UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Farida Yusuf
May 11, 2021

Recreating the Hammond organ through virtual additive synthesis and automatic synthesiser programming

Supervisor: Dr. Jize Yan
Examiner: Dr. Terrence Mak

A project report submitted for the award of
MEng Electronic Engineering with Artificial Intelligence

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of
MEng Electronic Engineering with Artificial Intelligence

by Farida Yusuf

Automatic synthesiser programming is the application of machine learning techniques to program a given synthesiser to reproduce a sound sampled in digital audio format. To date, research & literature within this field has documented success in estimating the parameters of sounds originating from the given synthesiser, but there is little address of approximating sounds which are foreign to the synthesiser, such as sounds from real instruments and nature. The former has been referred to as the intra-domain problem, and the latter the cross-domain problem.

This project founds research into the cross-domain problem, by attempting to resynthesise the recorded sounds of a Hammond organ through automatic synthesiser programming applied to a basic virtual emulation of the Hammond. Technical and perceptual evaluations are presented along with results, demonstrating success in partial optimisation.

Keywords: machine learning, sound synthesis, audio signal processing, automatic synthesiser programming, parameter estimation

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN Convolutional Neural Network
DSP Digital Signal Processing
LSTM Long Short-Term Memory
MFCC Mel-Frequency Cepstral Coefficient
PLP Perceptual Linear Prediction
STFT Short-Time Fourier Transform
VST Virtual Studio Technology

Acknowledgements

Thanks to Matthew John Yee-King, a prolific academic at Goldsmiths, University of London, and Valerio Velardo, a prolific YouTube content creator and founder of The Sound of AI, without whom this project would never have earnt my confidence.

Further thanks to Olivier Bélanger, author of the indispensable pyo library, as well as Lenore Byron and Andrew Sharp, two social media strangers who suggested improvements to my digital mimicry of the Hammond organ.

A personal thank you also to my project supervisor, Dr. Jize Yan, for his regular calls to monitor my progress, especially the ones I received asleep.

To Zahira, my little sister. You can be anything you want to be. Also to Aunty Bea, for who I have become.

Chapter 1

Introduction

1.1 Sound synthesis

Sound synthesis is the process of electronically generating sounds and timbres for use in music production. These can recreate acoustically produced sounds which can be heard from instruments, objects or landscapes in real life, or create completely new, artificial sounds & effects from the imagination and beyond. There are many methods of synthesis, such as subtractive, granular, wavetable, FM and so on, but the principle remains the same across all methods: waves in the audible frequency range are manipulated in the time and/or frequency domains to achieve a particular sound, of finite or infinite duration.

(a) The Minimoog (b) Primer (plugin)

FIGURE 1.1: Synthesisers, hardware (left) and virtual/software (right)

A (sound) synthesiser is a system, often referred to as a musical instrument, that generates sound electronically. This can be implemented in hardware as an electronic system or simulated in software as a digital program. The interface of a

synthesiser is conventionally comprised of various controls (such as knobs, sliders and switches) to set parameters related to filtering, modulation and amplification of an audio signal. This audio signal is generally sourced from one or more audible-frequency oscillators in the system.

The number of variable parameters among synthesisers can range from around a dozen to several hundreds, increasing the complexity of programming sounds exponentially. Sound synthesis by manual tuning of parameters is hence recognised as a complicated task, barely overcome by the professional dedication of sound engineers. This has inspired the research field of **automatic synthesiser programming**, revisited in 1.3.

1.2 Additive synthesis

1.2.1 Overview

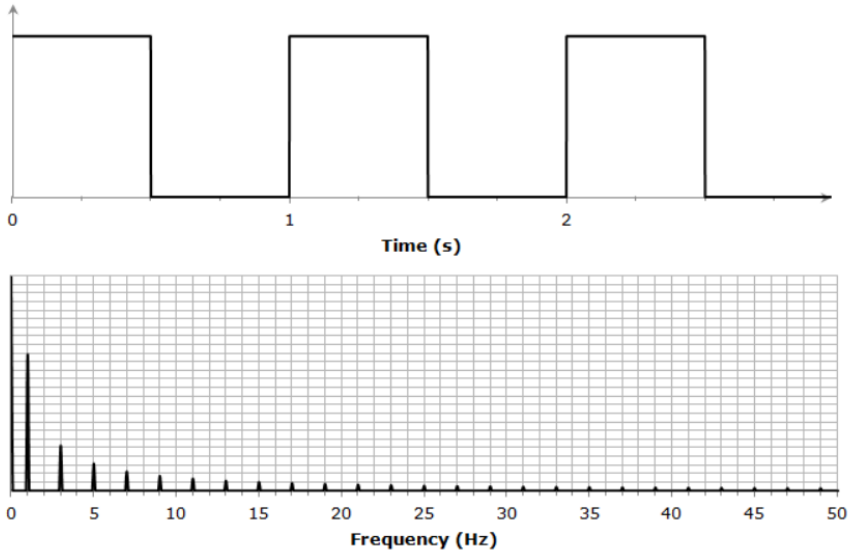


FIGURE 1.2: Time-domain waveform (top) and Fourier spectrum (bottom) of a 1Hz square wave, demonstrating additive synthesis [32]

Additive synthesis is a specific method of sound synthesis, in which pure tones are combined to form a more complex sound. The resulting sound spectrum can be further moulded through the application of filtering and amplification elements, as well as the modulation of these elements. Given a complex signal path of sources (oscillators, which produce tones) and processors (filters, amplifiers, etc.) such

as in a modular synthesis environment, it is evidently possible to produce many interesting and dynamic sounds through this method.

Physically, a pure tone is a sine wave of a given frequency. Hence additive synthesis offers the creation of unique sounds through different combinations and transformations of sinusoids. This may be known more generally as Fourier synthesis.

1.2.2 The Hammond organ

Recognised as a pioneering instrument of various music genres [4] during the 20th century—notably gospel, jazz, blues and funk—the Hammond B3 organ is one of the most well-known instances of a **tonewheel organ**. (This particular model is often referred to simply as "the Hammond organ" given its legacy as the most popular and successful iteration of the product.)

A tonewheel organ is an electromechanical instrument which generates sound through a set of tonewheels, each rotated near an electromagnetic pickup; the resulting alternating current is amplified and used to drive a speaker. Conventionally, each tonewheel produces a sine wave (tone) in the audible frequency range. With reference to a fundamental frequency produced by one of the tonewheels, other tonewheels each produced a harmonic of this frequency, e.g. a major third or an octave above. (Of course, the fundamental frequency is specific to a given note on the organ.)

Notably, the amplitude of each individual tone could be set in discrete values—on the Hammond organs, this was memorably by means of a drawbar mechanism—allowing players to construct unique sound configurations referred to as "registrations" or simply drawbar settings. This is easily recognised as additive synthesis, inferring that the tonewheel organ is an additive synthesiser. (Later models of the Hammond continued the basis of additive synthesis but through upgraded means relative to tonewheels, i.e. integrated circuits.)

The Hammond organ further extended its sound capabilities through two notable effects: the vibrato scanner and the Leslie speaker. The former is an on-board processing effect offering multiple chorus- and vibrato-inspired augmentations to the sound. The latter is a partially enclosed, rotary add-on loudspeaker, which produces complex modifications to its sound output through reverberation and the Doppler effect. Although both were optional effects and could be bypassed, they

FIGURE 1.3: The interface of the Hammond B3 organ. The drawbars can be seen across the top.

are widely recognised as part of the Hammond's sonic legacy and many digital recreations of these effects have been produced as commercial audio software.

1.3 Automatic synthesiser programming

1.3.1 Overview

Automatic synthesiser programming is the use of machine learning techniques to automatically program a synthesiser. This is usually for "sound matching" purposes, where a sound is presented as a digital audio sample to the programmer which must then estimate the parameter settings on a given synthesiser (or more generally, sound engine) to reproduce this sound as closely as possible.

Previous work in this field [43] [41] [3] has demonstrated sound matching through VST synthesisers; VST (Virtual Studio Technology) is a standard plugin format for commercially available software synthesisers. Notably in these experiments, the samples were generated from the same synthesiser used for resynthesis, referred to in [3] as the *"intra-domain problem"* of automatic synthesiser programming. The *"cross-domain problem,"* wherein reproducing foreign sounds on a given synthesiser is often addressed as "left for future investigation" in the current state of research within the field. This leads us to the objectives of this project.

1.3.2 Project objectives

This project seeks to break ground into the *cross-domain problem* by investigating the resynthesis of recorded samples from a Hammond organ on a virtual additive synthesiser, custom-built using a digital signal processing (DSP) toolbox. The signal path and properties of the virtual synth will be closely informed by the architecture of the Hammond organ, and the necessary attributes of its oscillators, filters, processing modules, etc. will be initialised as parameters to be estimated by an automatic synthesiser programmer.

It is thus the aim of this project to produce resynthesised sounds with demonstrable similarity to the original samples, shedding light onto prospects within the cross-domain problem of automatic synthesiser programming.

