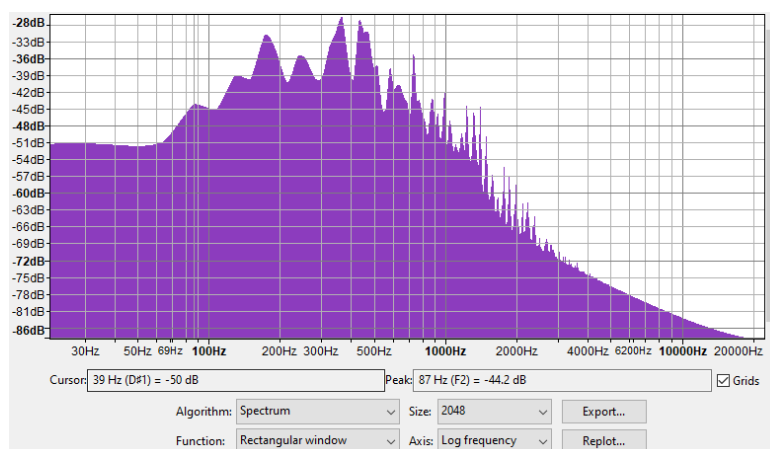




Figure 3.1.2.2 — A frequency spectrum plot of a piece of music created with a rectangular window function.

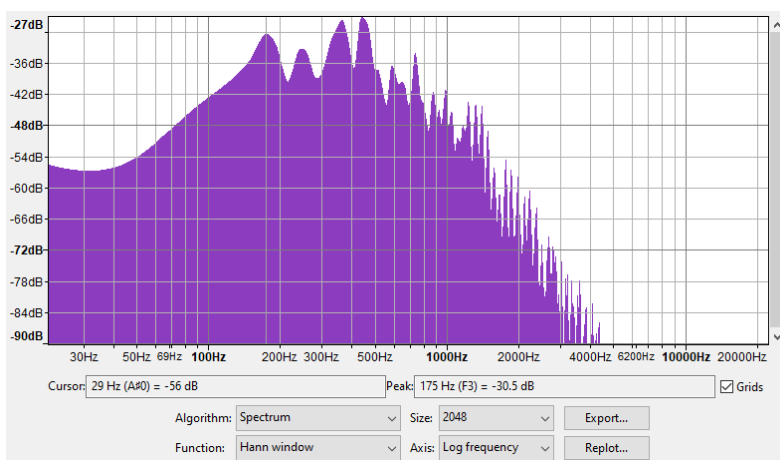


Figure 3.1.2.3 — A frequency spectrum of the same audio sample created with a Hann window function.

A window function is simply a vector of values to multiply the input audio sample values by before calculating the Fourier Transform. The window function is chosen to have a Fourier Transform that concentrates power near  $f = 0$  to prevent smearing peaks along the frequency axis in spectrograms. For example, the Hann window which was used to generate the spectrum in Figure 3.1.2.3 and Fourier Transform plot in Figure 3.1.2.4 below. It has a bell-curve-like shape, and its Fourier Transform falls off away from  $f = 0$  far more rapidly than the sinc function.

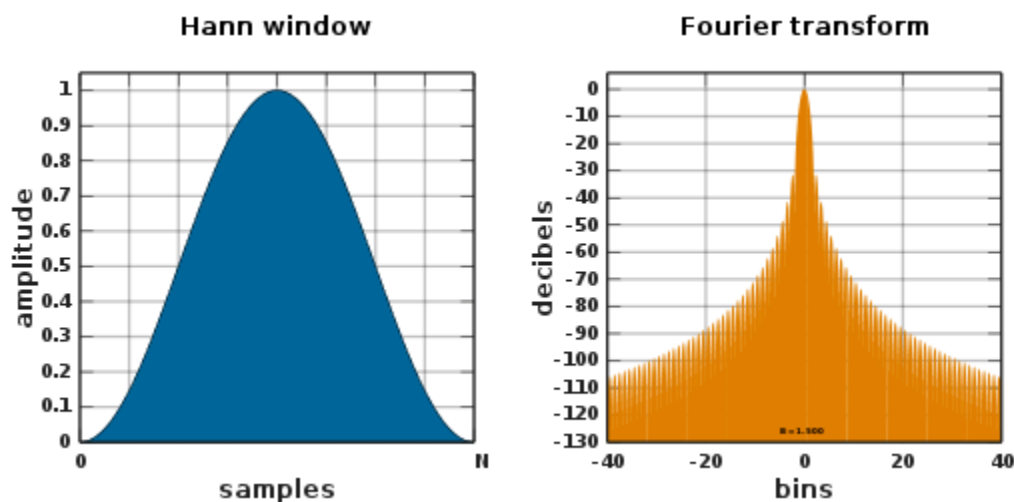


Figure 3.1.2.4 — The Hann window and its Fourier Transform



There are many other possible window functions to use, but the Hann window in particular is used in [2] for an algorithm to more precisely place spectrogram pixel coordinates. As such, we have chosen to use the Hann window when generating the spectrogram

Table 3.1.2 — Design Items Governing the Spectrogram Window Function

Design Item ID	Description	Requirement ID
D-3.1.2.1-A	The spectrogram generator will use a Hann window when sampling points	-

### 3.1.3 Spectrogram Sharpening

Music recordings are generally composed of four kinds of sound:

- Harmonic notes
- Percussive notes
- Distortion
- Noise

Our primary goal in designing the SMT is to identify harmonic notes. Hence, we wish to have a simple way to filter out the other kinds of noise while leaving harmonic notes easy to identify.

As previously seen in Figures 3.1, 3.1.2.2, and 3.1.2.3, harmonics tend to show up as particularly skinny peaks, in contrast with the other kinds of sound which tend to be wider peaks or even flat baseline noise levels. Hence, we may try to dampen the non-harmonic sound with a prominence filter.

A prominence filter simply subtracts the weighted average of the power in the region around a pixel from that pixel's value. It may be implemented as a convolution with a kernel that is the sum of a negative gaussian and a dirac-delta. An example prominence kernel is shown in Figure 3.1.3.1, where the gaussian has a standard deviation ( $\sigma$ ) of 21 and the kernel overall has a width of 63. Applying this filter along any axis has a sharpening effect, attenuating pixels close to peaks while leaving peaks themselves largely intact.

This filter is simple, but has two parameters that a user may want to control — the width of the Gaussian  $\sigma$  and the ratio of the power of the Gaussian compared with the power of the delta  $R$ . While we've had success experimenting with  $\sigma = 101$ ,  $R = 1$ , this may be a parameter users wish to change. It stands in line with our goal of providing visibility, and as this is a simple convolution, calculating and displaying the resulting sharpened spectrogram should be straightforward and fast enough to provide rapid



feedback to the user. Hence, we will provide the user with sliders to adjust these parameters.

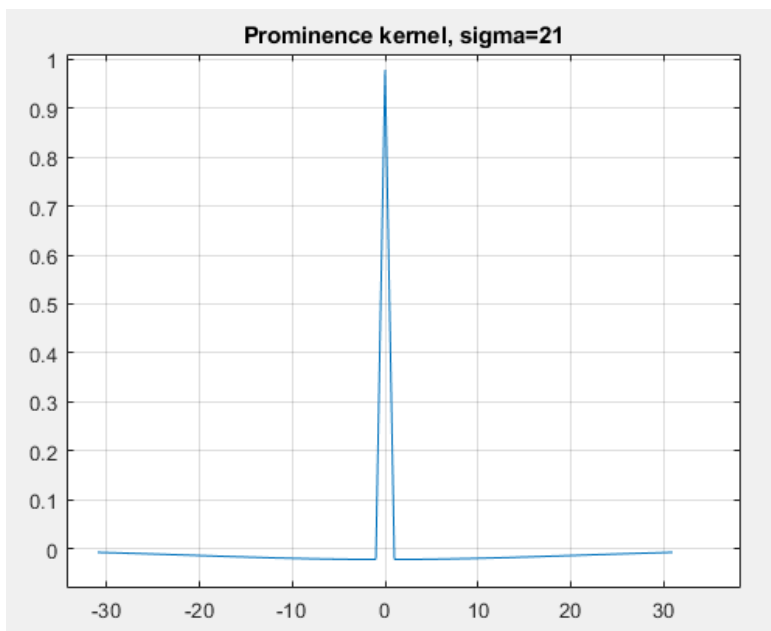


Figure 3.1.3.1 — A prominence kernel.

Table 3.1.3 — Design Items Governing Spectrogram Sharpening

Design Item ID	Description	Requirement ID
D-3.1.3.1-A	The spectrogram will be sharpened with a sharpening kernel defined as the sum of a negative gaussian and a dirac-delta function, applied along the frequency axis	-
D-3.1.3.2-A	The spectrogram sharpening algorithm will take input from the user specifying the standard deviation of the filter's gaussian component	R4.6-B
D-3.1.3.3-A	The spectrogram sharpening algorithm will take input from the user specifying the power ratio between the dirac-delta component and the gaussian component	R4.6-B

### 3.2 POI Identification

After applying general processing to all pixels in the spectrogram, it is important to identify POIs (Points of Interest). These may be possible notes or their harmonics. While harmonics should not appear in the final score, it is still important to identify them



for timbral analysis; that is, to identify what instrument played a note. While most algorithms outlined in this section deal with detecting harmonic notes, which tend to be highly concentrated peaks in the frequency domain, we are also interested in detecting notes played by percussion instruments. As such, this section includes discussion of a modified Hough Transform algorithm that may detect a different class of POIs representing percussive notes.

### 3.2.1 Power Thresholding

The simplest POI identification strategy is to simply filter and compare power at a spectrogram pixel against a threshold value. When the threshold suits the recording, as in Figure 3.2.1.1 below, note heads are highlighted with good accuracy and noise is fairly well-filtered.

However, different recordings will have different background noise levels and different levels of average power. Hence, there must be some method to adjust this threshold. One possible method of deciding on the threshold is by placing it at a known percentage of the spectrogram maximum pixel value. Unfortunately, as we have not tested this process with a sufficiently large sampling of recordings, we are not entirely confident in where this threshold may lie. In the name of providing the user visibility and control over the note detection process, it is also necessary to give the user control over where this threshold is. As such, the SMT will have a user-controlled input to decide where the POI power threshold lies.

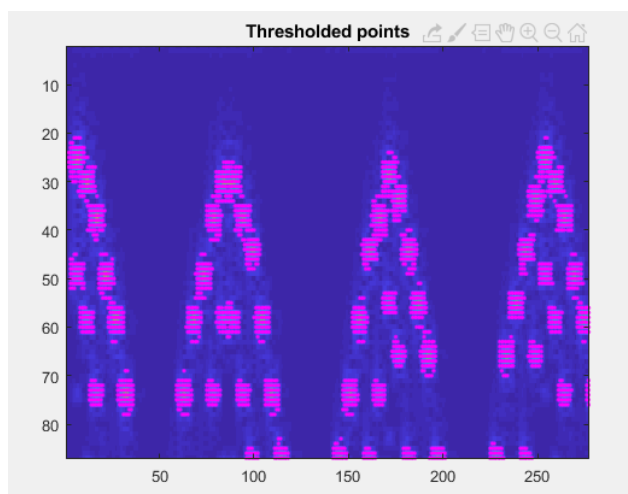


Figure 3.2.1.1 — A close up of a spectrogram generated from Top Gear Track 1, with pixels above a threshold highlighted in purple.





Table 3.2.1 — Design Items Governing the Power Thresholding Algorithm

Design Item ID	Description	Requirement ID
D-3.2.1.1-A	The POI identification algorithm will compare spectrogram pixel values to a threshold to obtain a more constrained list of POI candidates.	R3.1-A
D-3.2.1.2-A	The POI threshold will take input from the user controlling what exactly that threshold is.	R4.6-B

### 3.2.2 Frequency-Axis Peakfinding

Once POI candidates are determined through thresholding, we must further filter the points to obtain a list representative of note pitch information. As pitch is mapped from frequency, this is the same as finding the most representative frequency coordinate within the spectrogram of a thresholded region. However, as seen in Figure 3.2.1.1, thresholded regions tend to take up a wide range in the frequency axis. For bass notes, this wide range can bleed over note boundaries, resulting in POI candidates that can map to many adjacent pitch identifications.

To get around this, the SMT looks for the most representative point along the frequency axis within a thresholded region. This is most easily accomplished with peak-finding — simply tagging points that have a greater magnitude than their adjacents. When highlighted, this has the effect of creating horizontal lines in the spectrogram representing power ridges along the frequency axis, as seen in Figure 3.2.2.1 below.

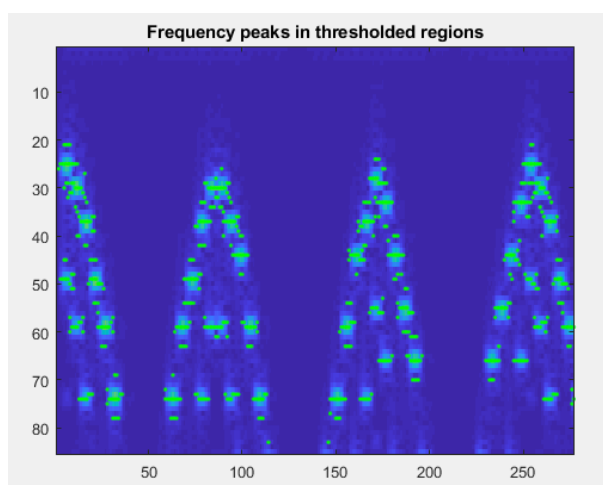


Figure 3.2.2.1 — A close up of a spectrogram generated from Top Gear Track 1, with frequency-axis peaks highlighted in green.



