

Cadenza – Automatic Music Transcription

Joel Rajabally

H446—04B

WBS 28384

A Level Computer Science

Project 2024

Table of Contents

[3.1.1a Describe and justify the features that make the problem solvable by computational methods 1](#_Toc433849958)

[3.1.1b Explain why the problem is amenable to a computational approach 1](#_Toc944993450)

[3.1.2a Identify and describe those who will have an interest in the solution explaining how the solution is appropriate to their needs (this may be named individuals, groups or persona that describes the target end user) 2](#_Toc704086922)

[3.1.3a Research the problem and solutions to similar problems to identify and justify suitable approaches to a solution 2](#_Toc1651834918)

[3.1.3b Describe the essential features of a computational solution explaining these choices 2](#_Toc1628708725)

[3.1.3c Explain the limitations of the proposed solution 2](#_Toc857750953)

[3.1.4a Specify and justify the solution requirements, including hardware and software configuration (if appropriate) 2](#_Toc410699652)

[3.1.4b Identify and justify measurable success criteria for the proposed solution 2](#_Toc1354453696)

[3.2.1a Break down the problem into smaller parts suitable for computational solutions justifying any decisions made 2](#_Toc274024261)

[3.2.2a Explain and justify the structure of the solution 2](#_Toc2133052025)

[3.2.2b Describe the parts of the solution using algorithms justifying how these algorithms form a complete solution to the problem 2](#_Toc792954888)

[3.2.2c Describe usability features to be included in the solution 2](#_Toc1141153114)

[3.2.2d Identify key variables / data structures / classes justifying choices and any necessary validation 2](#_Toc1192987518)

[3.2.3a Identify the test data to be used during the iterative development and post development phases and justify the choice of this test data 2](#_Toc58579335)

[3.3.1a Provide annotated evidence of each stage of the iterative development process justifying any decision made 2](#_Toc1582812059)

[3.3.1b Provide annotated evidence of prototype solutions justifying any decision made 2](#_Toc2102465381)

[1 – Drag and Drop Functionality 2](#_Toc416163468)

[2 – Fourier Transform 3](#_Toc1616935840)

[3 – Login GUI 4](#_Toc51125314)

[4 – File Backend 5](#_Toc450682382)

[5 – File system GUI 6](#_Toc456985911)

[6 – Find Note Function 7](#_Toc1212864249)

[7 – Overall Notes Graphic 8](#_Toc1226018121)

[8 – Hashing Password 9](#_Toc2039144353)

[9 – Lilypond File Execution 10](#_Toc2143303715)

[10 – PDF Construction 11](#_Toc1586179719)

[11 – Playback 12](#_Toc1702741576)

[12 – Recording 13](#_Toc975634738)

[13 – Login Backend 14](#_Toc996788463)

[14 – Notes Class 15](#_Toc156880136)

[15 – Load and Save 16](#_Toc315336288)

[3.3.2a Provide annotated evidence for testing at each stage justifying the reason for the test 17](#_Toc89044919)

[3.3.2b Provide annotated evidence of any remedial actions taken justifying the decision made 17](#_Toc1989260267)

[3.4.1a Provide annotated evidence of testing the solution of robustness at the end of the development process 17](#_Toc1948085832)

[3.4.1b Provide annotated evidence of usability testing (user feedback) 17](#_Toc95023089)

[3.4.2a Use the test evidence from the development and post development process to evaluate the solution against the success criteria from the analysis 17](#_Toc1100445197)

[3.4.3a Provide annotated evidence of the usability features from the design, commenting on their effectiveness 17](#_Toc515083184)