Chapter 2

Background & Literature

2.1 Introduction

A considerable body of work exists on the problem of automatic synthesiser programming. The main aspects addressed are the choice of audio features, learning algorithms and evaluation techniques. The concept of frameworks is less discussed but not unseen, with particular reference to previous work from Yee-King et al. [43]

2.2 Audio features & extraction

2.2.1 Introduction

One of the primary questions of implementation is how an audio sample can be represented to the learning model/algorithm. Characteristics that can be extracted from and used to represent an audio signal in machine learning applications are known as **audio features**. [24]

There are almost as many ways to classify audio features as there are features themselves. A useful technical classification is by domain: a given audio feature presents data in the time, frequency or time-frequency domain. Table 2.1 gives examples of audio features in each of these categories.

The appropriate choice of audio feature(s) depends on the intended application. For instance, spectral (frequency-domain) features are often employed for speech

Domain	Audio Features
Time	amplitude, zero-crossing rate, energy
Frequency	spectral centroid, spectral density, MFCCs, PLPs
Time-Frequency	spectograms, log-spectograms

TABLE 2.1: Examples of (low-level) audio features, classified by domain

recognition and classification. [29] [19] However, this is still a broad pool, which can be further narrowed by considering cross-categorisations and/or invariances among features to leverage. One of the simpler reasons MFCCs have proven popular in models for speech recognition [10] is that they are pitch-invariant. [25]

Extracting temporal (time-domain) features is straightforward compared to spectral features: the former are computed directly from the waveform, whereas a signal must first be mapped into the frequency domain using the Fourier transform to obtain spectral features. However, spectral features have the advantage in revealing global (as opposed to local) signal properties, hence their prevalence in applications involving timbral analysis [44] and in the general literature. [24] [2] Spectro-temporal features combine the advantages of spectral and temporal features and can outperform either in detection and classification problems [34] [1] but are the most complex to utilise. [40]

2.2.2 Application to sound synthesis

For our sound matching purposes, we are most interested in those features that present vivid information about a sound's timbre. Timbre is ill-defined in psychoacoustic theory [36] but is understood to collectively describe the properties that differentiate our perception of a note on, say, the guitar from the same note on a piano. This is independent of the note's pitch, intensity, duration and spatial position.

There are several options for audio features in this regard. [30] The use of MFCCs is popular in recent work on sound matching [43] [41] and musical analysis of signals in general. [11] Spectograms and log-spectograms are also seen in both, particularly paired with convolutional neural networks for learning, where techniques for 2D data widely applied to image processing are leveraged on spectograms for audio. [3] [7]

For a given audio sample, values for MFCCs and spectral features in general are obtained over several slices of the sample, called frames. Each frame aggregates

statistics over a very short duration of time, typically within 10–50 ms. (The signal exhibits virtually stationary behaviour on this timescale.) This allows insight into some time-dependent properties of signals as well; an example is amplitude, a well-established parameter in sound synthesis.

2.3 Learning models/algorithms

2.3.1 Overview

What we seek in a learning model/algorithm for our purposes is "a general heuristic to match synthesis parameters of a fixed sound engine to an arbitrary sound target," as summarised by Cáceres. [9] There are, of course, myriad machine learning methods in existence and in determining which ones are of interest, it is useful to summarise the pool. Hence potential solutions for sound matching can be grouped into two approaches: stochastic search methods and modelling methods.

2.3.1.1 Stochastic search methods

Stochastic search methods fall under the umbrella of algorithms (versus models) in machine learning, as no attempt is made to generate a predictive model from the search space. The search space is the domain of the solution, for which a theoretical function of the solution variables (in our context, synthesiser parameters) against the loss of the solution (the parameter configuration for the candidate sound) exists. The optimal solution to our problem, finding the best parameter configuration to recreate a given audio sample, is given at the minimum of this function. (The function is sometimes called the error surface.)

Befitting its name, a search method thus searches the space directly in some way for the optimal solution. In a *stochastic* search method, the initialisation and mutation of solution estimates over search iterations are *randomised*. Benefits of randomness include faster optimisation and reduced sensitivity to modelling errors.

Genetic algorithms [20] [28] [21] and particle swarm optimisation [18] [26] are two stochastic search algorithms seen repeatedly in recent work on sound matching.

2.3.1.2 Modelling methods

In contrast to search methods, modelling methods attempt to generate a predictive model from the search space, i.e. describe mathematically the relationship between an audio sample and its ideal parameter configuration, producing a general mapping between the two. Neural networks fall under this category.

CNNs [3] [15] and LSTM (or the extended variant, LSTM++) networks [43] [38] are two neural networks similarly seen in recent work on sound matching.

It should be noted that modelling methods and search methods are not necessarily mutually exclusive. Rather than being used to directly estimate solutions, search methods can be used as a means of optimising the hyperparameters (also known as "weights") of a model, improving their predictive accuracy.

2.3.2 Evaluation

Once a candidate sound has been produced, a technical evaluation of similarity can be performed by comparing the feature sets of the target and candidate sounds. (Once the parameters have been loaded into the synthesiser, the audio of the candidate sound will need to be rendered and its features extracted. During optimisation of the candidate, this is done repeatedly.) A simple means of comparison is the Euclidean distance between the two feature sets, giving a direct or indirect value for the loss/score. [43]

As a higher-level means of evaluation alongside technical evaluation, perceptual evaluation (listening tests) is also acknowledged in the literature. [41] In research to date, this addresses commercial viability for automatic synthesiser programmers as a creative aid, as well as rare but key scenarios in which results from feature analysis belie perceptual similarity between target and candidate sounds.

Perceptual evaluation is fundamentally relevant to this project, as there is no deducible benchmark for the error score in a cross-domain problem: we cannot be certain what is the nearest approximation to a foreign sound that a given synthesiser can produce, as it is a strongly non-convex optimisation problem. [43] Hence the minimum possible error score is unknown, a benchmark does not exist, and a technical evaluation is insufficient. This is in contrast to an intra-domain problem, where the benchmark is undeniably an error score of zero since the target sounds are sampled from the synthesiser.

2.4 Framework

There are multiple subprocesses involved in sound matching, namely feature extraction, parameter estimation, parameter loading, and sound rendering. Additionally, we must consider the implementation of the virtual synthesiser to emulate the Hammond organ. At the least, automating the overall process will require interfacing tools between different programming modules/libraries.

The Python language is recognised for its versatility and efficiency, continuously attracting a very active development community. It is also a popular language in machine learning and signal processing applications, leaving little doubt that there is a pool of suitable libraries to cover the range of processes concerned: digital signal processing, machine learning and/or optimisation, and audio manipulation. It is hence probable to implement the overall process solely using Python, interfacing different Python modules within a top-level script.

With regards to the literature: previous work from Yee-King et al. [43] acknowledges the availability and significance of frameworks in other applications of machine learning, and provides an original testbed used in this specific study—including machine learning models with testing and training data—as an open-source repository. [12] [14] (However, this is for VST programming applications and is hence not directly applicable to this project.) Collaborators from the University of Victoria have also developed and released SpiegeLib, an open-source "automatic synthesiser programming library," in the interests of furthering research in this field. [35]

Chapter 3

Design

3.1 Overview

The joint objectives of this project are:

- to develop an automatic synthesiser programmer with an embedded virtual synthesiser, modelling a Hammond organ
- and to further produce an evaluation of the resynthesis that determines its success in approximating the Hammond, hence in approaching the cross-domain problem referenced in 1.3 at large.

There are thus four aspects to consider discretely: obtaining audio samples of the Hammond, the design and implementation of the virtual synthesiser, the design and implementation of the automatic synthesiser programmer, and evaluation of the resynthesis. These are each considered through this chapter.

3.2 Implementation

Figure 3.1 is a block diagram outlining the overall process and implementation for the programmer, using a reinforcement learning approach. Note that the virtual synthesiser is represented as a process block, interfaced within the programmer. A description in text follows, and details regarding each aspect can be further found in the subsections. This design originates from Yee-King et al. [43]

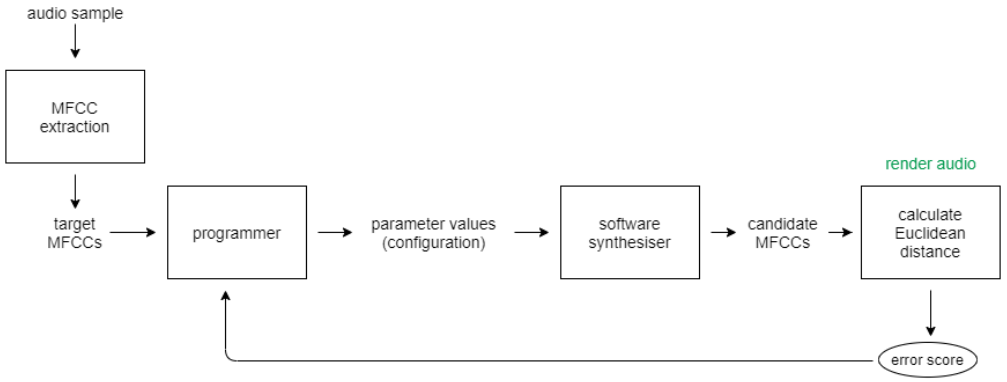


FIGURE 3.1: Block diagram of the overall process/implementation, using reinforcement learning

- The audio sample (waveform) of the target sound is uploaded and its MFCCs are extracted.
- The programmer outputs its estimation of the parameter values for the candidate sound.
- The parameter values are entered into the synthesiser and the audio for the candidate sound is rendered.
- The MFCCs of the candidate sound are extracted and the Euclidean distance between the target and candidate MFCCs is returned as an error score.
- If the final learning iteration has not been reached, this error score is returned to the programmer for optimisation of the candidate or model.