This already generates a somewhat suitable list of POIs, though there are some problems left to be resolved. First, and visible in Figure 3.2.2.1, are interference ridges; ridges that are close to each other diagonally tend to curve toward each other in diagonal patterns. Second, secondary ridges are detected running parallel to primary ridges; these come about due to the window function's Fourier Transform generally having a secondary peak on either side of the primary peak, which is then copied into the spectrogram due to convolution–multiplication duality. Third, some singular noise points are propagated from the thresholding stage into single-point ridges in the peakfinding stage. Additional filtering must be done to counter these effects

However, taking these points as our POI candidates already provides a solid baseline for note detection. Noise and secondary ridges will often map to the same pitch as a parent ridge, or will create a POI in the pitch domain that only lasts a single unit of time and is thus easily filtered. Additionally, the detection of diagonal ridges in addition to horizontal ridges leaves the future possibility of detecting and representing notes that change in pitch over time. This may occur with certain string instruments, such as guitar or violin, which can bend a string or slide along a string while playing on it to smoothly vary pitch. It also happens in electronic music, which may use FM modulation to create a unique sound.

*Table 3.2.2 — Design Items Governing Frequency-Axis Peakfinding*

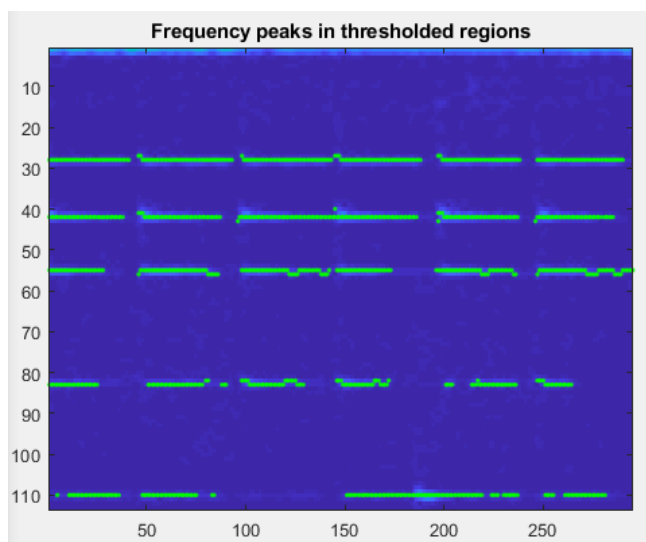
Design Item ID	Description	Requirement ID
D-3.2.2.1-A	The POI identification algorithm will detect peaks along the frequency axis within the thresholded spectrogram to be included as POIs	R3.3-A
D-3.2.2.2-B	The POI identification algorithm will filter ridges that are too short	R3.1-A
D-3.2.2.3-B	The POI identification algorithm will discount ridges that are in the expected region of secondary ridges to ridges with higher power	R3.1-A
D-3.2.2.4-V1	The POI identification algorithm may recognize diagonal ridges as part of FM-modulated notes	R3.1-A

### 3.2.3 Onset Detection with Time-Axis Peakfinding

POIs as identified with frequency-axis peakfinding cannot be naively mapped straight to notes while still detecting repeated notes at the same frequency. Observe, for example,



Figure 3.2.3.1, where a chord is played repeatedly and allowed to ring out long enough to bleed into the next time the chord is played. With a naive frequency-ridge-to-note mapping, this would be identified as a single chord held for the combined duration of both real chords in the audio sample.



*Figure 3.2.3.1 — A close-up of the spectrogram of the Skill Tree Song from the Rogue Legacy OST with frequency-axis peaks highlighted in green. Note the chord at frequency bucket 42 has no break when it is played again at time-sample 145.*

To give downstream algorithms a hint that the single frequency ridge is composed of multiple notes, the SMT algorithm must have some kind of onset detection. A simple solution to detect likely onset points is to check the downward curvature of the power when moving along the time-axis. A sharp downward curve in power over time indicates that a POI is rapidly beginning to decay, which is most common at the beginning of a plucked or percussive note. Figure 3.2.3.2 shows points in the same spectrogram as Figure 3.2.3.1 where this downward curvature is below -2. Observe that these points are most common around the beginning of notes.

This onset detection method requires another threshold parameter, which the user may want to adjust. The threshold of -2 used in figure 3.2.3.2 below works fairly well, and so we will have that as a default. This parameter may not be necessary for alternative onset-detection algorithms which are highlighted in Appendix B, section B.1.3.

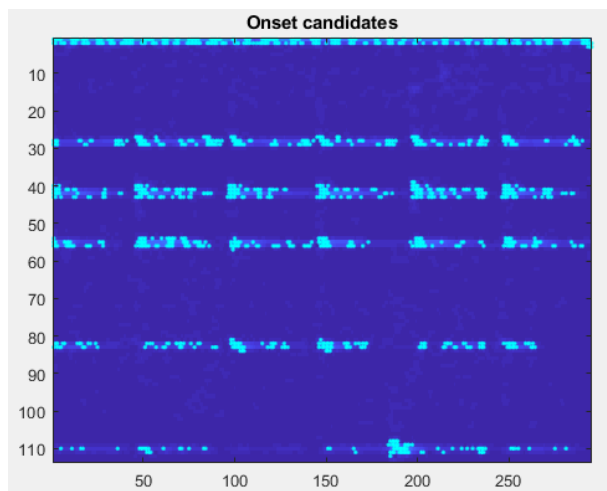


Figure 3.2.3.2 — Close-up of the same spectrogram as Figure 3.2.3.2, this time with points where the second derivative of power with respect to time is less than  $-2$  highlighted in cyan.

These onset points may be used as a hint in downstream algorithms for dividing frequency ridges into multiple notes.

Table 3.2.3 — Design Items Governing Onset Detection Hints

Design Item ID	Description	Requirement ID
D-3.2.3.1-A	The onset detection algorithm will take the second derivative of the power spectrogram along the time axis and tag all those pixels with a resulting magnitude less than a threshold value as onset candidates	R3.1.1-A, R3.1.3-B
D-3.2.3.2-A	The onset detection algorithm will take an input from the user governing what curvature threshold to use for onset detection	R4.6-B
D-3.2.3.3-A	The onset detection algorithm will aggregate onset candidate pixels into a most likely onset instant	-

### 3.2.4 Wide-Peak Detection with a Modified Hough Transform

Detection of percussive notes in a portion of music is of particular interest because the timbre of these instruments can be quite unique. In a standard drum kit, there are typically 5 drums and 3 cymbals. The drums exhibit a different frequency response from other instruments, with the logarithmic response on a graph resembling a normal



distribution. Cymbals (with the exception of a closed hi-hat) have a timbral profile that is typical to other instruments, with a fundamental frequency and certain overtones that can be identified.

The major exception is the hi-hat. A closed hi-hat has a frequency response similar to that of white noise, producing frequency components across it's spectrum. Below are some examples of a typical frequency response from a drum, cymbal, and hi-hat.

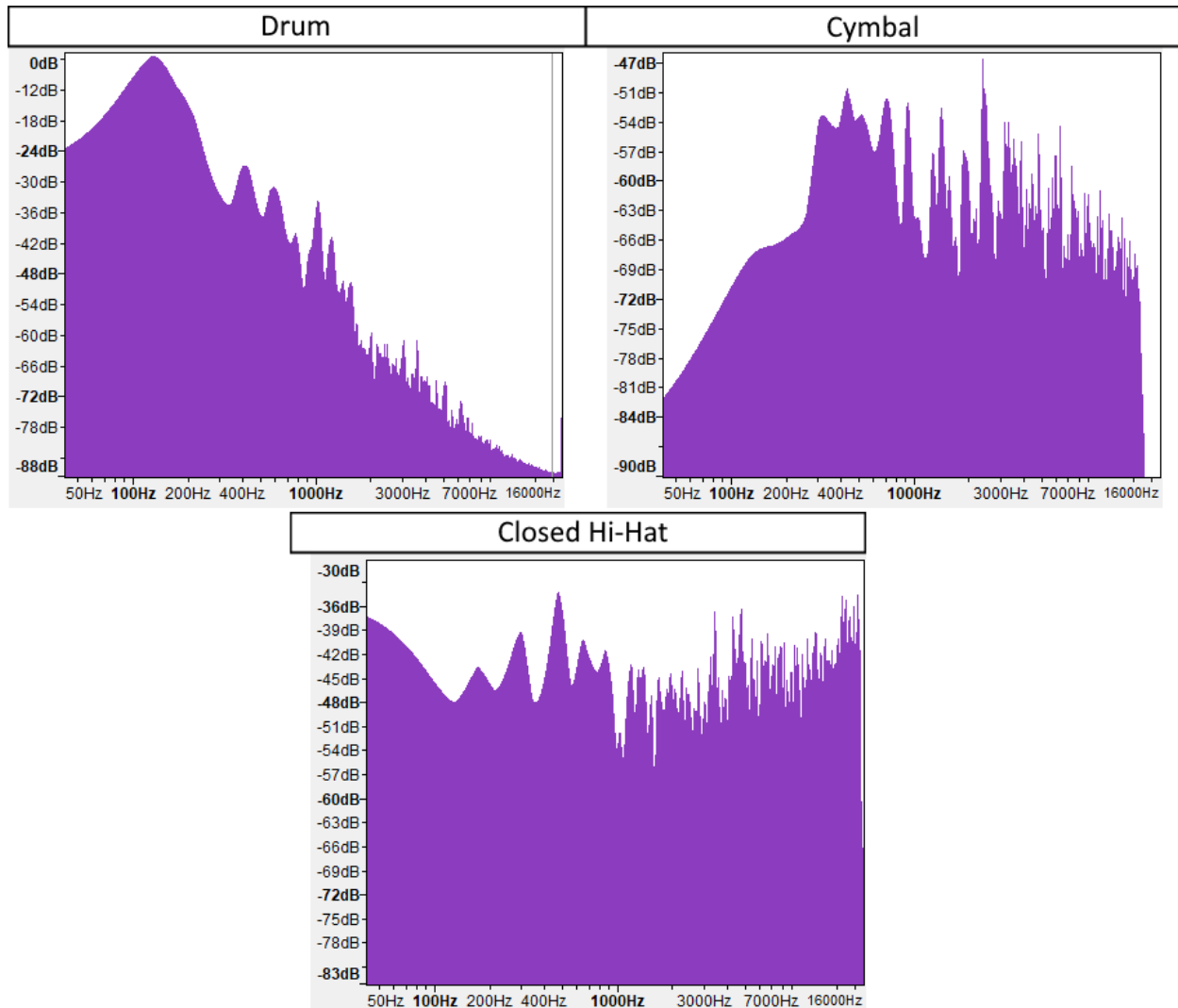


Figure 3.2.4.1 Frequency responses of a drum, cymbal, and closed hi-hat.

Because drums can be tuned for different frequencies, and the recording environment can influence the frequency captured in the audio sample, a more complex method for detection is needed. A useful method for finding shapes in an image is through the use of the Hough transform. This transform identifies primitive shapes within an image by summing the probabilities of the existence of all possible primitives existing on that particular pixel. While our application does not require the original Hough transform,



applying the concepts to the frequency response in one instance of time can help us identify a percussive note.

In our particular application, the modified Hough transform is applied looking for a normal distribution which is a similar shape to the frequency response of a drum at a particular location along the frequency axis. A matrix representing every reasonable normal distribution is created. The frequency response at a particular slice of time is then collected and each data point in that response is compared to each normal distribution that was outlined before. If the datapoint in question could be a part of a normal distribution, it adds one to that cell of the matrix. This is done for every data point and after which there should be a set of cells within the matrix that has a larger sum than all the others, thus giving us the most likely normal distribution and in turn giving us the frequency of the note played.

Table 3.2.4 — Design Items Governing Identification of Drum Hits

Design Item ID	Description	Requirement ID
D-3.2.4.1-B	The percussion identification algorithm will be able to distinguish which notes are percussion instruments, e.g. Drums, Cymbals, Hi-hats, etc.	R3.1.5-B, R3.1.5.2-B
D-3.2.4.2-A	The percussion identification algorithm will use a modified Hough Transform in the identification of drum notes	R3.1.2-A, R3.3-A

### 3.2.5 Full Spectrum Identification of Closed Hi-Hat Notes

One type of percussive instrument that can cause a lot of trouble is the closed hi-hat. This instrument, as shown in figure 3.2.4.1 adds power across the spectrum, almost similar to white noise. This can cause some of the previously mentioned algorithms to produce unexpected results. The most obvious problem that can occur is the lifting of bins that shouldn't be above the thresholding limit, which can cause other algorithmic processes to create false positives.

This however, can be very useful in trying to find the beat and tempo of a song. Hi-hat notes are strong indicators of the beat, or some portion of the beat. Therefore, identification of hi-hat notes provides a dual purpose, identifying hi-hat notes themselves and identifying when the beat within a song.

There are two possible methods that are possible to implement for finding hi-hat notes within a sample. The first method is counting the number of bins that are positive on the



frequency axis at each instance of time. If a certain number of bins are positive, then the whole spectrum at that point in time can be subtracted by the lowest bin, and this process is repeated until there are fewer than a threshold number of bins that are positive. This would hopefully filter out enough of the hi-hat to be reasonably processed by the rest of the algorithms. The threshold used would have to be adjustable by the user however, as setting the threshold too low can lead to the inadvertent filtering out of notes.