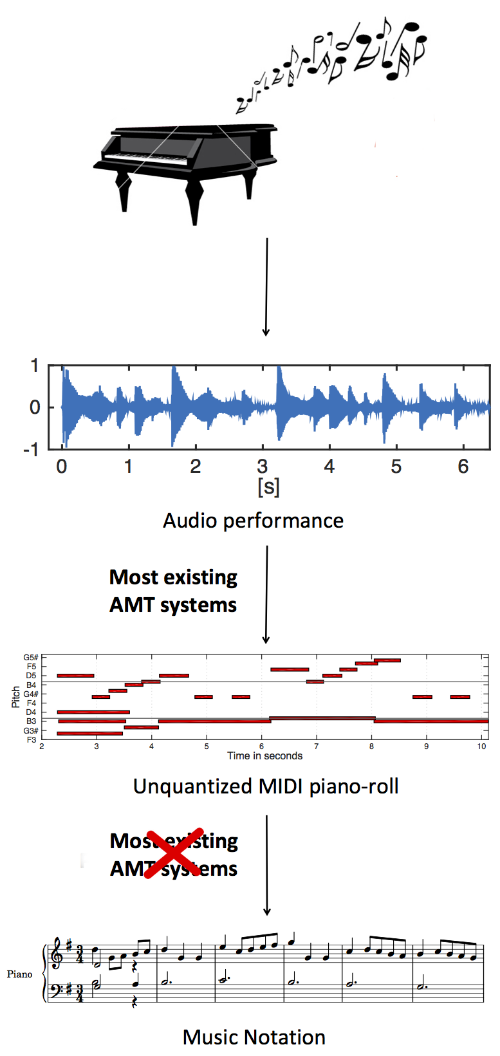
[3.4.4a Discuss the maintainability of the solution 17](#_Toc1242102693)

[3.4.4b Discuss potential further development of the solution 17](#_Toc71477726)

# 3.1.1a Describe and justify the features that make the problem solvable by computational methods



In modern Western music, the primary form of communication has been a standardised staff notation since it was created in the in the 10th century by an Italian music theorist. Traditionally the process of transcribing (writing down musical ideas into this notation) has been done manually by composers, which is inefficient and limiting in several ways.

It would be helpful a piece of software that can automate the tedious process of manual transcription, preventing the loss of ideas that can occur during writing music down (inexperienced composers may not be able to keep ideas in their head while transcribing). The increase in modern computer speed allows the complex signal processing required to process audio data to be done without. Currently, there are limited tools available to transcribe audio to sheet music without bouncing between different programs, as programs often split up the task by using .xml files as an intermediary step. These solutions are also often tricky to use, overflowing with features that can confuse composers and overcomplicate the process, which should be as simple as possible to allow a greater focus on the creative process, rather than the writing up process. The length of this process means that quick retakes and new ideas are discouraged; essentially mandating only one take further impairs creative freedom. My project could therefore create a friendlier and more intuitive user experience.

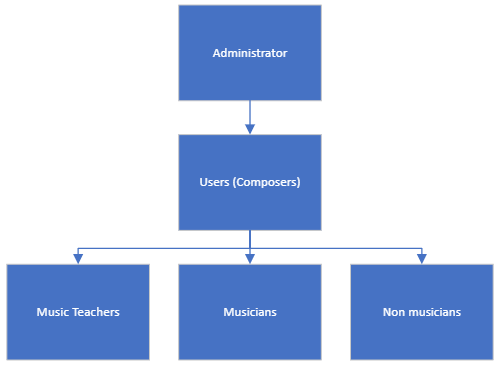
https://labsites.rochester.edu/air/projects/AMT.html

As music moves further into a communal space, as sharing creations becomes easier, a lack of formal training should not become a barrier to joining the community of composers. Music should be a unifying tool, and not being able to share music with others because you can’t write it can prevent many amazing pieces of music from being shared with the world. Even if musicians can play an instrument, they may feel that to transcribe their music easily, they need to learn piano to make use of MIDI recording capability. MIDI is a protocol to allow musical information to be recorded, but it only works with electric instruments. My solution should therefore be able to be used with acoustic instruments as well.

# 3.1.1b Explain why the problem is amenable to a computational approach

|  |  |
| --- | --- |
| Computational Method | Explanation |
| Interfacing | Lilypond is a program which can turn a specific text format into PDF sheet music. My Python code can communicate with Lilypond via the modules subprocess and lilypond. This allows for the solution to be carried out with minimum client intervention. |
| OOP, Inheritance & Polymorphism  Pitch: E5  Duration: 1 | The similar qualities that all music notes share make them easily applicable to being objects, having parameters like pitch and duration. Inheritance and polymorphism can be used specifically in the design of a “Rest” class, which would interact with the program in the same way that normal notes would, but would be turned into text using a different process. |
| Database | The one-to-many relationship of Users to Pieces and Pieces to Takes means that a relational database can be easily employed to hold the data files. This allows information to be stored in an easy to access and parse way, with each take being held under specific categories to keep them separated. |
| Signal Processing | Mathematical manipulation of data to extract musical features is a powerful tool to recognise music in audio. In particular, computational audio analysis allows for acoustic instruments to be used as well electric ones.  <https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft> |
| GUI | A good GUI allows the music to be edited post-recording in an intuitive way as not to undermine the simplicity of the program to use. Dynamic graphics based on current musical data allows users to quickly find and correct mistakes in their recording before it is processed (or indeed after initial processing). |