3.2.1 Feature extraction

MFCCs have been chosen to represent audio for this application. A comprehensive, well-cited justification for the use of MFCCs is presented in the second chapter of Yee-King's doctoral thesis [42] on techniques and applications for automatic synthesiser programming, with the salient points being its wide usage and validation for timbral analysis, its fundamental correlations with our perceptual appreciation of sound, and the availability of libraries for performing extraction.

To obtain the MFCCs of a digital audio sample, the sample is first divided into frames and each frame is multiplied by a set of windowing coefficients. The short-time Fourier transform (STFT) is then applied to each frame, the results of which are used to compute the MFCCs through a series of mathematical transformations.

Frame size, hop size, window type and window size are all variables of the STFT for consideration; to some extent, these have default values/types associated for use in audio applications.

Many open-source audio/sound libraries across languages offer MFCC extraction functions, unburdening the user of the mathematical processes and parameters involved. An example from Python is Librosa, which is exploited in SpiegeLib to further automate the comparison of MFCCs between audio files.

3.2.2 Programming

3.2.2.1 Dataset (samples)

As noted in 1.2.2, the Hammond organ is an instrument with a well-known legacy in music. Recorded samples of the instrument are hence likely to be found on websites, forums, etc. dedicated to providing instrument samples and loops under creative licence; such websites are a staple in the music production community.

Alleged samples of the Hammond B3 organ can be found on the website Freesound, uploaded by a registered user and released into the public domain under the Creative Commons 0 licence. [16] The sample pack contains recordings of individual notes spanning the range B2-B5, three full octaves, and an extra note G2 below. All recordings are 1-2 seconds long and provided as WAV files.

Through discussion within a social media group for sound synthesis and music electronics enthusiasts, these have been informally verified to be authentic samples of the Hammond organ.

3.2.2.2 Parameter estimation

Again informed by Yee-King's work, the genetic algorithm has been chosen as the means of parameter estimation to implement the programmer. Results presented in the third chapter of [42] conclude that "the genetic algorithm is the most effective optimiser for all of the fixed architecture synthesizers [sic]," which describes the virtual synthesiser to be implemented.

Note that since a search method is used as opposed to a model, the extracted MFCCs of the target sound are not involved in the estimation of parameters, only in the optimisation of these estimates.

3.2.3 Synthesis & rendering

3.2.3.1 Overview

Various sound synthesis and sound generation libraries exist for Python, many of which include functions to render the generated sound to a standard audio file format. One such library is pyo, a library for DSP script creation that implements classes for various common (and uncommon) signal processing elements—waveform generators, filters, audio effects, and so on. [5] Objects can be sequenced to create a custom virtual signal processing chain, the output of which can be rendered to a specified audio file format.

Using pyo, the virtual synthesiser can hence be implemented as a DSP script to generate a sound and immediately render it to an audio file. The necessary object attributes are treated as synthesiser parameters and initialised as variables from an array.

3.2.3.2 Virtual synthesiser: emulating the Hammond

The aim with the virtual synthesiser is to create a suitable model of the Hammond organ, such that the organ's sound can be approximated to a convincing extent given a specific set of parameters. This does not necesssarily mean reproducing each component within the organ's architecture explicitly; a suitable simplified model can fulfill the purpose with great efficiency. This is hence the approach taken in the design of the virtual synth.

It is noted that the proposed model is a critical aspect of this project and must be carefully considered; if it cannot feasibly model the sound generation of a Hammond organ, there is no means of success towards the project objectives.

The Hammond organ can be considered as three distinct, sequenced main parts in the given order: the sound generator, the vibrato scanner, and the Leslie speaker. These are now expanded upon.

Sound generator: As explained in 1.2.2, the sound generator is an additive synthesiser mixing nine unique sine tones, whose frequencies are fixed harmonic intervals[1] of the fundamental frequency produced by one of the tonewheels. [33]

[1] As follows: sub-octave, fifth, octave, octave & fifth, two octaves, two octaves & third, two octaves & fifth, three octaves

(Recall that this fundamental frequency is determined by the note being played.) The amplitude of each sine is variable, underpinning the variety in timbre the organ offers.

Vibrato scanner: Often referred to simply as the scanner, this produces a unique effect by sweeping the sound output across taps in a short delay line at the output of the sound generator, as illustrated in 3.2. The result is a shimmering quality to the sound, likened to vibrato and chorus but not identical; it is caused by the variation in relative phase between the emitted sine tones at different taps.

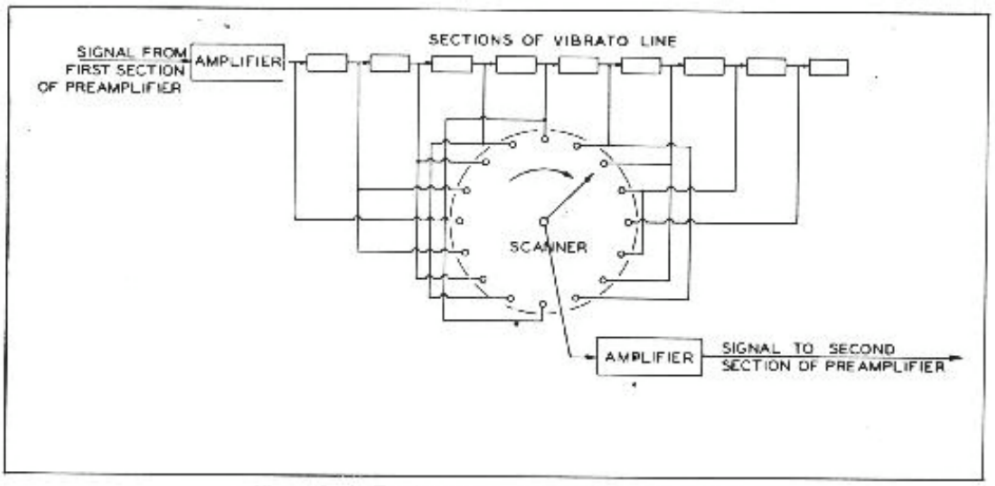


FIGURE 3.2: Block diagram of the vibrato scanner, from p37 of the service manual [8]

The scanner can thus be modelled using a simple delay effect whose delay time is modulated by a low-frequency sine oscillator.[2] The Hammond has multiple modes of use for the scanner, some of which mix the swept delay output with the original signal, hence a mixer for both versions of the signal is also implemented.

Leslie speaker: The Leslie speaker is in fact a pair of rotary speakers partially enclosed in a wooden cabinet. The treble speaker reproduces frequencies above 800Hz, while the bass speaker reproduces those below the same threshold. Both speakers notably rotate at independent speeds.

The effects of the Leslie speaker are by far the most complex, as various physical interactions of sound are invoked. These can be roughly inferred as phasing, reverberation (or reverb), resonance and low-pass filtering due to reflection and

[2]It may seem incorrect to use a continuous function as a modulator when the scanner architecture shows discrete taps, but this actually accounts for the glide in the output's pickup as it moves between taps.

transmission between mediums, as well as simultaneous tremolo and vibrato—the former due to rotation, the latter the Doppler effect.

To acknowledge the independent rotation speeds of the treble and bass speakers, the signal should be filtered into corresponding high-passed and low-passed signals which are modulated independently, again using a low-frequency sine oscillator. It is further surmised that effects like phasing, tremolo and vibrato can be (rather roughly) encompassed and induced through a modulated delay line, which is already present. Finally, reverb, low-pass filtering and parametric equalisation (for resonance) are added to the signal chain to represent the remaining effects.

Figure 3.3 is a block diagram hence summarising the design of the virtual synthesiser, illustrating the signal path and modulation. This represents the simplest design deemed reasonable to mimic the organ's acoustics, as greater complexity would disproportionately affect the time taken[3] for convergence of the programmer while yielding uncertain and diminishing returns on the final fitness score.

It should be noted that just as an effect could be bypassed on a Hammond organ, so can it be bypassed in this virtual implementation by assigning certain parameters on the relevant object a value of zero. There are no situations in which this would break the signal path.

3.2.4 Evaluation

3.2.4.1 Technical evaluation

As shown in Figure 3.1, once the programmer's estimated parameters are loaded into the synthesiser, the audio of the candidate sound is rendered and its MFCCs extracted. An error score is given from the Euclidean distance between the MFCCs of the target and candidate sounds, which is fed back to the programmer for optimisation if the final learning iteration has not been reached. At the end, a final candidate is produced with an associated final error/fitness score, indicating how much distance the algorithm achieved from the initial error. This lends insight into the synthesiser's performance.

There are multiple statistics to draw from the convergence of the programmer towards a detailed technical evaluation of the project. For each sample, the error score convergence and final error score will be recorded from the programmer during resynthesis, towards illustrating this.

[3] On a scale of hours.