Another possible method would be to create a timbre profile for hi-hats and identify when a hi-hat note is present, and thus filter it out according to the timbre profile. This is the planned method for identification of other instruments. However, there is a problem in that the spectrum of a closed hi-hat hit, as shown previously, has magnitude across the spectrum similar to that of noise. Therefore, it can be hard to create a timbre profile for this instrument. Hard enough that identification using this method could lead to a very low accuracy during testing.

*Table 3.2.5 — Design Items Governing Identification of Hi-Hat Hits*

Design Item ID	Description	Requirement ID
D-3.2.5.1-A	The percussion identification algorithm will be able to Identify hi-hat notes	R3.1.5-B, R3.1.5.2-B, R3.2-B
D-3.2.4.2-A	The overall algorithm will be able to filter out hi-hat hits	-

### 3.3 Pitch Mapping

Once POIs are identified, getting a note’s timing and duration information is simply collected from the start- and end-points of the POI ridge, but obtaining the note’s pitch is not so linear.

In the frequency spectrum, the ideal from a processing perspective would be a delta function indicating exactly where the fundamental pitch of a note would be. However in reality this is not the case. As mentioned before in 3.2.2, when thresholding to obtain a POI in the frequency direction, there are multiple bins that will be marked as positive. If viewed in the log scale, this can cause different POIs to become ambiguous as to which particular pitch they belong to. To increase the accuracy and clarity for pitch mapping purposes, there are considerations that must be taken when dealing with the raw data itself.



### 3.3.1 Logarithmic Mapping of POIs to Pitch

The Discrete Fourier Transform, and by extension the standard spectrogram, categorizes power in linearly-spaced frequency buckets. However, pitch difference in music and to a listener operates logarithmically rather than linearly. In Western music theory, instruments are tuned using 12TET, with 12 tones in an octave equally spaced such that every tone is  $\sqrt[12]{2}$  time higher in frequency than the next-lowest tone.

Hence, mapping POIs to note pitch involves taking the logarithm of the frequency, finding the difference from a reference tone, and rounding to the nearest pitch identification. The standard reference tone is A4, the 49th key on the piano, which has a fundamental frequency of 440 Hz in concert tuning. Hence, the formula to map frequency to the index of a pitch would look as such, where 1 represents A0, the lowest key on the piano:

$$n = 12\log_2\left(\frac{f}{440 \text{ Hz}}\right) + 49$$

In the interest of giving users visibility into the music and the algorithm's note identification process, we have also decided to give users the option to display the spectrogram with a logarithmic-scale frequency axis to match how pitch perception actually works.

*Table 3.3.1 — Design Items Governing Mapping POIs to Pitch*

Design Item ID	Description	Requirement ID
D-3.3.1.1-A	The SMT note detection algorithm will map POIs to note pitches using a logarithmic formula	R3.1.2-A
D-3.3.1.2-B	The SMT will have an option to display the spectrogram with a log-scale frequency axis to match how listeners perceive pitch difference	R4.5.4-B





## 4 Software Design — Program State Management

### 4.1 Implementation Frameworks

#### 4.1.1 Qt Framework

Qt is a cross platform, event driven framework with a broad list of built-in components and responsive layout styling support [11]. The Qt Foundation maintains several flavours of the framework with the main C++ implementation as well as an officially supported python binding called pyside [11]. Both of these are included within an official IDE called Qt Creator which comes with plugin support, syntax highlighting, debugger and a built-in extensive layout designer [11].

*Table 4.1.1 — Design Items Governing the UI Framework*

Design Item ID	Description	Requirement ID
D-4.1.1.1-A	The SMT UI will be implemented using the Qt framework	R4.1-A R4.2-A R4.1.1-A



### 4.1.2 Project Layout Structure

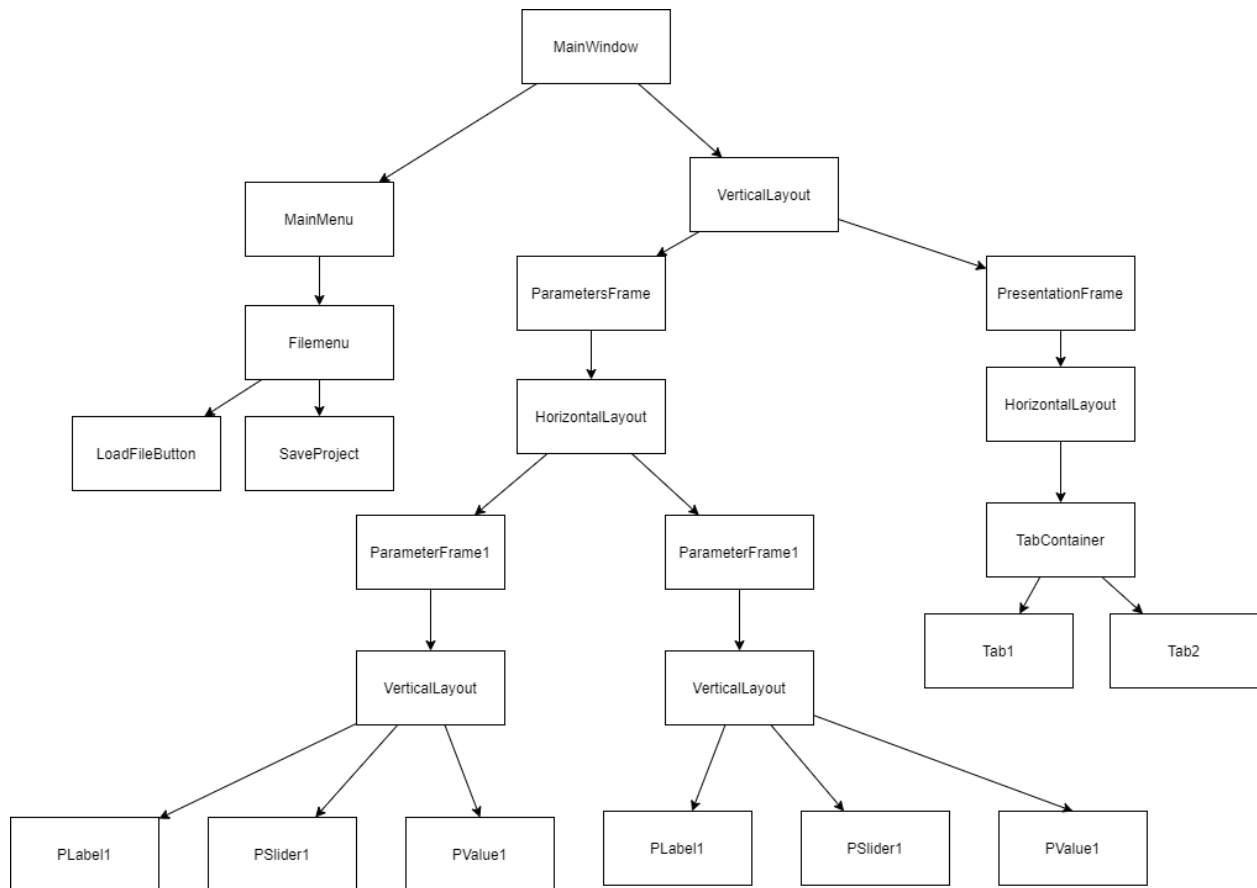


Figure 4.1.2 — Layout Structure Example

Figure 4.1.2, provides an example of the layout structure that is implemented within our project. The layout follows a tree like structure which is crucial for providing responsive positioning [12]. Responsive positioning is required for supporting different form window sizes as well as successfully adjusting to different screen sizes without compromising on the overall readability and visibility.



Table 4.1.2 Design Items governing project layout structure

Design Item ID	Description	Requirement ID
D-4.1.2.1-A	A user should be able to view program window on different screen sizes without compromising on the readability and visibility of the application	R4.1-A

### 4.1.3 Qt Event framework

Qt is an event driven framework which means that elements are capable of generating events that can be handled, based on predefined behaviour. In QT, these events are referred to as “signals”[7]. Each built-in element class provides a basic set of signals that can be emitted with various user interactions such as clicking, dragging, focus acquisition, toggle, etc [7]. Signals must be connected to event handlers which are called “slots” in Qt terminology, and they are just function definitions that are triggered by connected signals that execute desired behaviour [7].

As parameters are being applied via UI elements, we will connect the “Apply” button signal to a slot function that will act as the entry point for executing sound analyses sub-routines. Assuming that sub-routine is synchronous, we will proceed loading the results from the intermediate format to the UI presentation elements. Intermediate format will be further processed by the algorithms and saved to disk to keep processed output results. The described procedure is illustrated in the Figure 4.1.3 below.

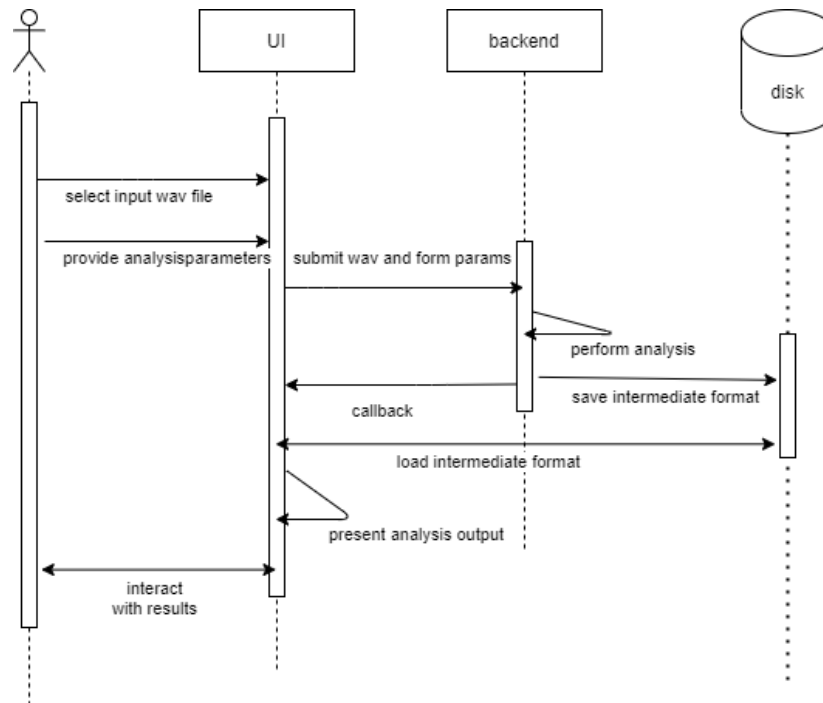


Figure 4.1.3 Overall program workflow.

Note: the above diagram uses .wav as an example input file while the program itself is not limited to this format

## 4.2 Intermediate Data Properties

There are 3 main types of data that needs to be passed the subsystems in the SMT: parameters adjusted by the user, raw music data, and the processed data. The whole project that the user is working on also needs to be able to be saved to the disk in some format for later use by the user. The performance of this data is integral to the whole operation of the whole program.

### 4.2.1 User Adjustable Parameter Management

Parameters being passed from the UI need to be read by the backend. These parameters need to be saved in a readable format such that the algorithmic system does not slow down. A bottleneck in this portion of the program would cause the user experience to be hampered to a large degree, and as such should have very specific design requirements related to it.



Table 4.2.1 Design Items Governing User Adjustable Parameters

Design Item ID	Description	Requirement ID
D-4.2.1.1-A	The SMT backend should extract parameter values from the UI when values are updated by the user	-
D-4.2.1.2-A	The SMT backend should save these parameter values in a format that is readable by the algorithmic subsystem	-
D-4.2.1.3-A	The preprocessed music data should be able to be read quickly by the algorithmic subsystem	-

#### 4.2.2 User Input Raw Music Data Management

The specifications related to this set portion of data should be fairly simple. An array of magnitudes with a separate array listing the relevant metadata of the music file (such as length, sampling frequency, etc) would be sufficient enough for the algorithmic sub-system to utilise. One feature that should be in this section however is a processing stage, converting from different music files to the aforementioned arrays. This will simplify the algorithmic processing and make for a much easier flow of data. This raw music data should be easily readable by both the algorithmic subsystem and the UI subsystem, as this data has to be processed by both. The algorithmic system has to operate on this data in order to create spectrogram data, detect POIs, and detect the beat within the music among other things. The UI will have an option to display the waveform of the music, and thus will need access to the data as well in order to create the corresponding waveform view should the user choose to view it. This portion of data should also be very easily readable by both subsystems, otherwise the user experience will suffer due to a large processing time.

Table 4.2.2 Design Items Governing User Input Music Data

Design Item ID	Description	Requirement ID
D-4.2.2.1-A	The SMT should be able to pre-process multiple music formats including: WAV, MP3, FLAC	-
D-4.2.2.2-A	The SMT should store the relevant metadata in a standardized array for the use of both subsystems	-



D-4.2.2.3-A	The SMT should store the preprocessed music data in an array for the use of the both subsystems	-
D-4.2.2.4-A	The preprocessed music data should be able to be read quickly by both subsystems	-

### 4.2.3 Algorithm Post-Processed Data Management

This set of data will contain the data processed by the algorithmic subsystem. This data should be similar to the raw music data provided by the user. This data contains the spectrogram data, points identified as POIs, and any other processed data. This data should also be easily readable by both subsystems, as this data needs to be displayed by the UI subsystem, and the algorithmic subsystem might have to process portions of this processed data in order to further produce more results.