# 3.1.2a Identify and describe those who will have an interest in the solution explaining how the solution is appropriate to their needs (this may be named individuals, groups or persona that describes the target end user)



|  |  |  |
| --- | --- | --- |
| User | Requirement | Justification |
| Administrator | Will be able to access and edit all parts of source code, as well as oversee the database | Highest level access is required to debug the program post-production and deal with any unfound errors |
| Composers | Will be able to access all client-side parts of the program, including recording and editing features in order to transcribe efficiently | Composers are the primary clients, and such all features should be initially designed around their needs. They will want to be able to make use of all the automatic transcription tools, for example in the writing down of an improvised solo |
| Music Teachers | Should be able to view all the data of their students and add comments privately | Teachers' role in student composition is mainly analytical and they can use the software to give students feedback – for example during a GCSE composition task |
| Musicians | Will be able to view all public data and add public comments | Other musicians may want to make use of current transcriptions – for example when looking for transcriptions of famous pieces |
| Non musicians | Will only require basic level interaction (e.g. view names of files and users) | Non-musicians only need to view current file names and other metadata – for example a parent may want to see how their children are getting on in music |

# 3.1.3a Research the problem and solutions to similar problems to identify and justify suitable approaches to a solution

# 3.1.3b Describe the essential features of a computational solution explaining these choices

# 3.1.3c Explain the limitations of the proposed solution

# 3.1.4a Specify and justify the solution requirements, including hardware and software configuration (if appropriate)

# 3.1.4b Identify and justify measurable success criteria for the proposed solution

# 3.2.1a Break down the problem into smaller parts suitable for computational solutions justifying any decisions made

# 3.2.2a Explain and justify the structure of the solution

# 3.2.2b Describe the parts of the solution using algorithms justifying how these algorithms form a complete solution to the problem

# 3.2.2c Describe usability features to be included in the solution

# 3.2.2d Identify key variables / data structures / classes justifying choices and any necessary validation

# 3.2.3a Identify the test data to be used during the iterative development and post development phases and justify the choice of this test data

# 3.3.1a Provide annotated evidence of each stage of the iterative development process justifying any decision made

# 3.3.1b Provide annotated evidence of prototype solutions justifying any decision made

## 1 – Drag and Drop Functionality

|  |
| --- |
| Prototype 1 – Drag and Drop Functionality |
| To test the use of drag-drop in the music notes GUI. This would be the ideal way for the user to interact with the software, as it matches other MIDI type music editors    Side effect of being able to resize vertically  Desired effect of extending box  Original window  Invoking the dndr commands  Creating the window using tkinter  Importing the necessary modules  The problem with using this is that there does not seem to be a way to prevent both dimensions from being resizable, meaning that I can’t use this module in my project. Moving forwards, the box may need to be resized using text inputs embedded in the GUI. |

## 2 – Fourier Transform

|  |
| --- |
| Prototype 2 – Fourier Transform |
| To test the built in Fourier transform functions that come with the SciPy module. This is to transform the audio signal (mapped over time) into its constituent frequencies (represented as magnitude spikes on the graph)    Plotting the results  Computing the Fourier transform  Setting constants  Importing modules, as well as matplotlib to check the results  The plot output    This is using the SciPy documentation on using its FFT capability. Moving forward, I need to take a deeper look into the maths behind FFT and DFT to further refine its application into my project. |

## 3 – Login GUI

|  |
| --- |
| Prototype 3 – Login GUI |
| A framework for how the login GUI will look using Tkinter    The window created  Setting up the window  Setting up all the widgets  Functions to be able to show and hide the password box  Importing modules  This was helpful as it allowed me to get more  comfortable with using the different placement  methods, using pack for the main window but grid  inside the canvases. |

## 4 – File Backend

|  |
| --- |
| Prototype 4 – File Backend |
| A test using sqlite3 to access a database and extract the available files    Output to the command line  Simple command line choice  Accessing the table Takes  Accessing the table Projects  Importing modules    Here I learned to use relational database tables, using foreign keys to allow a one to many relationship between Users and Projects, and Projects and Takes. |