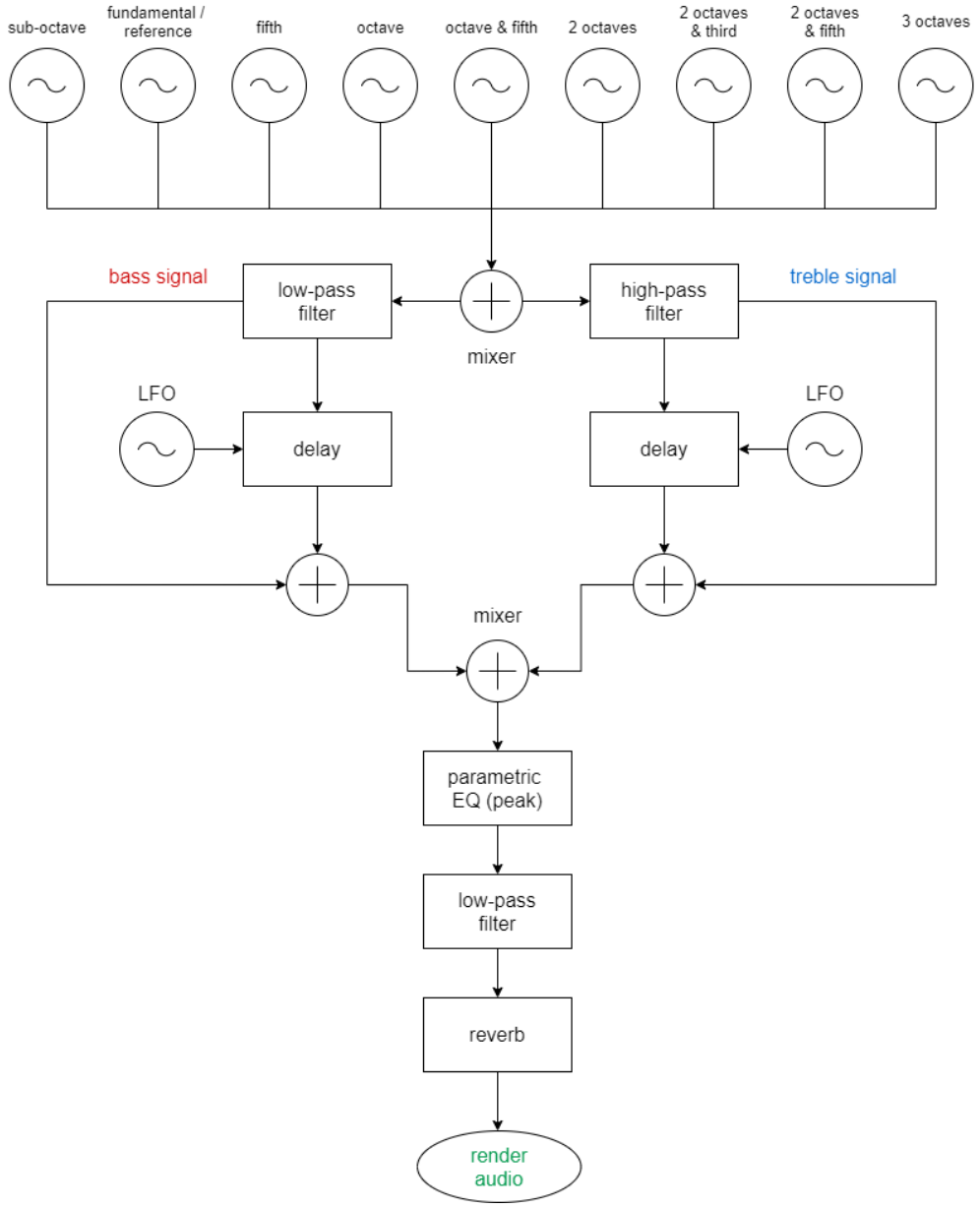


FIGURE 3.3: Block diagram of the virtual synthesiser

3.2.4.2 Perceptual evaluation

As there is no benchmark for our technical evaluation, a perceptual evaluation in the form of a user listening study will lend further critical insight into the synthesiser's performance and general success of the project. Volunteers will be invited to participate in an asynchronous online questionnaire in which they are asked to rate the similarity of multiple sample-resynthesis pairs. Demographic information on experience with sound analysis[4] will also be collected.

[4]Such as sound synthesis, audio processing, acoustics, etc.

Chapter 4

Development

4.1 Overview

The programmer is implemented across three interfaced scripts: `hammond.py`, `evaluate.py` and `programmer.py`. These are contained in a working directory with organised subdirectories for storing samples, resynthesised sounds and process results.

`programmer.py` is the executable script, representing the overall recursive process for resynthesis. `hammond.py` and `evaluate.py` are modules each defining a function to be called for each process iteration of the programmer. The former implements the virtual synthesiser and renders a candidate sound to an audio file for a given set of parameters, while the latter returns the error score for the candidate sound as described in 3.2.4.1.

The process for resynthesising the entire directory of samples is automated through `programmer.py`, including functionality to resume from the last completed resynthesis if interrupted.

4.2 hammond.py

Using pyo as introduced in 3.2.3.1, the virtual synthesiser is implemented in `hammond.py`. The script interfaces various objects as described in the library's documentation [6] to recreate the architecture presented in Figure 3.3.

The audio server and objects are instantiated within a defined function. This function accepts an array of parameters for these objects, as well as the fundamental frequency, duration and filename for the candidate sound as arguments, allowing `programmer.py` to set these dynamically at runtime. This enables the process to be automated for the entire directory of samples through the top-level script.

All parameters (normalised where necessary) are continuous values in the range 0-1, simplifying the specifications to the genetic algorithm in `programmer.py`. There are 20 parameters specified in total. Furthermore, frequencies for each of the nine oscillators are automatically derived from the given fundamental frequency using multipliers defined from cents. [31]

After the signal path is defined, the sound at the output is rendered as a wav file to the subdirectory `./resynthesis`, with the filename specified to the function. This filename is reused for all convergence iterations seen by `programmer.py`.

4.3 evaluate.py

In `evaluate.py`, the process of extracting MFCCs for a target and candidate sound, comparing these and calculating the error score is again implemented within a defined function. This function accepts filenames for both the target and candidate audio files, as well as the note name, e.g. A4. (Note names are unique to individual samples.) These are again dynamically specified by `programmer.py`.

MFCC extraction and calculation of the Euclidean distance are implemented using classes from SpiegeLib, [39] [35] briefly introduced in 2.4. A JSON file is created and edited in the subdirectory `./stats` to extract the Euclidean distance from a set of error metrics returned at each comparison. The filename is prefixed by the note name.

4.4 programmer.py

`programmer.py` is the executable script in the working directory. Within, the entire directory[1] of samples is iterated over to produce a corresponding resynthesis and log the results of the programmer's convergence for this resynthesis.

[1] `./18844_hammondman_tonewheel-organ-sound-samples`

Classes from the library geneticalgorithm [37] are used to implement the programmer's genetic algorithm to repeatedly estimate and converge parameters for the synthesiser. The algorithm uses elitist selection, notably with a population size of 500 and a mutation rate defined as the reciprocal of the number of parameters. (This choice for the mutation rate is informed by 3.3.3 of [42], while a suitable population size was inferred from the library's examples.) No maximum number of iterations is specified, allowing for a fully natural convergence.

A custom error function is defined for the algorithm which calls the functions defined in `hammond.py` and `evaluate.py`. This function returns the error score for a given candidate sound as a float scalar. From lines 61-63 of `programmer.py`:

```
def errorfunc(X):
    hammond.render(resynth_fn, X, frequency, duration)
    return evaluate.run(target_fn, resynth_fn, note)
```

Once the algorithm has converged for a given sample and a final candidate sound has been rendered, the programmer stores the algorithm's output to a text file in `./stats`, titled with the note name. This output is a record of the error score for the fittest candidate in each population during convergence, including the error score of the final candidate. Upon convergence, the programmer also appends the note name of the sample to a text file in the working directory, `log.txt`.

`log.txt` records each completed resynthesis through note names. Each time `programmer.py` is launched, the script first checks against this log to avoid repeating any resynthesis that has previously been completed, allowing the program to be run and exited at will.

Chapter 5

Project Management

5.1 Development & testing

It is evident from Chapters 3 and 4 that the implementation of the project is considerably complex, with several points of interface both within and between the synthesiser and programmer. (Refer again to Figures 3.1 and 3.3.) Many hidden points of interface also exist where multiple subtasks are required to implement a process, due to the functionality offered by chosen libraries. A highly iterative approach to development and integration was thus necessary, supported by frequent and repetitive testing alongside version control.

This approach was realised using Jupyter Notebooks [22] during active development, allowing for easy isolation and segmentation of code for testing interactively. This was exploited in three phases of application and testing for each object implemented: class testing, function testing and integration.

Class testing instantiated and tested objects following the documentation's own examples, ensuring the environment was configured correctly and supported the object. Function testing would then implement the object as intended for the design and test that it functioned as expected. Integration would finally test it interfaced with other objects as specified in the design, within the scope of the process represented by the script. In the later stages of development, as more experience was gained with the classes and opportunities for efficiency were seized, class testing was made redundant, followed by function testing.

Details of progress and notes during development were recorded in a dated project diary, documenting every action taken, resource consulted and problem overcome

in fulfilling the implementation. In this diary, clear milestones and development criteria were outlined at each stage of progress and marked with feedback upon completion. Maintaining the project diary was a form of risk management, discussed below.