Table 4.2.3 Design Items Governing Post-Processed Data

Design Item ID	Description	Requirement ID
D-4.2.3.1-A	The SMT should store the relevant metadata in a standardized array for the use of both subsystems	-
D-4.2.3.2-A	The SMT should store the processed music data in an array for the use of the both subsystems	-
D-4.2.3.3-A	The processed music data should be able to be read quickly by both subsystems	-

### 4.2.4 Saving and Loading

The whole project that a user is working on should be able to be saved for future use by the user. This save state should not take too long to read from and write to, otherwise the user experience will suffer. The SMT should also not re-write any user input music data if possible, as this would lead to a significant increase in processing time.

Table 4.2.4 Design Items Governing Project Saving and Loading

Design Item ID	Description	Requirement ID
D-4.2.4.1-A	The active SMT project should be able to be written to a user specified location for future use in the form of a save file	-



D-4.2.4.2-A	The active SMT project should be able to store project parameters in the save file	-
D-4.2.4.3-A	The active SMT project should be able to store preprocessed music data in the save file	-
D-4.2.4.4-A	The active SMT project should be able to store processed music data in the save file	-
D-4.2.4.5-A	The SMT project state should be able to be read from a user specified location	-
D-4.2.4.6-A	The SMT should be able to read project parameters from the save file	-
D-4.2.4.7-A	The SMT should be able to read preprocessed music data from the save file	-
D-4.2.4.1.8-A	The SMT should be able to read processed music data from the save file	-
D-4.2.4.1.9-A	The SMT should not re-write over existing user input music data if it has not changed	-



## 5 Software Design — User Interface

The User Interface of SMT is the medium between user and music files (data). Our user interface design aims to bring users the minimalist, practical and aesthetic visual style. We are providing a dark theme, which is comfortable for eyes in both bright and dim environments. We are creating an interactive and intuitive interface: buttons, sliders, tabs and windows will be responsive and informative; symbols will be used to replace texts and provide shortcuts for various features. The structure will be clear and straightforward, functions will be grouped and categorized. We are also implementing animated widgets, to make our UI appearance more modern and stylish.

The tools used to design and implement the user interface are figma, pyside6, python 3, Qt designer. Figma is used for concept design. PySide 6 and Qt Designer are used for graphical component implementation. Python 3 is used for animation widget and toning for the user interface. The output file (.ui) from Qt Designer can be converted to a python file (.py) and then can be used to do detailed editing. The UI development will be splitted into three parts: 1. Menubar and topbar. 2. Left sidebar. 3. Main display.

The aesthetic design of the UI will follow specific colour themes. The symbols will also follow uniformed patterns. The structure and shortcut design will follow Windows 10 operating system style.



Figure 5.1 User Interface Framework





The figure 5.1 above shows the framework for the user interface, where menubar, sliders and buttons have been added. The figure also shows the option to change between different views.

## 5.1 Use Cases

Table 5.1 Design Items Governing UI Use Cases

Design Item ID	Description	Requirement ID
D-5.1.1-A	User should be able to see the program as an icon as .exe file on Windows 10	-
D-5.1.2-A	User should be able to open the software by double click on the program icon	R4.1-A
D-5.1.2.1-A	User should be able to run the program from console (Windows command prompt)	-
D-5.1.3-A	User should be able to use dark theme	R4.11-A
D-5.1.4-A	User can use the menu bar to manipulate files	R4.2-A
D-5.1.4.1-A	User can import a file from file system	-
D-5.1.4.1.1-A	User can find the file by navigating through the file system with a graphic interface	-
D-5.1.4.1.2-B	User can import a MusicXML (.xml) file	-
D-5.1.4.1.3-A	User can import a mp3 (.mp3) file	R4.2.2-B
D-5.1.4.1.4-A	User can import a wav (.wav) file	R4.2.1-A
D-5.1.4.1.5-B	User can import a MIDI (.mid) file	R4.2.2-B
D-5.1.4.1.6-B	User can import a OGG (.ogg) file	R4.2.2-B
D-5.1.4.2-A	User can export a file to the file system	-
D-5.1.4.2.1-A	User can open a window and choose location to export the file	-
D-5.1.4.2.2-B	User can export a MusicXML (.xml) file	R4.9-V1
D-5.1.4.2.2.1-B	User should be able to specify score presentation such as title and performer	R4.9.1-V1



D-5.1.4.2.3-A	User can export a PDF (.pdf) file	-
D-5.1.4.2.4-B	User can export a MIDI (.mid) file	R4.8-B
D-5.1.4.2.4.1-B	The user must have an input method to write track information to an exported MIDI file	R4.8.1-B
D-5.1.4.2.5-A	User can export spectrum amplitudes (.csv)	-
D-5.1.4.2.6-B	User should be able to export certain parts, filtered by timbre	R4.9.2-V1
D-5.1.4.3-A	User should be able to open a previously saved file	R4.18-B
D-5.1.4.4-A	User should be able to save the progress	R4.17-B
D-5.1.4.4.1-A	User can use shortcut to save the file	R4.17-B
D-5.1.4.5-B	User should be able get help by searching keywords	-
D-5.1.5-A	User can use the left side panel to adjust parameters	R4.6-B
D-5.1.5.1-A	User should be able to configure thresholds on what the software considers a note	R4.6.1-B
D-5.1.5.2-B	The user should be able to configure the maximum number of timbres to detect in an audio sample	R4.6.2-V1
D-5.1.6-A	User can use top bar to edit views	-
D-5.1.7-A	User should be able to save the file directly by click on the icon in the top bar	-
D-5.1.8-A	User can see different views in the central window	R4.5-A
D-5.1.8.1-B	User can see sheet music	R4.5.3-B
D-5.1.8.1.1-B	User should be able to change the key signature of displayed sheet music	R4.5.3.1-B
D-5.1.8.1.2-B	User should be able to change the base note duration for the displayed sheet music between half notes, quarter notes, eighth notes, or sixteenth notes	R4.5.3.2-B



D-5.1.8.2-A	User can see waveform view	R4.5.1-A
D-5.1.8.3-A	User can see the spectrum view	R4.5.4-B
D-5.1.8.3.1-B	User can see pitch, timing, and duration of notes on the spectrum view	R4.4-A
D-5.1.8.3.2-B	user should be able to place markers with text annotations on specific points of the spectrogram	R4.5.4.1-B
D-5.1.8.4-B	User can display a piano roll of detected notes	R4.5.2-B
D-5.1.8.4.1-B	User should be able to edit existing notes in the piano roll view	R4.7-V1
D-5.1.8.5-A	User should be able to zoom in and out of the various views	R4.5.6-A
D-5.1.8.6-A	User should be able to label identified timbres	R4.10-V1
D-5.1.8.7-A	User should be able to close views	-
D-5.1.8.8-A	User should be able to swap views	-
D-5.1.8.9-A	User can see explanation when hover over a interactive component for more than 0.4 second	-
D-5.1.9-A	User can stop the software and close the window	R4.1.1-A
D-5.1.9.1-A	User should be able to see a popup window warning the user when exit	-
D-5.1.9.1.1-A	User should be able to disable the popup window and exit directly	-
D-5.1.10-A	User can minimize the window	-
D-5.1.11-B	User should be able to record from an external microphone using the program	R4.3-B



## 5.2 Theme

In modern UX (User Experience) and UI (User Interface) design, colour choice becomes more and more important. Different colours evoke different emotions [5]. After doing research and experiments, we eventually chose the dark theme with blue tones. According to the research, blue is considered the most preferred color universal, many most commonly used apps, such as Microsoft, Twitter, Facebook, Skype, etc [5] [6]. Dark UIs are often associated with power, elegance and mystery, which has become a formidable trend in recent years[7]. Dark mode is often believed to be able to not only reduce eye strain but also save battery life under certain circumstances [7].

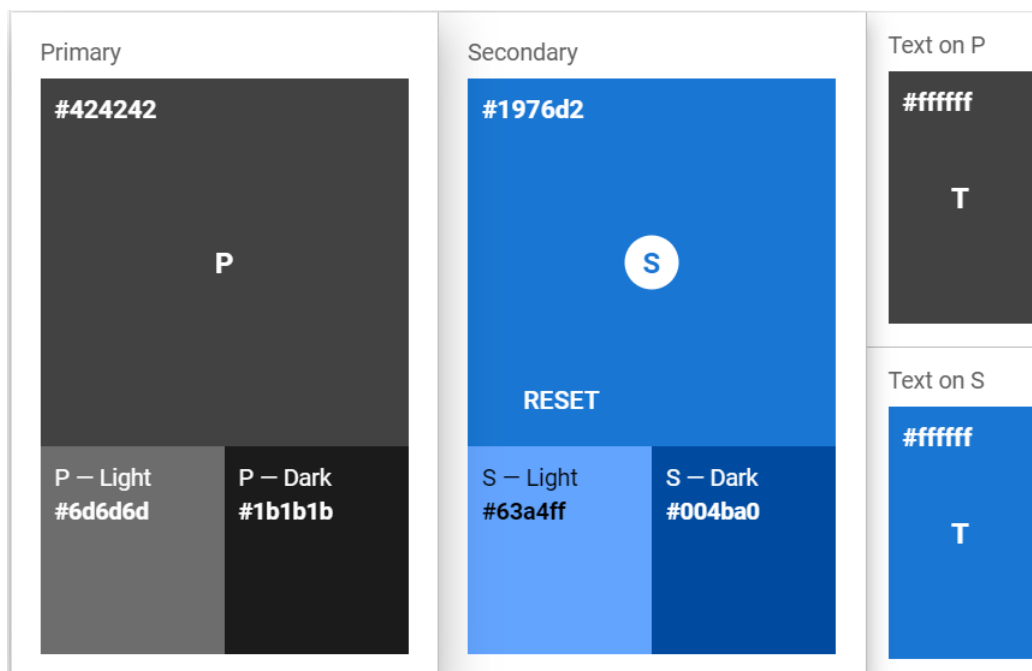


Figure 5.2.1. Color Palette for UI theme

Figure 5.2.1 above shows a color palette with dark grey as primary color, and blue as secondary color. In the User Interface, we will follow the color palette shown here. The primary color is the main color for dark mode and the secondary color is for functionalities.

## 5.3 Structure

The functional area has been divided into 3 parts, topbar (includes menubar), sidebar and central window. Different parts hold different functionalities and features corresponding to the use cases. Menu Bar is for navigating and modifying the files; top bar is a shortcut for menu bar's functionalities; sidebar can be used to adjust the



parameters and the central window will be used to display sheet music, audio spectrum, audio waveform and piano roll.

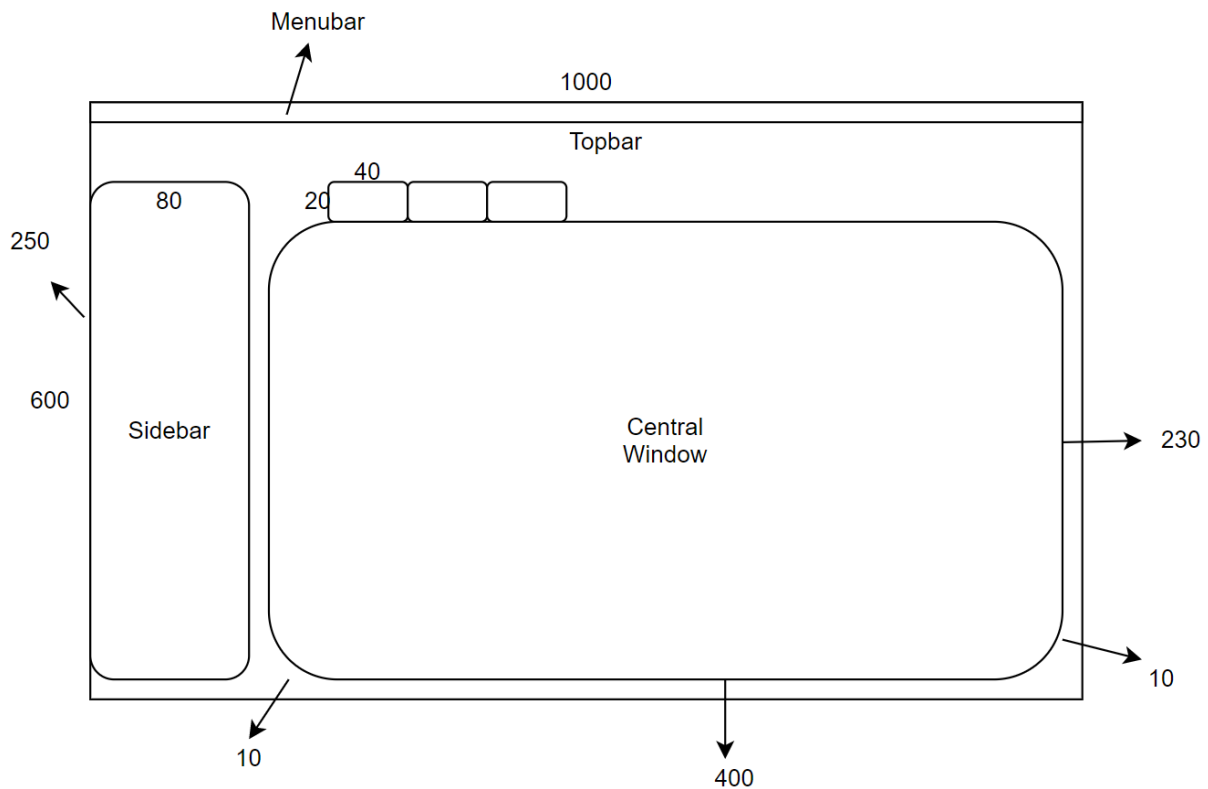


Figure 5.3.1 A scaled graph of the structure of user interface, which can be use as a reference to Figure 5.1

File	Edit	View	Help	
File	Edit	View		
Open	Copy	Show Waveform		
Import	Paste	Show Spectrum		
Export	Delete	Show Piano Roll		
Save	Change Frequency	Show Sheet Music		
Save As...	Change Key	Zoom In		
Settings	Add Filter	Zoom Out		

Figure 5.3.2 scheme for menu bar.