## 5 – File system GUI

|  |
| --- |
| Prototype 5 – File system GUI |
| A framework for how the file system GUI will look using Tkinter    Function that takes input list and outputs choice to a function  Calling the test  Creating the window  Defining a test folder structure    The GUI created  Here I learned to use the Listbox Tkinter widget for a user-friendly choice GUI. |

## 6 – Find Note Function

|  |
| --- |
| Prototype 6 – Find Note Function |
| A function to return a note name given a frequency in Hertz    Test outputs  The maths behind the frequency to pitch conversion  The maths behind the frequency to pitch conversion  Importing NumPy for the logarithms    Taken from a tutorial on chicken.com on guitar tuners. Demonstrates the logarithmic relationship behind equal temperament tuning |

## 7 – Overall Notes Graphic

|  |
| --- |
| Prototype 7 – Overall Notes Graphic |
| A framework for how the note adjustment GUI will look using Tkinter    Initial notes drawn on as well as an extra added via command line  Allowing notes to be added via command line  Setting up the window with initial test notes  Function to add a note to the right coordinates  Populating the canvas with the background graphic  Dictionaries to decide on coordinates and colours  I learned to use the dictionaries as references for efficient drawing, using coordinates |

## 8 – Hashing Password

|  |
| --- |
| Prototype 8 – Hashing Password |
| A test of bcrypt to encrypt passwords to provide extra security to the program    Checking guess against password  Creating password hash  Incorrect guess case    Correct guess case    This module seems to be easy to interface with, moving forwards I need to check if storing the hashed password in a database then fetching it on a later run of the program causes any problems. |

## 9 – Lilypond File Execution

|  |
| --- |
| Prototype 9 – Lilypond File Execution |
| A test of the lilypond module for interacting with the lilypond program for creating the visuals of the sheet music from a specific text format    Writing text above to a file and executing the program  Hardcoded text “piece” I wrote as the initial test  Importing modules  The PDF created from the above code    Using the Lilypond program required a steep learning curve to understand the text format syntax, however it should be reproducible via code now I understand how it is structured. The advantage to using this is the PDF creation can be abstracted, the disadvantage being that an external executable is required for the program to work. |

## 10 – PDF Construction

|  |
| --- |
| Prototype 10 – PDF Construction |
| A test of reportlab to check the feasibility of creating PDFs from scratch    Experimentation and then use of canvas drawing features  Creating the PDF file  Importing modules  The PDF created (by lines 10-13)    This would be the ideal way to create the PDFs, completely in house, but as a method it would require a lot of additional time to create, due to problems such as the drawImage method not working easily using small images. In the future, I will have to weigh up whether it’s worth it to design the PDFs manually. |

## 11 – Playback

|  |
| --- |
| Prototype 11 – Playback |
| A test of sounddevice and soundfile to read wav files and play sound    Reading and then playing the WAV file  Importing modules  These modules seem fairly easy to use in my program, I will look more into what dtype means, and whether it should be changed to fit the recording quality I need |

## 12 – Recording

|  |
| --- |
| Prototype 12 – Recording |
| A test of sounddevice to record audio and SciPy to save the audio as a WAV file    Beginning recording and waiting until end  Setting constants  Importing modules  Here I used the rec method to record the audio, which works well for simple recording features, however I may need to look into the stream method if I want to process the audio live (during its recording) |

## 13 – Login Backend

|  |
| --- |
| Prototype 13 – Login Backend |
| A test of sounddevice to record audio and SciPy to save the audio as a WAV file |

## 14 – Notes Class

|  |
| --- |
| Prototype 14 – Notes Class |
| A test of sounddevice to record audio and SciPy to save the audio as a WAV file |

## 15 – Load and Save

|  |
| --- |
| Prototype 15 – Saving and Loading |
| A test of sounddevice to record audio and SciPy to save the audio as a WAV file |

# 3.3.2a Provide annotated evidence for testing at each stage justifying the reason for the test

# 3.3.2b Provide annotated evidence of any remedial actions taken justifying the decision made

# 3.4.1a Provide annotated evidence of testing the solution of robustness at the end of the development process

# 3.4.1b Provide annotated evidence of usability testing (user feedback)

# 3.4.2a Use the test evidence from the development and post development process to evaluate the solution against the success criteria from the analysis

# 3.4.3a Provide annotated evidence of the usability features from the design, commenting on their effectiveness

# 3.4.4a Discuss the maintainability of the solution

# 3.4.4b Discuss potential further development of the solution