5.2 Risk management

As a basic software project, this project carried no special risks. However, there were of course fundamental risks to address, especially in pursuing a novel research problem: these can be summarised as time, resource and contingency management. Although these are discrete categories, it is often the case that at least two can be addressed simultaneously through certain actions.

A virtual environment was created to ensure a stable package environment throughout development, given the amount of explicit and implicit dependencies anticipated. As mentioned in 5.1, a project diary was also maintained and stored on a personal remote filedrive provided by the University, for accessibility and security. Both of these actions served as contingency management for the project, curtailing potential time spent debugging during development.

Where possible, alternative libraries for core aspects of implementation were documented and in some cases tested. Examples are the mlrose library [17] for implementation of stochastic search algorithms, including genetic algorithms, and various sound/music libraries [23] for audio manipulation, feature extraction and digital signal processing. This was another action under contingency management.

The main machine for development, testing and execution was a personal laptop. Additionally however, once development was complete, the program and working directory were copied onto the Lyceum (Iridis 4) computing servers [27] as a secondary machine. This was action towards resource and time management, noting that the convergence of genetic algorithms is generally time-consuming and computationally expensive. This allowed progress on the sample set to continue on occasions where the student's laptop had to be dedicated to other assignments or was otherwise a limiting factor.

Last but not least, all development and project files were regularly backed up to personal remote filedrives provided by the University and department, ensuring cloud access. Hence in the worst case scenario that the student's laptop encountered technical failures, remote or on-campus computing facilities could be used to

continue development—especially with the support of the project diary, as noted earlier.

5.3 Project timeline & retrospective

Figure 5.1 recounts the actual project timeline, summarised from the project diary. Figure 5.2 presents the original project schedule submitted in December's progress report.

Comparing these, there are two things to note: the calendar period differs between the two and so do the outlined tasks, to some degree. This reflects a redirection with the project between now and the time of writing the progress report: the original aims cited in the progress report were to implement an automatic synthesiser programmer for use with VST synthesisers, recreating existing experiments (1.3) within the field. However, the required attempt to develop a VST host failed, due to unresolved errors using an unfamiliar audio programming framework. Furthermore, no verified VST hosting libraries could be found for use on Windows. Hence being mindful of time constraints, an alternative project aim to explore the cross-domain problem arose as a means of progress without the need for VST hosting functionality. The project diary has documented the iterations/evolution of this project since.

A direct comparison hence cannot be drawn between the previous schedule and actual timeline. That being said, comments can be made on the projected versus actual duration of their shared development subtasks.

At the time the previous schedule was drafted, there had not been a thorough review of libraries and frameworks relevant to the project, and it was also anticipated that certain processes—such as MIDI/audio rendering—would need to be implemented independently within the VST host. Hence in retrospect, the duration of some tasks is overestimated, such as implementing the genetic algorithm (programmer) and MFCC extraction & comparison, along with audio rendering. Other tasks however, such as (final) testing & evaluation, compare well in projected duration to their corresponding tasks in the actual timeline.