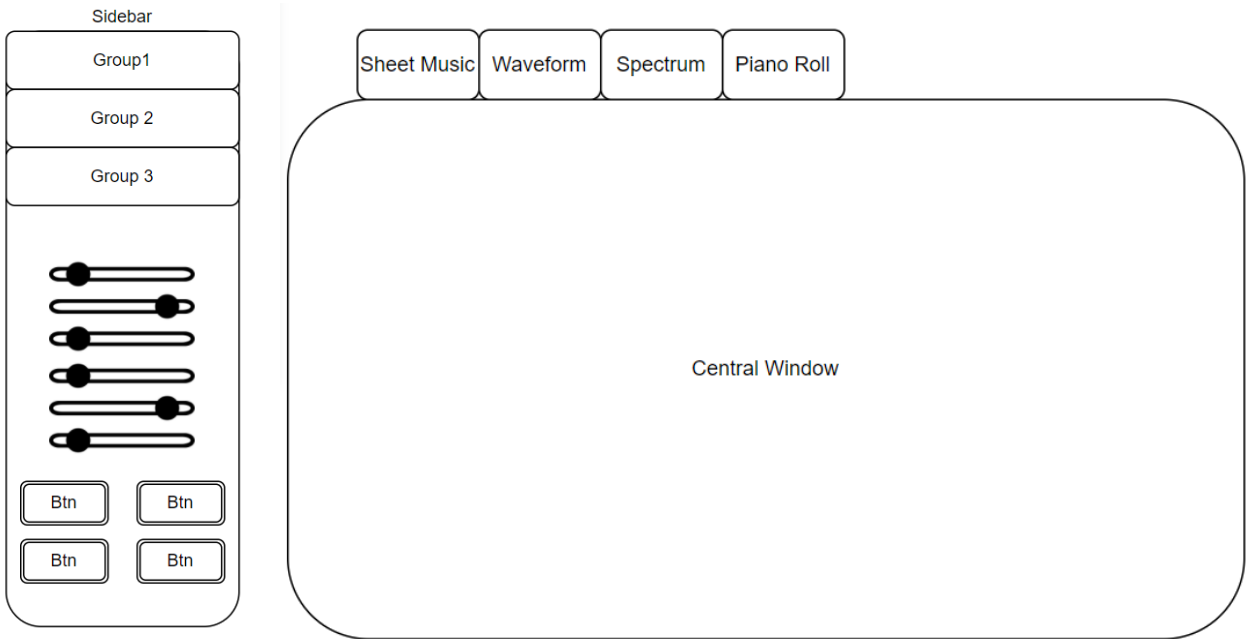


Figure 5.3.3 main windows including sidebar and Central Window, which displays sheet music, audio waveform, audio spectrum and piano roll.



## **6 Conclusion**

This Design Specification Document summarizes the design choices for Sheet Music Transcriber by HappyJam. The main purpose of our project is to make the task of music transcription interactive and hassle free for professionals as well as aspiring artists. The design decisions outlined in this document should offer substantial help in allowing musicians to analyze music and create scores with far less work than a traditional manual method. For learner or aspiring musicians, the SMT will help bridge the knowledge gap needed to make a music score from audio.

The algorithmic system of the SMT will generate a spectrogram using the fourier transform, identify Points of Interest (POIs), identify any percussion, and find the tempo of a piece of music. The program state management system will make sure the relevant data is available to all parts of the system in a format that is computationally efficient. Furthermore this system will be able to save its state to the disk and read from the disk to enable a user to save their work and load it at a future time. The User Interface (UI) will take a clean and minimalist approach, providing the user clarity and ease of use in order to facilitate a positive user experience. The color palette of this system is chosen to be pleasing to the eye and the layout will be intuitive for any user who uses the software.

These design specifications are aimed at creating the best music transcription aid for any musician to use. HappyJam is fully confident that the SMT will create a seamless process for any musician, lowering the bar for transcription from that of a tedious task to that of a fun one. Because in the end, music should be fun.



## 7 References

- [1] K. Fitz, “Time-Frequency Analysis,” *CERL Sound Group*, 15-Jan-2010. [Online]. Available: <http://www.cerlsoundgroup.org/Kelly/timefrequency.html>. [Accessed: 01-Jul-2021].
- [2] K. R. Fitz and S. A. Fulop, “A Unified Theory of Time-Frequency Reassignment,” *arXiv*, 30-Sep-2005. [Online]. Available: <https://arxiv.org/pdf/0903.3080.pdf>. [Accessed: 01-Jul-2021].
- [3] “Window function,” *Wikipedia*, 08-Jun-2021. [Online]. Available: [https://en.wikipedia.org/wiki/Window\\_function#A\\_list\\_of\\_window\\_functions](https://en.wikipedia.org/wiki/Window_function#A_list_of_window_functions). [Accessed: 03-Jul-2021].
- [4] S. Vaniukov. “Colors in UI Design: A Guide for Creating the Perfect UI”. UsabilityGeek. [Online]. Available: <https://usabilitygeek.com/colors-in-ui-design-a-guide-for-creating-the-perfect-ui/> [Accessed: 07-Jul-2021].
- [5] T. Liu. “How To Use Color In UI Design Wisely to Create A Perfect UI Interface?”. UX Collective. 17-Oct-2017. [Online]. Available: <https://uxdesign.cc/how-to-use-color-in-ui-design-wisely-to-create-a-perfect-ui-interface-2af42f901f4>. [Accessed: 03-Jul-2021].
- [6] M. PHILIPS. “In the Spotlight: the Principles of Dark UI Design”. [Online]. Available: <https://www.toptal.com/designers/ui/dark-ui-design>. [Accessed: 03-Jul-2021]
- [7] “Signals & Slots: Qt Core 5.15.5,” *Signals & Slots | Qt Core 5.15.5*. [Online]. Available: <https://doc.qt.io/qt-5/signalsandslots.html#signals-and-slots>. [Accessed: 11-Jul-2021].
- [8] E. Huang, *5 Programming Language That Produce Code Least Prone to Bugs*, 2021. [Online]. Available: <https://edward-huang.com/programming/software-development/2021/03/02/5-programming-language-that-produce-code-least-prone-to-bugs/#expressiveness>. [Accessed: 11-Jul-2021].
- [9] L. Hohmann, in *Beyond software architecture: creating and sustaining winning solutions*, Boston: Addison-Wesley, 2008, Chapter 5.
- [10] “System call,” *Wikipedia*, 18-Jun-2021. [Online]. Available: [https://en.wikipedia.org/wiki/System\\_call#Privileges](https://en.wikipedia.org/wiki/System_call#Privileges). [Accessed: 11-Jul-2021].
- [11] “About Qt,” *About Qt - Qt Wiki*. [Online]. Available: [https://wiki.qt.io/About\\_Qt](https://wiki.qt.io/About_Qt). [Accessed: 11-Jul-2021].





- [12] “Layout Management: Qt 4.8,” *Layout Management | Qt 4.8*. [Online]. Available: <https://doc.qt.io/archives/qt-4.8/layout.html>. [Accessed: 11-Jul-2021].



## Appendix A - Test Plan

### A.1 Algorithm Testing

Algorithmic testing should align with two overall purposes of our program, identify and separate notes, and process this information to be usable by the user for adjustments.

*Table A.1.1 General SMT Algorithmic Functionality Testing Plan*

Testing Item ID	Description	Result	Comments
D-A.3.1.1-A	The SMT identifies the times at which an instrument plays a note in an audio sample		
D-A.3.1.2-A	The SMT identifies the pitches at which an instrument plays a note in an audio sample		
D-A.3.1.3-B	The SMT identifies the lengths of a notes played by an instrument in an audio sample		
D-A.3.1.4-B	The SMT identifies the tempo of the sample audio		
D-A.3.1.5-B	The SMT identifies the relative volume of a notes played by an instrument in an audio sample		
D-A.3.1.6-V1	The SMT identifies a tempo changes in the audio sample		
D-A.3.1.7-A	The SMT algorithm packages data processed by the algorithm in a format usable by other portions of the program and the user		

*Table A.1.2 Single Instrument Testing Plan*

Testing Item ID	Description	Result	Comments
D-A.3.2.1-A	The SMT identifies and processes notes from a single instrument audio sample		



D-A.3.2.1.1-A	The SMT identifies the notes of a piano audio sample and displays them accurately		
D-A.3.2.1.2-A	The SMT identifies the notes of a guitar audio sample and displays them accurately		
D-A.3.2.1.3-B	The SMT identifies the notes of a drum-set audio sample and displays them accurately		
D-A.3.2.2-A	The SMT identifies the overtones of a single instrument and eliminate them in an audio sample		
D-A.3.2.2.1-A	The SMT identifies which instrument is playing in an audio sample based on a timbre profile		

Table A.1.3 Dual Instrument Testing Plan

Testing Item ID	Description	Result	Comments
D-A.3.3.1-B	The SMT identifies and processes notes from two instruments simultaneously in an audio sample		
D-A.3.3.1.1-B	The SMT identifies the notes from both a piano and guitar at the same time in an audio sample		
D-A.3.3.1.2-B	The SMT identifies the notes from a piano and violin at the same time in an audio sample		
D-A.3.3.1.3-B	The SMT identifies the notes from a piano and drum set at the same time in an audio sample		
D-A.3.3.1.3.1-B	The SMT identifies and filters out any notes related to percussion that interferes with note detection of other instruments		
D-A.3.3.1.4-B	The SMT identifies which instruments		



	are playing which notes in an audio sample		
D-A.3.3.2-B	The SMT identifies and eliminates overtones of each instrument without disrupting the note identification of the other instrument		

*Table A.1.4 Multi-Instrument Testing Plan*

Testing Item ID	Description	Result	Comments
D-A.3.3.1-V1	The SMT identifies and processes notes from multiple instruments simultaneously in an audio sample		
D-A.3.3.1.3.1	The SMT identifies and filters out any notes related to percussion that interferes with note detection of other instruments		
D-A.3.3.1.4	The SMT identifies which instruments are playing which notes in an audio sample		
D-A.3.3.2	The SMT identifies and eliminates overtones of each instrument without disrupting the note identification of the other instruments		

## A.2 Project Manager Testing

*Table A.2.1 Project Manager Test Plan*

Testing Item ID	Description	Result	Comments
D-A.4.1.1-A	The SMT is able to load audio files of various formats (.wav, .mp3, etc.)		
D-A.4.1.2-A	The SMT is able to save parameters from the imported audio file		
D-A.4.1.3-A	A change in parameter values is registered by the algorithmic subsystem		



D-A.4.1.4-A	The SMT can save user defined parameter values		
D-A.4.1.5-A	The SMT can save post processing data as a project		
D-A.4.1.6-A	The SMT can load parameter values saved as a project		
D-A.4.1.7-A	The SMT can process data from imported file		



## Appendix B - Supporting Design Options

### B.1 Algorithm Design Options

In the course of developing, prototyping, and researching the note detection algorithms, we reviewed many alternative options and algorithms.

#### B.1.1 Window Function Options

Wikipedia lists several commonly used window functions [3], including but not limited to:

- Rectangular window
- Parzen window
- Welch window
- Sine window
- Hann window
- Hamming window
- Blackman window
- Gaussian window

As discussed in section 3.1.2, window functions will have different Fourier Transforms which will create different magnitudes and locations of artifacts around peaks in the spectrogram.

During prototyping, we used Hann, Hamming, and Blackman windows with success. All three worked well thanks to their tight concentration of power around the central lobe of their Fourier Transforms. We ultimately settled on Hann windows due to its use of time-frequency reassignment as outlined by K. Fitz and S. Fulop [2]. After having time to experiment with their interoperability with the time-frequency reassignment algorithm and modified Hough Transform discussed in section 3.2.4, we may give the user the option to select a window from any of these.