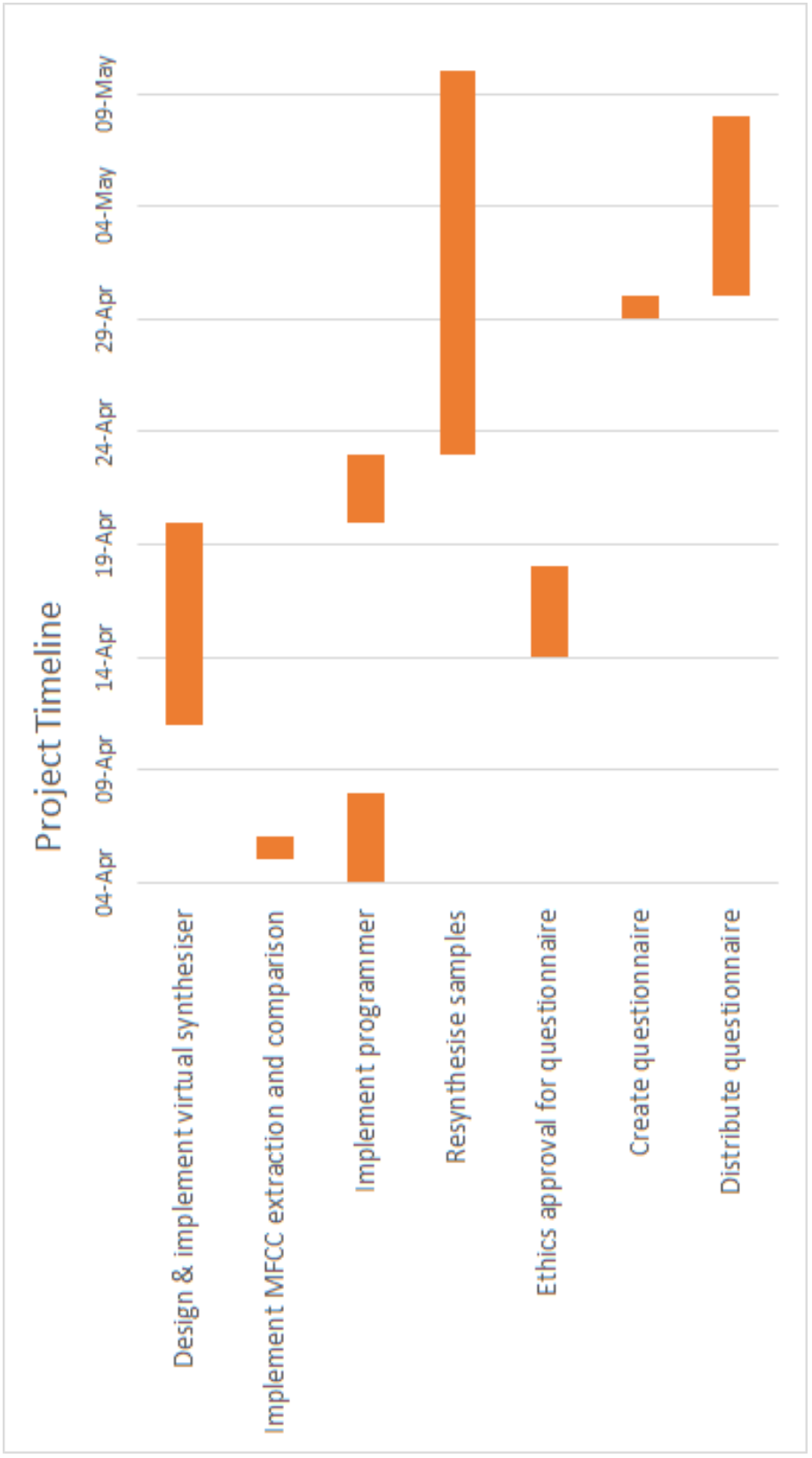


FIGURE 5.1: Project timeline as Gantt chart

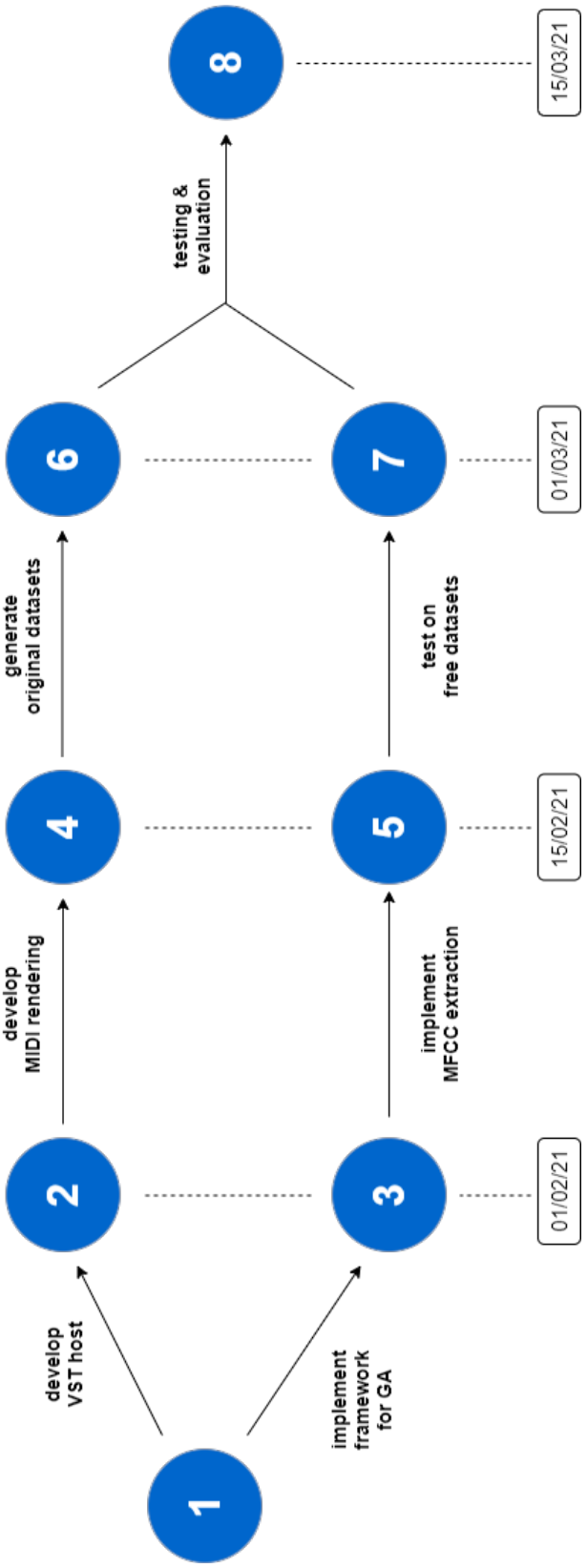


FIGURE 5.2: Original project schedule as PERT chart

Chapter 6

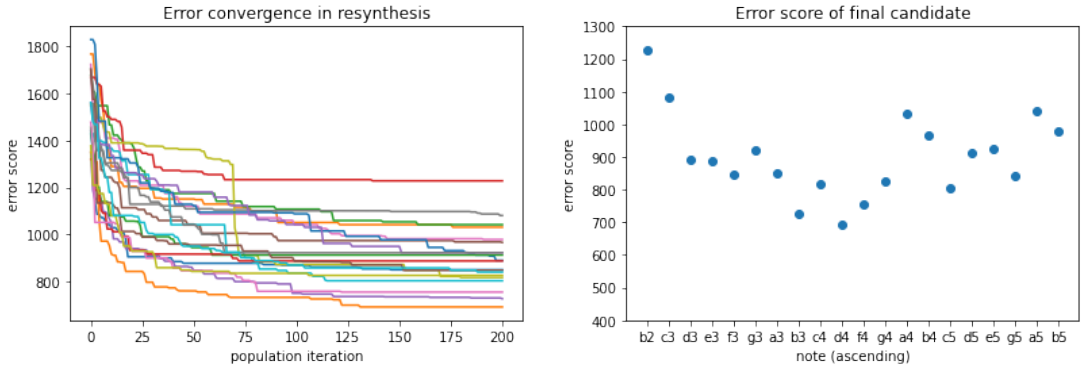
Results & Evaluation

6.1 Results

All development criteria for the implementation were met, and all 23 sample sounds were successfully resynthesised. A 100% success rate was observed for convergence, confirming that the genetic algorithm is applicable to cross-domain optimisation. The time taken for each resynthesis was consistently between 6-7 hours.

6.2 Evaluation

6.2.1 Technical evaluation



(a) Error convergence in resynthesis

(b) Final error scores for resynthesised sounds

FIGURE 6.1: Results of convergence across samples

Figure 6.1 presents the results of convergence. The mean final error score for the resynthesised sounds is 909.2 with a standard deviation of 130.1, or 14.3% of the mean. Figure 6.2 further shows the total percentage error reduction across the resynthesised sounds as a frequency distribution.

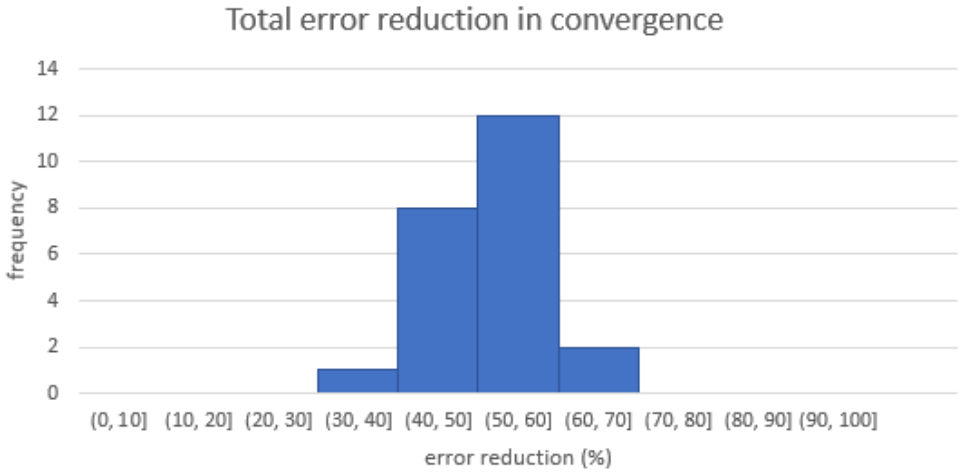


FIGURE 6.2: Total percentage error reduction among resynthesised sounds

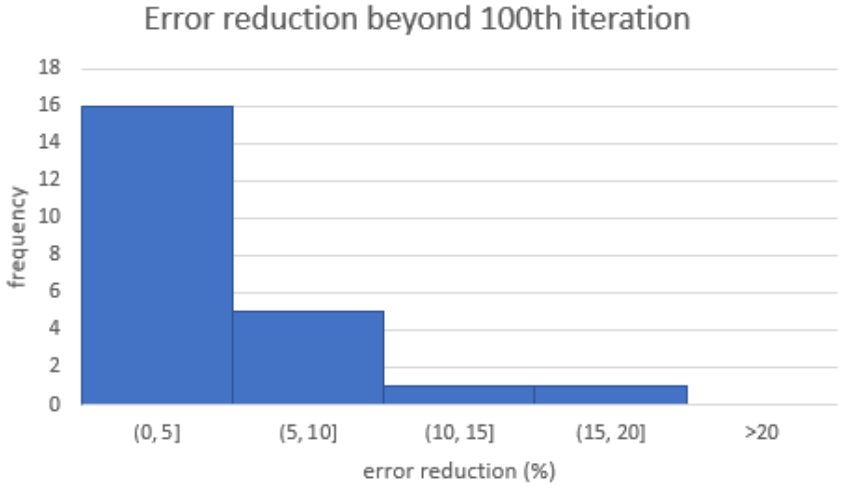


FIGURE 6.3: Percentage error reduction after 100th iteration

Reviewing Figure 6.1, it appears that the majority of convergence for each resynthesis is achieved by the 100th iteration of the population. This is further considered in Figure 6.3 through a frequency distribution. The mean percentage error reduction after the 100th iteration is 3.84% with a standard deviation of 4.19%. Ignoring outliers defined to be 1.5× the standard deviation away from the mean, the maximum reduction after the 100th iteration is 7.90%.

6.2.2 Perceptual evaluation

As intended, a listening study was arranged and approved by the University's Faculty Research Ethics Committee (FREC), under Ethics/ERGO number 64480. This was delivered as an online questionnaire, in which volunteering participants were asked to rate the timbral similarity of six pairs of randomly selected target/resynthesis sounds. Ratings were given on a scale of 1–10. Participants were advised to use good-quality headphones at an appropriate volume.

Alongside general distribution, the questionnaire was also distributed to undergraduate & MSc students in Acoustics and PhD students in the University's ISVR[1] group, permitted under the study's approval. Within the questionnaire, participants were asked for their experience with sound analysis (None/Amateur/Student/Professional) and allowed to provide further comments.

A total of 27 complete responses were received from participants with a diversity of experience, including audiologists, live sound engineers, instrumentalists and music producers.

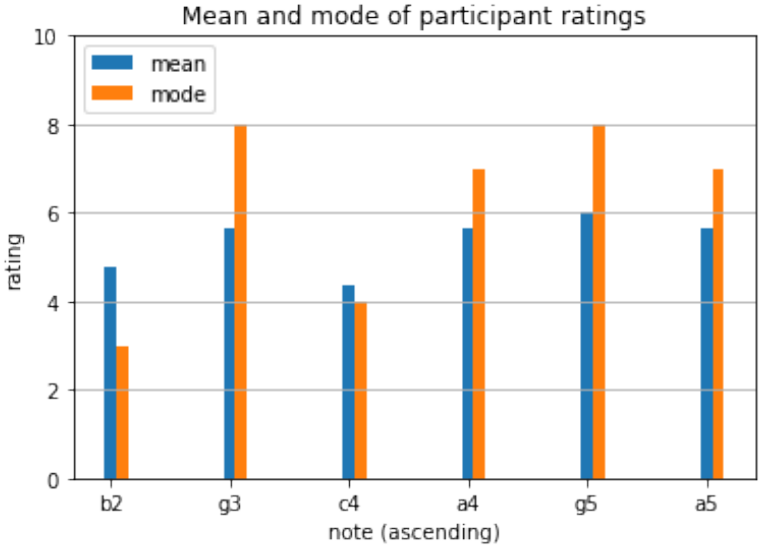


FIGURE 6.4: Mean and modal ratings for sound pairs

Figure 6.4 presents the mean and modal ratings for the target-resynthesis sound pairs. Figure 6.5 further breaks this down by presenting mean ratings for each pair classified by experience of the responder.

[1]Institute of Sound and Vibration Research.