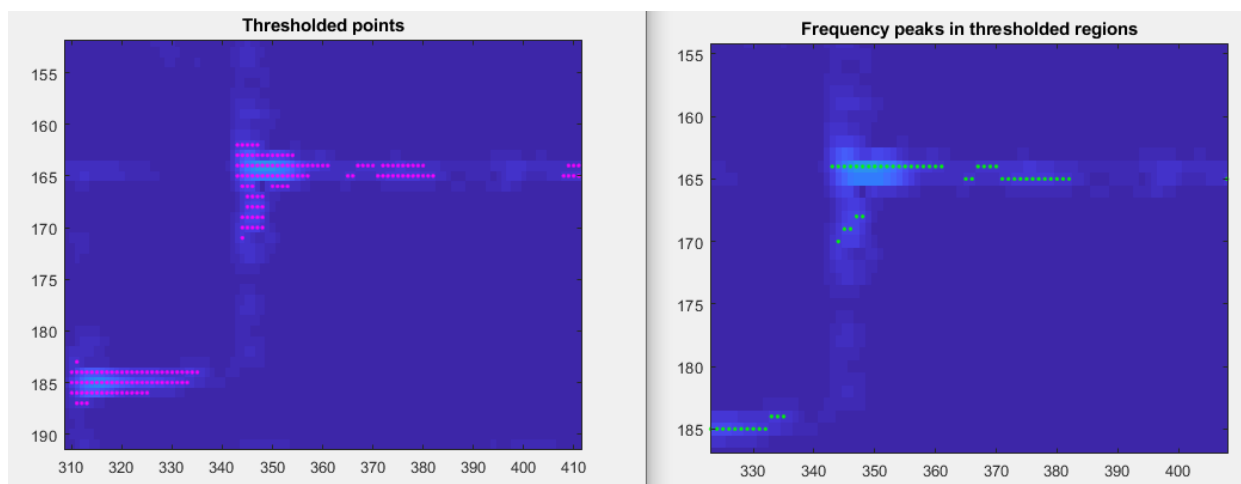
#### B.1.2 POI Identification Alternatives

POI identification currently relies on thresholding and peakfinding, which generally narrow in on regions of interest and specific points respectively. There are alternative strategies for both phases we have opted not to use.

As outlined in section 3.2.3, it is possible to obtain information from the curvature of the power in the spectrogram. We use downward curvature along the time-axis to indicate onset incidence, but downward curvature along the frequency axis could indicate harmonic peak presence and be seen as a convolutional analog to region of interest identification. However, downward curvature could also come from noise or from wide-peak percussive notes. As this method failed to filter these regions, we instead opted for simple thresholding as the initial phase of POI identification.



There are more complicated POI identification algorithms we have considered but have not fully experimented with, largely due to their complexity. At their core is the idea of processing each connected region of interest as a separate entity. Doing that enables algorithms that search for the maximum frequency in the region instead of just frequency peaks. This offers the potential to reduce detection of unwanted ridges like we see in Figure B.1.2.1, at the cost of much more computer spent identifying each connected region and processing it independent of the rest of the image.



*Figure B.1.2.1 — An unwanted POI ridge that is part of the same connected region of interest as the main ridge. Using a per-region maximum detection algorithm would keep this unwanted ridge from being highlighted as a POI.*

Unfortunately, as discussed in section 3.2.1 and shown in Figure 3.2.1.1, regions of interest often merge together for closely spaced notes. For different pitches played at the same time within this same region, the two POIs will interfere. In the best case, the ridge containing more power will dominate over the one containing less power, preventing it from being detected. In the worst case, both ridges will rapidly alternate which is higher and create a “chattering” effect — that is, rapid breaks in the detected ridge. As such, we have elected to pass these algorithms over in favour of peak-finding, which can be more easily performed for the entire spectrogram at once and is more likely to lead to POIs that look like continuous ridges.

One alternate strategy we are hoping to implement in future is dynamic threshold adjustment. At the simplest level, this would involve changing the region of interest threshold to be higher or lower for pixels farther along the frequency axis. This would allow the user to compensate for any peculiarities of the recording or instrument that may lead to certain registers being consistently higher or lower than others. At a more advanced level, we could detect fundamentals and use that to reduce the threshold at



the expected locations of the fundamentals' overtones, allowing for more consistent overtone detection which could lead to more detailed and consistent timbre profile construction down the line. As these downstream algorithms have not been implemented yet, and the UI elements that would give the user this fine control over the threshold haven't been developed yet, we will hold off on further investigating this algorithm until after the alpha release.

### B.1.3 Onset Detection Alternatives

As we have used peakfinding along the frequency-axis to detect POIs, it makes sense to ask what peaks along the time-axis can be used for. During prototyping, we used these points to indicate likely note onset points. However, the nature of peakfinding leads to choosing more sparsely-positioned points, and hence is worse at differentiating between random noise and the actual onset. Compare Figure B.1.3.1 below using peakfinding and Figure 3.2.3.2 in section 3.2.3, which show the same section of the same spectrogram but with different point highlighting methods, to see this in action. While both are rather noisy, the curvature-based onset detector has a greater density of points near the actual onset of the notes than the peak-finding based one.

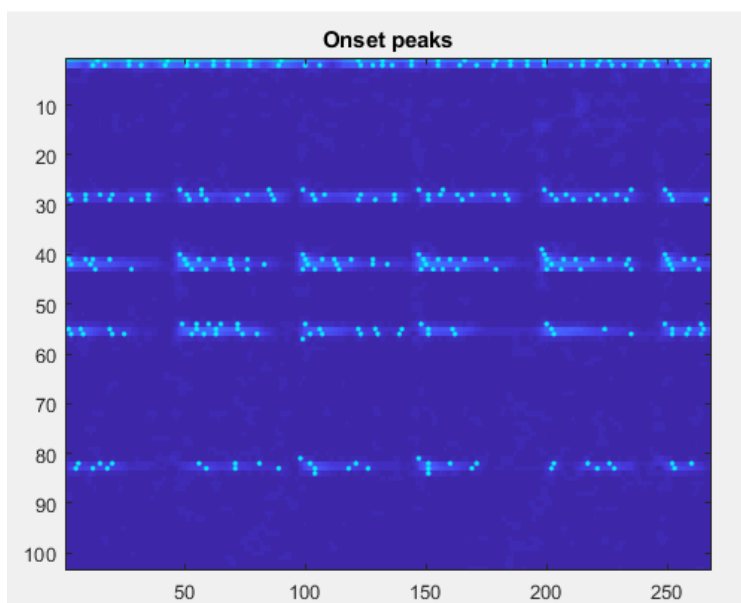


Figure B.1.3.1 — A thresholded spectrogram with peaks along the time-axis highlighted in cyan.

### B.1.4 Note Mapping Enhancements

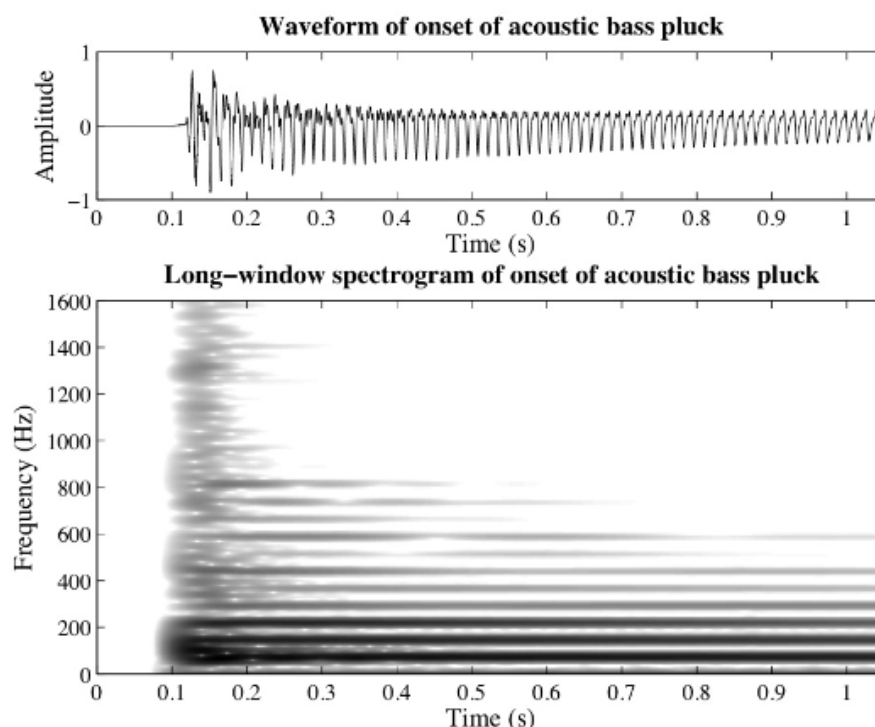
Our current POI identification algorithms map straight from pixel position to note pitch. However, as the mapping is from a linear-scale view to a logarithmic-scale view, bass



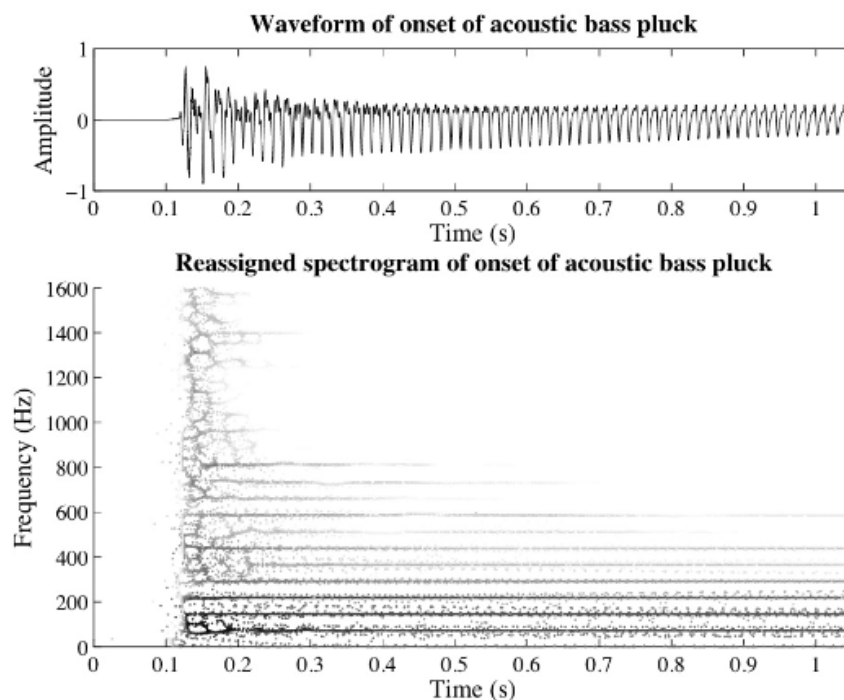


notes are actually very closely-spaced. This means that error can be rather high especially for bass notes.

One possible course to reduce this error is to use time-frequency reassignment [2]. This is an algorithm that uses phase information in a spectrogram to remap pixel coordinates to the location of the greatest power contributor. Observe Figures B.1.4.1 and B.1.4.2, which show unmodified and time-frequency reassigned spectrograms of a bass audio sample, and note how the reassigned spectrogram gives very detailed information on the note in both time and frequency domains.



*Figure B.1.4.1 — A spectrogram of an acoustic bass playing a note around 73.4 Hz, showing off the time-smearing with a normal spectrogram. Taken from A Unified Theory of Time Frequency Reassignment [2]*



*Figure B.1.4.2 — Reassigned spectrogram of an acoustic bass playing a note around 73.4 Hz. Taken from A Unified Theory of Time Frequency Reassignment [2]*

We have yet to implement and fully experiment with this algorithm, though it provides the possibility of getting much more accurate time- and pitch-mapping for our POIs. We have not committed to using this algorithm because we are unsure of a prerequisite to using it, as outlined in the paper [2] — separability. If multiple power sources overlap their contributions at the same time and frequency, then the reassignment is potentially thrown off. For that reason, if we do include this algorithm in the final SMT algorithm, it will likely come with the option for the user to turn it off or make adjustments to the resulting reassigned POIs.



## B.2 Technical Stack Selection

### B.2.1 Implicit requirements

Below we summarized implicit requirements derived from desired user experience and project timeline constraints. They expand upon explicit requirements items within the requirements document and as such will not be presented in the usual indexed format. We will refer to the requirements as IR-A for implicit requirements.

#### B.2.1.1-A Performance

Audio analysis can be a computationally intensive task from the most basic sampling techniques to the more ambitious goals that might even require AI algorithms. Some stacks are better suited than others for providing optimization potential to the developers. Performance will have a direct impact on our user experience in the time sensitive and competitive environment.

#### B.2.1.2-A Syntax complexity

Some languages are more expressive than others. In a time-sensitive development environment, that can result in a huge difference in our ability to deliver on our promised feature set. More lines of code take more time to write code physically but also can bring higher risks by exposing more potential for introducing errors in the code [8]. It is often the case that higher level languages take away some of the control of resource allocation. Ideally, we want to find a language with reduced syntax complexity without compromising on the desired performance, mentioned above.

#### B.2.1.3-A Third-party and community support

(Libraries, Github resources, Stackoverflow, etc)

In the interest of saving as much time as possible we are motivated to conduct a thorough exploration of the third-party resources available to the various language options. Ideally we want to settle on the language with the most well established existing libraries available for satisfying the various aspects of our application. Primarily, this includes the availability of audio analysis libraries and UI components as well as audio and display device interface support.

Community support goes beyond just having the appropriate libraries as it is often the case that commonly encountered problems have associated materials published across various online resources and documentation material. The more we can avoid “reinventing the wheel”, the more we can increase our throughput in feature delivery without compromising the implementation quality [9].



### B.2.1.4-A Integrated development environment support and dev tooling

Some stacks provide a very extensive set of tools for code generation especially when considering user interface layouts. Any such tool should give us a dramatic edge to focus on the more important aspects of our already challenging project. Code completion, syntax highlighting and common formatting can serve to further optimise our efficiency as developers.