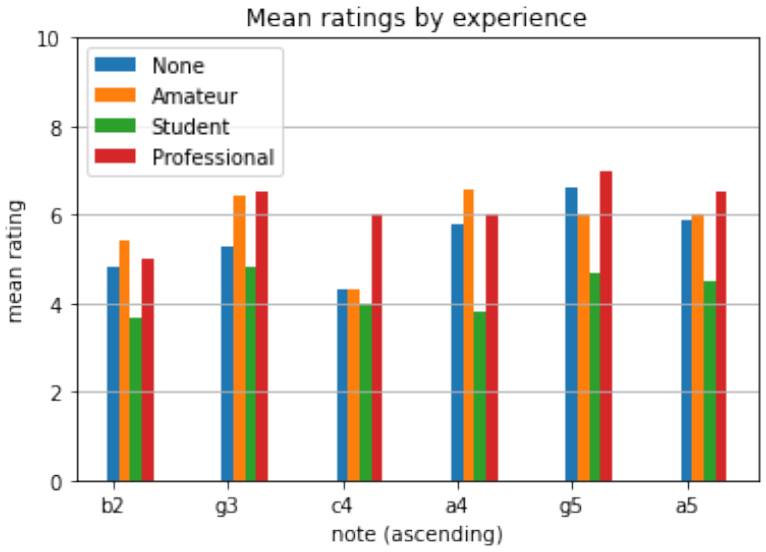


FIGURE 6.5: Breakdown of mean ratings by alleged experience

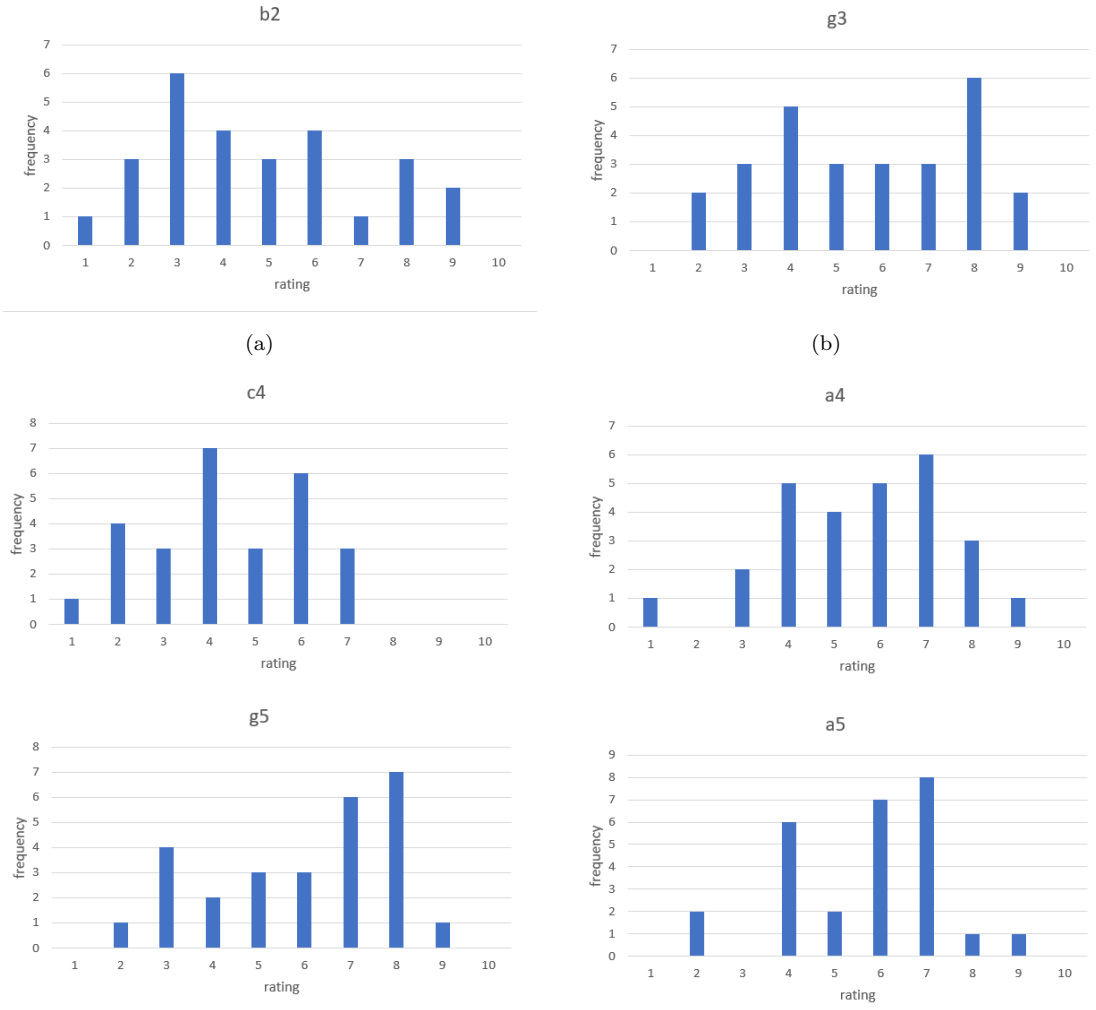
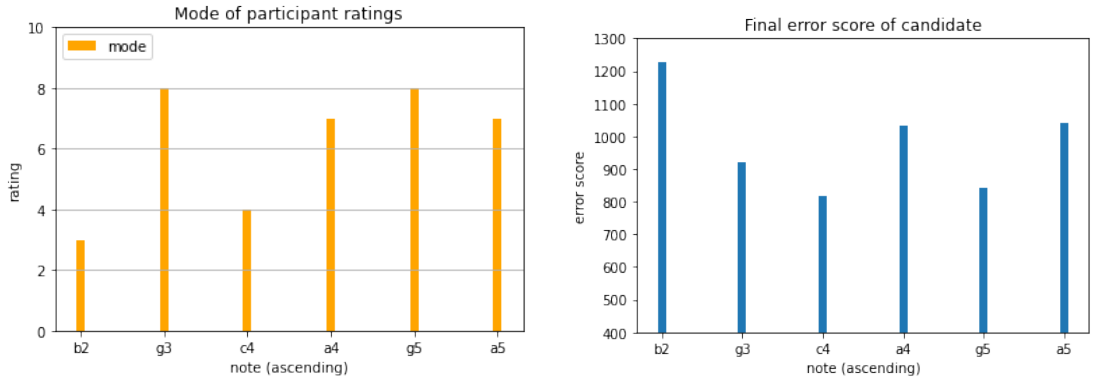


FIGURE 6.6: Frequency distribution of received ratings for each sound pair

Figure 6.6 presents more detailed results with a frequency distribution of the ratings received for each sound pair. A high variance is observed in these data across all pairs, though with appreciable positive or negative skew for each.

6.2.3 Correlation between evaluations

To close this evaluation, results from technical and perceptual evaluations are compared for correlation. This is presented in Figure 6.7.



(a) Modal ratings from participants

(b) Final error scores of candidates

FIGURE 6.7: Comparing technical and perceptual evaluations for resynthesised sounds

Modal ratings are considered from the perceptual evaluation, due to the high variance in ratings for each sound pair resulting in minimal deviation among mean ratings. Relatively, it would be expected for a higher modal rating to correlate with a lower final error score.

A moderate negative correlation can be seen comparing the modal rating and error score across candidates, although it is inconsistent at points such as in comparing C4 to its neighbours G3 and A4. This reinforces that technical and perceptual means of evaluation should be considered jointly in cross-domain optimisation, and the correlation between the two under partial optimisation merits investigation.

6.2.4 Analysis

From 6.2.1, a mean final error score of 909.2 with a standard deviation of 14.3% evidences a consistent settling range for convergence. This implies the programmer succeeds at partial optimisation, hence the design of the virtual synthesiser is

successful to some degree in emulating the Hammond organ. This is further supported by statistics on the total percentage error reduction in resynthesis (Figure 6.3), which shows an overwhelming majority of candidates achieving reductions between 40-60%, positively skewed within this range.

The conclusion of partial optimisation is further supported in the perceptual evaluation, where although notions of perceptual similarity vary among participants given the variances displayed in Figure 6.6, modal ratings and skewness of the ratings distribution indicate some generally concludable degree of similarity for each sound pair.

A crucial takeaway is that partial optimisation yields varying results for perceptual similarity among the candidates. The skewness and modal rating for G5 (refer again to Figure 6.7) suggests a high degree of perceptual similarity; this candidate has the second lowest error score of those surveyed, coinciding with expectations. Similarly, results for B2 suggest a low degree of perceptual similarity, validated by this candidate presenting the highest error score of those surveyed. Yet C4, the candidate with the lowest error score in the group, also ranks low in perceptual similarity by the same metrics.

Hence as noted in 6.2.3, as further progress is made exploring the cross-domain problem in automatic synthesiser programming, the correlation between technical and perceptual evaluations under partial optimisation should be studied accordingly for novel trends and insights.

Chapter 7

Conclusions

7.1 Critical evaluation

Through partial optimisation, the project is successful in demonstrating the prospects of automatic synthesiser programming in addressing the cross-domain problem, as introduced in 1.3. This demonstration is limited by the simplicity of the virtual synthesiser designed to emulate the Hammond organ, yet it is acknowledged that this was itself a crucial factor to the completion hence success of the project, given the scope and time available.

7.2 Further work

There is a clear opportunity for further development in improving the emulation of the Hammond organ. As noted in 1.2.2, digital emulations of the Hammond organ and/or its effects exist as commercially available VST plugins. By replacing the virtual synthesiser designed in this project with one of these established emulations, significantly better results for resynthesis of the sampled Hammond sounds could be achieved.

Towards this end, the virtual synthesiser should be replaced by a VST host application which would allow the VST plugin to interface with the programmer. The RenderMan library [12] [13] provides VST hosting capability in Python and is also embedded as a class within SpiegeLib. Note that the library is only available for use on Mac OS and Linux.