### B.2.1.5-V1 Cross platform compatibility

It should come as no surprise that the success of the project would benefit from targeting the widest possible user base by including support for as many platforms as possible. Today this implies the need to support at least the following major platforms

- Windows
- macOS
- Linux

As we all know these major platforms share some similarities but can rely on some specific layout and API differences that make them unique in their own right [10]. Developing platform-specific applications can give us access to some of the more unique features available to each platform, however this does come at the cost of increased development time required to adapt the implementation.

### B.2.1.6-V1 Software distribution and packaging

We should aim to provide a self contained package for distribution to the end customer. We cannot rely on the external third-party platform for executing our final solution, especially those that require separate licensing such as Matlab.

## B.2.2 UI frameworks overview

*Table B.2.2.1 Base Design Specification Table for the UI framework*

Design Item ID	Description	Requirement ID
B.2.2.1 - V1	Software should be easily adjusted for running our application on different OS such as Linux, Windows, macOS without sacrificing team developer's time	<b>IR5-V1</b>
B.2.2.2 - A	Software needs to be able to make system calls to the underlying OS to be able to utilize hardware resources,	<b>IR1-A</b>

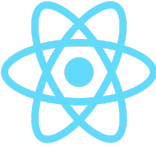




	required for fast algorithmic processing as well local data storage	
B.2.2.3 - A	Application does not expose unsecure sockets that can be exploited by external network applications	R8.5-B

There are a lot of popular UI frameworks that can be used to achieve our project objectives of providing cross-platform responsive user interface applications, described in the Table B2.2.1 above. For the purposes of this project we will consider the following 3 options in particular:

- React
- Electron
- Qt

*Table B.2.2.2 summary of design specification of the above frameworks.*

Design Option	Specification
React and Electron  	Both frameworks require browser engines. For utilization of hardware infrastructure, the frameworks require breaking applications into front-end and back-end. Separated application components require a network for data transmission as well additional software code for providing data security and users confidentiality.
QT framework 	The framework is designed for applications that run on a local machine which means that there is no need to separate back-end and front-end, no need to protect users data as it is stored within a local machine, and there is no extra browser overhead.



As described in Table B.2.2.2, a major disadvantage of running a browser application with the React or Electron framework is the need to heavily separate the front-end and back-end which means that we would have to develop two separate applications. Otherwise, we would have to run our application within a web browser environment which prohibits us from making system calls and leveraging resources for memory and concurrency in case if we require to speed up our algorithmic computations. At this point these options may be more attractive if we want to provide our solution as a web application with the backend hosted separately on a dedicated machine however that comes with additional challenges of maintaining authentication, user confidentiality and security. Introducing network delay to our solution is also disadvantageous to our performance objectives.

Therefore we will focus on the lattermost option which is to use the cross platform Qt framework application with UI and backend integrated within a local device.





### B.2.3 Stack Comparison Matrix

*Table B.2.3.1 Design requirements that are fulfilled with the stack choice*

Design Item ID	Description	Requirement ID
B.2.3.1.1 - A	Code cannot be overly complicated and must be easy to maintain	IR2-A
B.2.3.1.2 - A	Avoid “reinventing the wheel” whenever possible	IR3-A
B.2.3.1.3 - V1	Overall application should be easily packable and self-contained for distribution to the end-user	IR6-V1



Table B.2.3.2 Stack Design Options

Design Option	Specification
<p>Use Python with QT only (UI and backend)</p> 	<ul style="list-style-type: none"> <li>• Higher level language - easier to work with and more expressive</li> <li>• Easier dependency management with pip package manager</li> <li>• Better 3rd party library support for sound analytics and backend</li> <li>• Better libraries for math and image processing</li> <li>• Inferior Qt documentation and support</li> <li>• Inferior debugging capabilities of Qt objects</li> <li>• Inferior integration with Qt creator</li> <li>• Weaker control over memory management and memory-related performance</li> </ul>
<p>Use C++ with QT only</p> 	<ul style="list-style-type: none"> <li>• Superior Qt documentation and support</li> <li>• Superior debugging capabilities of Qt objects</li> <li>• Superior integration with Qt creator</li> <li>• Better memory management</li> <li>• Better optimization potential-Lower level language more error prone and less expressive, which may increase development time</li> <li>• Inferior dependency management</li> <li>• Lack of good libraries for sound analytics</li> <li>• Less portable (Qt is able to solve this problem)</li> </ul>
<p>Use both (separate processes, C++ for GUI with QT and Python for backend)</p>  	<ul style="list-style-type: none"> <li>• Superior Qt documentation and support</li> <li>• Superior debugging capabilities of Qt objects</li> <li>• Superior integration with Qt creator</li> <li>• Better 3rd party library support for sound analytics and backend using python process</li> <li>• Flexibility to use additional languages or 3rd party command-line tools as well if needed</li> <li>• IPC (inter-process communication) can be challenging to synchronize the steps (wait for step to complete) in separate processes (use shared medium for inter process communication - could be just a file on disk)</li> <li>• Packaging the entire solution can be more complicated</li> </ul>



After we have outlined specifications of design options that we have considered in the above Table B.2.3.2, we summarized the results in Figure B.2.3.3, identifying advantages and disadvantages of each option.

*Table B.2.3.3 Comparison Summary*

	Ease of development	Dependency management	3rd Party support	Doc support	Granular Debugging	Ease of Integration	Performance
Python	✓	✓	✓	✗	✗	✓	✓
C++	✗	✗	✗	✓	✓	✓	✓ <sup>++</sup>
Mixed	✓	✓	✓	✓	✓	✗	✗

In our struggles to find balance between advantages and disadvantages in regards to constraints identified in this section, we have concluded that building our solution, using the Python implementation of the QT framework, compares favorably overall to all the alternatives that we have considered.





## Appendix C - User Interface and Appearance

Note that citations within this appendix are limited to the scope of this appendix, with the corresponding references in Section C.9.

### C.1 Introduction

HappyJam’s User Interface (UI) and appearance design aims to provide simplistic, intuitive, and familiar control for musicians in the Music Transcriber program.

HappyJam’s Music Transcriber automates the recognition of pitch, timing, duration, timbre, volume, notes, cords and articulation while allowing the user to modify select parameters to maximize the effectiveness of the algorithm for his or her melody. The Music Transcriber seeks to turn music transcription into a quick and frustration-free process and aims to follow the goals described in Table C.1.

*Table C.1 The Music Transcriber UI Goals*

Goals	Description
1. Simplicity	All settings and options will be up to 3 clicks away.
2. Discoverability	The location and function of buttons, menus, and icons will be easily understood.
3. Feedback	The user will receive feedback for each program change. Any error messages will feature detailed description with error prevention measures available.
4. Efficiency	All UI will be fluent and will respond to the user quickly with no delay.

The user will interact with our Graphical User Interface (GUI), where our program will guide the user in transcribing their melody. First, the user will upload their song to the program. Our software will prioritize goals 1, 2, and 3 to ensure that the uploading process is intuitive and easy to understand. Once the uploading process is complete, our program will display the detected pitch, timing, duration, timbre, volume, notes, cords and articulation, as well as the parameters used by the algorithm. During this step our program will prioritize goal 4 to quickly and efficiently provide the user the results. Lastly, the user will change the different parameters to match the algorithm better for his or her unique melody and will export the results once complete. During this last step our program will prioritize goal 3 to provide the user with detailed insights on the transcription to allow them to make the best decision.



### C.1.1 Purpose

The purpose of this document is to provide the proper documentation and breakdown of the Music Transcriber's User interface. This document also lists the design choices made during the brainstorming process and will detail the tests and results for the user interface.

### C.1.2 Scope

This document covers the prototype user interface for HappyJam's Music Transcriber. Aside from detailing the user interface and design decisions, this document aims to follow the goals listed in Table C.1 to provide a friendly and intuitive experience for the user. Lastly, this document covers the Engineering Standards that were applied during the prototyping of the user interface.

## C.2 Audience and User Analysis

The ideal user for the Music Transcriber is a musician or an individual with musical background (formal or informal). The user is recommended to possess the ability to read and understand music sheets with the specific capacity to recognize:

- Pitch
- Timing
- Duration
- Timbre
- Volume (also known as dynamics or velocity)
- Articulation

The user must possess a Windows 10 PC with an accessible mouse and keyboard. The user must also have the ability to acquire the melody by their own means, either via a recording or download.

## C.3 User Interface Analysis

The analysis performed on the Graphic User Interface prototype will be subject to Don Norman's Design Principles. These principles will also be used throughout this text to test and prove the effectiveness and reliability of the user interface. Don Norman's Design Principles are:

- Visibility
- Feedback
- Constraints
- Mapping
- Consistency



- Affordance

### C.3.1 Visibility

Visibility allows the user to better discover where all the settings and parameters are located. Following goal 2 and 3 from Table C.1, the user will be able to easily locate all the functions and understand the current state of the program. This will be achieved via the following criteria:

- 1) The user will be guided via a wizard-style interface from stage to stage.
- 2) The Music Transcriber will contain standardized icons and layout common to other major programs for Music analysis.
- 3) All functions will be labeled and categorized in accordance to their subject.
- 4) The buttons, menus, and other intractable elements will be colored distinctly for better visibility.

### C.3.2 Feedback

Good feedback occurs when the software communicates states, errors, and results in a clear and detailed manner. To provide proper feedback and achieve goal 3 from Table C.1, the program will follow the criteria below:

- 1) The UI will display the real-time transcribed music and all the recognized elements in the melody via a labeled spectrogram
- 2) The UI will display the real-time parameters used by the algorithm
- 3) Each parameter will have a corresponding slider which will update its state in accordance to the parameter value
- 4) Each of the three stages (upload, parameter tuning, music sheet) in the wizard is clearly labeled and is contained in its own distinct tab / window.
- 5) The resulting music sheet will be updated in real-time as the user modifies the parameters.
- 6) The impact after modifying a parameter will be subsequently reflected on the spectrogram and its labels

### C.3.3 Constraints

A well designed user interface restricts the type of user interaction to prevent an invalid action and clarify what actions can be achieved to guide the user to the next step. Our program will fulfill this requirement and goal 1 and 3 from Table C.1 via the criteria below:

- 1) Using the wizard-style design, the user will be guided between the three stages (upload, parameter tuning, music sheet).
- 2) The range for each parameter will be restricted in accordance to our research



### C.3.4 Mapping

Mapping is referring to the layout of the buttons and the relationship between the controls and their effect. Our user interface will contain proper mapping by satisfying the following criteria:

- 1) Each parameter will be paired with a slider that will move in real-time and display the value of the parameter.
- 2) The layout of the software will follow the design of the most popular windows programs.
- 3) All parameters will be grouped according to their functionality and relation to signal processing or music.
- 4) The impact after modifying a parameter will be subsequently reflected on the spectrogram and its labels

### C.3.5 Consistency

A consistent user interface is one where the same actions will cause the same impact every time they are triggered. Visually, the user interface must also provide grouped controls their own distinct style to reflect to the user that they are grouped. The Music Transcriber aims to meet this requirement and goal 4 from Table C.1 by following the below criteria:

- 1) All controls will be grouped and properly divided into distinct categories
- 2) Each type of element (sliders, buttons, textboxes, etc) will follow the same color pattern and visual identity so the user will understand what each control is.
- 3) The program will utilize a set color profile that will be used throughout the different elements of the software.
- 4) The layout of the software will follow the design of the most popular windows programs.

### C.3.6 Affordance

A user interface with good affordance is one where the controls like how they are supposed to be used. The Music Transcriber aims to meet this requirement and goals 1 and 2 from Table C.1 by following the below criteria:

- 1) Each parameter will be paired with a slider that will move in real-time and display the value of the parameter.
- 2) All parameters will be grouped according to their functionality and relation to signal processing or music.
- 3) Binary decisions will be mapped to buttons.
- 4) Tabs will be used to make each of the 3 steps in the wizard distinct and separate.



## C.4 Engineering Standards

Table C.4.1 Engineering Standards

Engineering Standards	Description
<b>ISO/IEC 14496-23:2008</b>	Information technology — Coding of audio-visual objects — Part 23: Symbolic Music Representation [2]
<b>ISO/IEC 23000-12:2010</b>	Information technology — Multimedia application format (MPEG-A) — Part 12: Interactive music application format [3]
<b>ISO 10957:2009</b>	Information and documentation — International standard music number (ISMN) [4]
<b>IEC 60417</b>	Graphical Symbols for Use on Equipment [5]
<b>ISO/IEC 24752-8:2018</b>	Information technology — User interfaces — Universal remote console — Part 8: User interface resource framework [6]
<b>ISO/IEC TR 29119-11:2020</b>	Software and systems engineering — Software testing — Part 11: Guidelines on the testing of AI-based systems [7]
<b>ISO/IEC TR 24029-1:2021</b>	Artificial Intelligence (AI) — Assessment of the robustness of neural networks — Part 1: Overview [8]
<b>ISO 9241-210:2019</b>	Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems [9]
<b>ISO/IEC 11581-5:2004</b>	Information technology — User system interfaces and symbols — Icon symbols and functions — Part 5: Tool icons [10]
<b>ISO/IEC 29138-1:2018</b>	Information technology — User interface accessibility — Part 1: User accessibility needs [11]
<b>ISO/IEC 23007-2:2012</b>	Information technology — Rich media user interfaces — Part 2: Advanced user interaction (AUI) interfaces [12]



## C.5 Audience Research

The following survey was conducted to test the different scenarios that a musician might want to conduct in the program:

### C.5.1 User Profile 1

**Name:** Michael

**Occupation:** Music student at University of British Columbia

**Instruments played:** Piano and guitar

**Genre:** Mostly classic, pop or sometimes country

**Level:** Intermediate

#### How often do you find yourself needing to transcribe a melody or a song?

I often like to play freely and try new chords and music types. When I find an interesting toon I often record it and later attempt to transcribe it. Transcription alone takes many hours, so having the ability to automate the process will greatly help me produce more songs.

#### Have you tried any automated solutions such as AnthemScore or Melody Scanner?

Yes, while both AnthemScore and Melody Scanner are advertised as music transcribers, they often have trouble picking up on the correct notes between instruments. You are only able to modify the results given by AnthemScore or Melody Scanner and not the way it detects the chords, so there are a lot of corrections that must be made. If I had the ability to modify some parameters to detect different sections better, that would be greatly welcomed.

The following questions were given to Michael to gain a better understanding of the type of User Interface that will most appeal to the user:

*Table C.5.1 Michael's Q&A*

Questions	Answers
How would you prefer to go about uploading a song to the program?	Ideally, I would simply start the program and be presented with either a main screen where there is only an uploading button, or a blank project screen where I will have to start a new project via the file menu.



Would you prefer to have the ability to pre-set the types of instruments used in the song you are uploading?	Yes, although I would also like to upload songs that I have not played myself and have the program automatically detect the instruments on my behalf.
Would you be interested in the ability to manually tune parameters in the program to better transcribe the music?	Yes, that would be a major benefit to be since it will a lot of time in trying to correct wrong transcriptions
How many instruments do you most likely have in all your songs?	I like to transcribe my own music, which consists of either just piano or guitar, or sometimes a piano and guitar. I do occasional collaborations which may sometimes include a saxophone.
Do you have good recording equipment?	Yes, since this is a big hobby of mine that I hope to pursue professionally in the future, I have a studio microphone along with an amp.

### C.5.2 User Profile 2

**Name:** Jane

**Occupation:** Arts student at Simon Fraser University

**Instruments played:** Violin and guitar

**Genre:** Classical and pop

**Level:** Intermediate

#### How often do you find yourself needing to transcribe a melody or a song?

I really like to experiment and learn new songs on both my guitar and violin. Whenever I am not learning a song I spend time just trying to make my own melody. If I find myself stroken by inspiration, I sit down and try to make my own song, which involves many hours of experimentation and going back and forth from the paper to the instrument. Thus, I would probably say a few times a month.

#### Have you tried any automated solutions such as AnthemScore or Melody Scanner?

I have heard of them, but they look very complicated and overwhelming to use so I often stick to simple pen and pencil. I know they may make the transcribing process quicker, but I am not very technology literate so I am often uncomfortable with large user interfaces with a lot of buttons that are presented without an explanation.



The following questions were given to Jane to gain a better understanding of the type of User Interface that will most appeal to the user:

Table C.5.2 Jane's Q&A

Questions	Answers
How would you prefer to go about uploading a song to the program?	I would like the process to be as simple as possible. If there was a window that simply presented an upload button, that would be perfect.
Would you prefer to have the ability to pre-set the types of instruments used in the song you are uploading?	I understand that it may be difficult to detect the instruments, so I would say yes just in case the program may have a hard time doing it itself.
Would you be interested in the ability to manually tune parameters in the program to better transcribe the music?	Yes, but if I am presented with such options they must be placed in a simplistic and not overwhelming manner. Options must be grouped or explained as clearly as possible and labeled accordingly.
How many instruments do you most likely have in all your songs?	Usually only one instrument, either my violin or guitar.
Do you have good recording equipment?	Yes, I have the Blue Yeti microphone which is one of the best USB microphones on the market.

### C.5.3 User Profile 3

**Name:** Alex

**Occupation:** Piano tutor

**Instruments played:** Piano, Saxophone, violine, guitar

**Genre:** Classical

**Level:** Professional

#### How often do you find yourself needing to transcribe a melody or a song?

As an intermediate musician, I transcribe music quite often. I often attempt to create my own score for the purpose of practice and self-promotion. My process often involved playing a section of my score and going back and forth between the paper and instrument to get the most accurate translation to paper. This often takes me weeks to a month, depending on how much I like my results at the time.





### Have you tried any automated solutions such as AnthemScore or Melody Scanner?

No, I very much believe that paper and pen is the best solution for transcription. It may be tedious, but this way I am able to add all the details required. However, I am open to the possibility of trying one such automation program as my scores become increasingly long.

The following questions were given to Alex to gain a better understanding of the type of User Interface that will most appeal to the user:

Table C.5.3 Alex's Q&A

Questions	Answers
How would you prefer to go about uploading a song to the program?	I would like to have the ability to upload both compressed and uncompressed recordings.
Would you prefer to have the ability to pre-set the types of instruments used in the song you are uploading?	Yes, in the future my melodies may include a large number of instruments so I want the ability to add each one manually to ensure that the transcription works properly.
Would you be interested in the ability to manually tune parameters in the program to better transcribe the music?	Yes. There are many scenarios where I want the sound to come out exactly as I want, so I don't want to battle an automated program and instead give it the exact parameters I want.
How many instruments do you most likely have in all your songs?	As of right now, I may have anywhere from a single instrument to a handful.
Do you have good recording equipment?	Yes, I invested a great deal of money on good-sounding and high quality equipment.

### C.6 Testing

The following criteria was created to test the user interface and ensure that all requirements were met:



Table C.6.1 UI Testing Plan

Requirement	Description
R3.1-A	The SMT must be able to analyze an audio sample to record a list of notes and their properties
Criterion:	The program can output text representing the note
R3.1.1-A	The time of a detected note must be recorded
Criterion:	The output note list includes the offset into the audio sample, in either seconds or samples, at which every note begins
R3.1.2-A	The pitch of a detected note must be recorded
Criterion:	The output note list includes the pitch name, including the octave number, of every note in the list
R4.1-A	The software must be able to open a window on a desktop PC running Windows
Criterion:	Running the executable opens a window.
R4.1.1-A	The user must be able to stop the software and close the window by pressing the window's X button.
Criterion:	After opening the window, press the X button and ensure the window closes. Check Task Manager to ensure the process is no longer running.
R4.2-A	The software must have a submenu to select an audio file to import
Criterion:	Clicking the audio import option in the submenu opens a file explorer window.
R4.2.1-A	The import feature must take uncompressed audio files in .wav format
Criterion:	Opening the audio import file explorer and selecting a .wav file correctly opens and displays the audio data.
R4.4.5-A	The user must be able to zoom in and out of the audio view formats
Criterion:	After importing an audio sample, turning the scroll wheel stretches or squashes the audio waveform display.
R4.10-A	The UI must display in dark mode
Criterion:	On opening the UI window, the user's eyes are soothed with



	dark-grey colours.
--	--------------------

### C.7 Graphical Representation

The purpose of the Music Transcriber's user interface is to present the user with a simple, intuitive, and transparent program to take in any melody containing three or less instruments, and transcribe it to the musical sheet. The program will guide the user to choose the best settings for transcription and will allow the user to modify parameters accordingly.

The following are some of the screenshots from the prototype currently being developed:



Figure C.7.1 Spectrogram view functionality

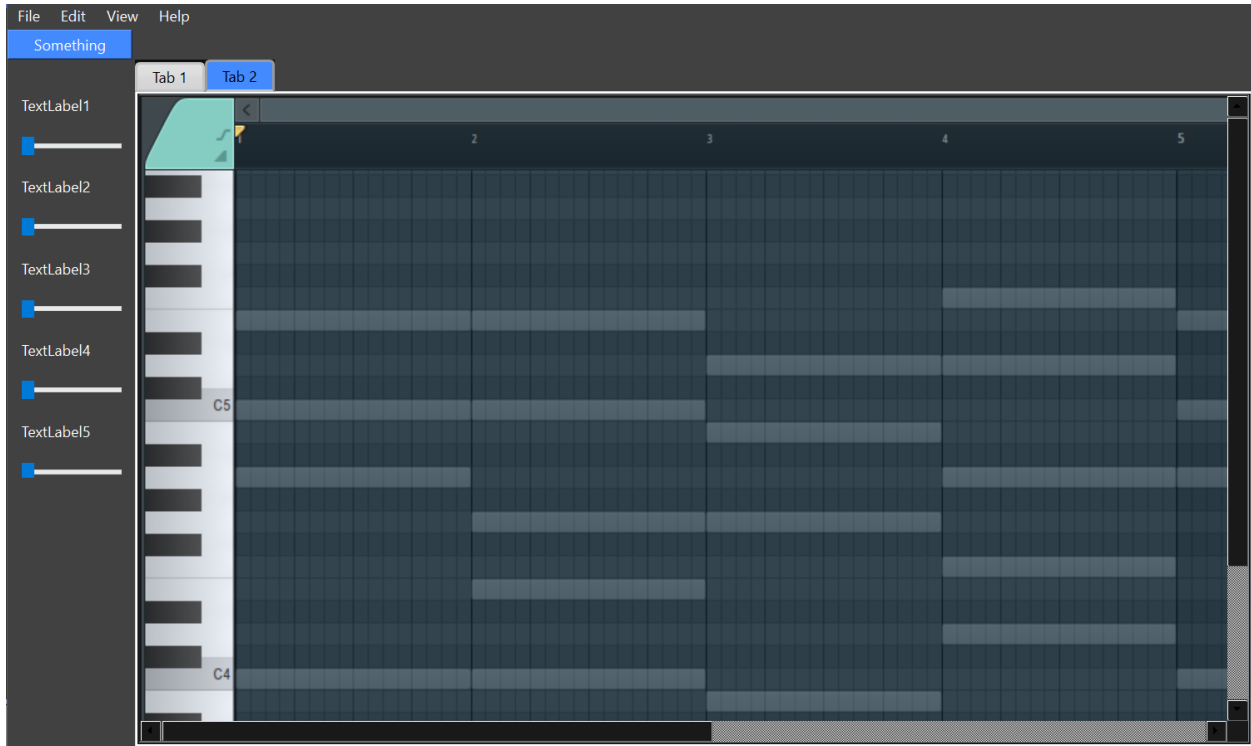


Figure C.7.2 Piano view



Figure C.7.3 Musical Sheet view



## **C.8 Conclusion**

The user interface of HappyJam's Music Transcriber aims to deliver the user a simple, intuitive, and transparent program to take in any melody containing three or less instruments and transcribe it to the musical sheet. HappyJam's Music Transcriber automates the recognition of pitch, timing, duration, timbre, volume, notes, cords and articulation while allowing the user to modify select parameters to maximize the effectiveness of the algorithm for his or her melody.

The user will interact with our Graphical User Interface (GUI), where our program will guide the user in transcribing their melody. First, the user will upload their song to the program. Our software will prioritize goals 1, 2, and 3 to ensure that the uploading process is intuitive and easy to understand. Once the uploading process is complete, our program will display the detected pitch, timing, duration, timbre, volume, notes, cords and articulation, as well as the parameters used by the algorithm. During this step our program will prioritize goal 4 to quickly and efficiently provide the user the results. Lastly, the user will change the different parameters to match the algorithm better for his or her unique melody and will export the results once complete. During this last step our program will prioritize goal 3 to provide the user with detailed insights on the transcription to allow them to make the best decision.

Currently, further research is required in developing a friendly user interface that will be intuitive and functional for intermediate and professional level musicians. For our first prototype, we aim to map all backend functionality to the interface and work together to simplify and categorize different options.



## C.9 References

- [1] S. Rekhi, “Don Norman's Principles of Interaction Design,” *Medium*, 25-Feb-2018. [Online]. Available: <https://medium.com/@sachinrekhi/don-normans-principles-of-interaction-design-51025a2c0f33>.
- [2] “Information technology — Coding of audio-visual objects — Part 23: Symbolic Music Representation,” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso-iec:14496:-23:ed-1:v1:en>.
- [3] “Information technology — Multimedia application format (MPEG-A) — Part 12: Interactive music application format,” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso-iec:23000:-12:ed-1:v1:en>.
- [4] “Information and documentation — International standard music number (ISMN),” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso:10957:ed-2:v1:en>.
- [5] “Music,” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iec:grs:60417:5085>.
- [6] “Information technology — User interfaces — Universal remote console — Part 8: User interface resource framework,” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso-iec:24752:-8:ed-1:v1:en>.
- [7] “Software and systems engineering — Software testing — Part 11: Guidelines on the testing of AI-based systems,” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso-iec:tr:29119:-11:ed-1:v1:en>.
- [8] “Artificial Intelligence (AI) — Assessment of the robustness of neural networks — Part 1: Overview,” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso-iec:tr:24029:-1:ed-1:v1:en>.
- [9] “Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems,” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-2:v1:en>.
- [10] “Information technology — User system interfaces and symbols — Icon symbols and functions — Part 5: Tool icons,” *ISO*. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso-iec:11581:-5:ed-1:v1:en>.



[11] “Information technology — User interface accessibility — Part 1: User accessibility needs,” *ISO*. [Online]. Available:  
<https://www.iso.org/obp/ui/#iso:std:iso-iec:29138:-1:ed-1:v1:en>.

[12] “Information technology — Rich media user interfaces — Part 2: Advanced user interaction (AUI) interfaces,” *ISO*. [Online]. Available:  
<https://www.iso.org/obp/ui/#iso:std:iso-iec:23007:-2:ed-1:v1:en>.