It is suggested in the results of [43] that a Euclidean distance of 10–15 or less between target and candidate MFCCs correlates with almost identical perceptual similarity. This is perhaps worth noting as an informal benchmark for improvements in emulation.

References

[1] Noor Almaadeed, Muhammad Asim, Somaya Al-Maadeed, Ahmed Bouridane, and Azeddine Beghdadi. Automatic Detection and Classification of Audio Events for Road Surveillance Applications. *Sensors*, 18(6):1858, Jun 2018. ISSN 1424-8220.

[2] Francesc Alías, Joan Socoró, and Xavier Sevillano. A Review of Physical and Perceptual Feature Extraction Techniques for Speech, Music and Environmental Sounds. *Applied Sciences*, 6(5):143, May 2016. ISSN 2076-3417.

[3] Oren Barkan and David Tsiris. Inversynth: Deep estimation of synthesizer parameter configurations from audio signals. *CoRR*, abs/1812.06349, 2018.

[4] Kenneth Bindas and Ted Gioia. The History of Jazz. *The Journal of American History*, 87:265, 06 2000.

[5] Olivier Bélanger. Pyo - ajax sound studio. `http://ajaxsoundstudio.com/software/pyo`, 2020.

[6] Olivier Bélanger. Welcome to the pyo 1.0.3 documentation. `http://ajaxsoundstudio.com/pyodoc`, 2020.

[7] Hugo Caracalla and Axel Roebel. Sound texture synthesis using convolutional neural networks. *CoRR*, abs/1905.03637, 2019.

[8] Hammond Organ Company. *Hammond Organ B3–C3 Service Manual*. Hammond Organ Company.

[9] Juan-Pablo Cáceres. Sound design learning for frequency modulation synthesis parameters. 08 2009.

[10] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, 1980.

[11] A. Eronen. Comparison of features for musical instrument recognition. In *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)*, pages 19–22, 2001.

[12] Leon Fedden. fedden/RenderMan: The v1.0.0 release for publication of paper, December 2017.

[13] Leon Fedden. GitHub - fedden/RenderMan: Command line C++ and Python VSTi Host library with MFCC, FFT, RMS and audio extraction and .wav writing. https://github.com/fedden/RenderMan, 2017.

[14] Leon Fedden and Matthew John Yee-King. Automatic VST Programmer Experimental Framework. https://doi.org/10.5281/zenodo.1080859, 2017.

[15] Leonardo Gabrielli, Stefano Tomassetti, Stefano Squartini, and Carlo Zinato. Introducing Deep Machine Learning for Parameter Estimation in Physical Modelling. 2017.

[16] hammondman. Freesound - pack: tonewheel organ sound samples by hammondman. https://freesound.org/people/hammondman/packs/18844, 2016.

[17] Genevieve Hayes. mlrose: Machine Learning, Random Optimisation and SEarch — mlrose 1.3.0 documentation. https://mlrose.readthedocs.io, 2019.

[18] Sebastian Heise, Michael Hlatky, and Jörn Loviscach. Automatic cloning of recorded sounds by software synthesizers. *Journal of the Audio Engineering Society*, october 2009.

[19] Hynek Hermansky. Speech recognition from spectral dynamics. *Sadhana*, 36, 10 2011.

[20] Andrew Horner, James Beauchamp, and Lippold Haken. Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis. *Computer Music Journal*, 17(4):17–29, 1993. ISSN 01489267, 15315169.

[21] Colin Johnson. Exploring the Sound-Space of Synthesis Algorithms Using Interactive Genetic Algorithms. 09 1999.

[22] Project Jupyter. Project Jupyter — Home. https://jupyter.org/, 2021.

[23] Neil McBurnett. PythonInMusic - Python Wiki. https://wiki.python.org/moin/PythonInMusic, 2019.

[24] Dalibor Mitrović, Matthias Zeppelzauer, and Christian Breiteneder. Chapter 3 - Features for Content-Based Audio Retrieval. In *Advances in Computers: Improving the Web*, volume 78 of *Advances in Computers*, pages 71 – 150. Elsevier, 2010.

[25] A. B. Nielsen, S. Sigurdsson, L. K. Hansen, and J. Arenas-Garcia. On the Relevance of Spectral Features for Instrument Classification. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 2, pages II–485–II–488, 2007.

[26] Gustavo Nishihara and Tiago Tavares. Parameter estimation of FM synthesis. *Revista dos Trabalhos de Iniciação Científica da UNICAMP*, (27), November 2019.

[27] University of Southampton. The Iridis Compute Cluster — iSolutions — University of Southampton. https://www.southampton.ac.uk/isolutions/staff/iridis.page, 2021.

[28] Palle Dahlstedt. Creating and Exploring Huge Parameter Spaces: Interactive Evolution as a Tool for Sound Generation. In *Proceedings of the International Computer Music Conference, Habana, Cuba*, 2001.

[29] Dietrich W. R. Paulus and Joachim Hornegger. *Spectral Features and Speech Processing*, pages 214–236. Vieweg+Teubner Verlag, Wiesbaden, 1995. ISBN 978-3-322-87867-0.

[30] G. Peeters, Bruno L. Giordano, P. Susini, N. Misdariis, and S. McAdams. The timbre toolbox: extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America*, 130 5:2902–16, 2011.

[31] B.H. Suits. Physics Department, Michigan Technological University. Making Sense of Cents. https://pages.mtu.edu/~suits/cents.html, 1998.

[32] The Pulsar. Generating sine wave from square waves. http://www.thepulsar.be/article/generating-sine-wave-from-square-waves, 2015.

[33] Gordon Reid. Synthesizing Tonewheel Organs: Part 1. *Sound on Sound*, 2003.

[34] Jens Schroder, Niko Moritz, Marc Schadler, Benjamin Cauchi, Kamil Adiloglu, Jörn Anemüller, Simon Doclo, Birger Kollmeier, and Stefan Goetze.

On the use of spectro-temporal features for the IEEE AASP challenge 'detection and classification of acoustic scenes and events'. pages 1–4, 10 2013.

[35] Jordie Shier, George Tzanetakis, and Kirk McNally. SpiegeLib: An automatic synthesizer programming library. 05 2020.

[36] Kai Siedenburg and Stephen McAdams. Four Distinctions for the Auditory "Wastebasket" of Timbre. *Frontiers in Psychology*, 8:1747, 10 2017.

[37] Ryan (Mohammad) Solgi. geneticalgorithm · PyPI. https://pypi.org/project/geneticalgorithm, 2020.

[38] Nathan Sommer and Anca Ralescu. Towards a Machine Learning Based Control of Musical Synthesizers in Real-Time Live Performance. *CEUR Workshop Proceedings*, 1144:61–67, 01 2014.

[39] University of Victoria. SpiegeLib–SpiegeLib 0.0.4 documentation. https://spiegelib.github.io/spiegelib, 2019.

[40] Lonce Wyse. Audio Spectrogram Representations for Processing with Convolutional Neural Networks. *CoRR*, abs/1706.09559, 2017.

[41] Matthew Yee-King and Martin Roth. Synthbot: An unsupervised software synthesizer programmer. 2008.

[42] Matthew John Yee-King. *Automatic sound synthesizer programming: techniques and applications*. PhD thesis, University of Sussex, September 2011.

[43] Matthew John Yee-King, Mark d'Inverno, and Leon Fedden. Automatic Programming of VST Sound Synthesizers using Deep Networks and Other Techniques. *IEEE Transactions on Emerging Topics in Computational Intelligence 1*, 2018.

[44] Cynthia Zhang and Zbigniew Ras. Analysis of Sound Features for Music Timbre Recognition. pages 3–8, 04 2007.

Appendices

Appendix A

notes	mean	mode
b2	4.8	3
g3	5.7	8
c4	4.4	4
a4	5.6	7
c4	6.0	8
c4	5.7	7

TABLE 1: Mean and modal ratings for sound pairs, relevant to Figure 6.4

	b2	g3	c4	a4	g5	a5
None	4.8	5.3	4.3	5.8	6.6	5.9
Amateur	5.4	6.4	4.3	6.6	6.0	6.0
Student	3.7	4.8	4.0	3.8	4.7	4.5
Professional	5.0	6.5	6.0	6.0	7.0	6.5

TABLE 2: Mean ratings by experience, relevant to Figure 6.5

Appendix B

Below is a brief list of contents for the design & data archive, jointly submitted with this report.

- **evaluation:** A directory containing spreadsheets and interactive notebooks used to process data from evaluation, as well as the resulting figures.
- **hammond:** The working directory for the project, reviewed in Chapter 4.
- **archive.zip:** Deprecated design and data files generated in the process of development, including those not relevant to the final iteration of the project. All files within are referenced in the project diary.
- **Project Diary.docx:** The project diary referenced in Chapter 5.

Appendix C

In fulfilment of the project report standards, the following pages present a copy of the original project brief. This is as submitted in October 2020.

COMP3200: Part III Individual Project Brief

Farida Yusuf
30389038
fy1n18@soton.ac.uk

Project Title: Music, machines & mimicry: exploring deep learning, audio parameterisation and APIs for independent sound re-synthesis

Supervisor: Dr. Jize Yan

Description:

Sound synthesis is the process of electronically generating sounds and timbres for use in music production. These can recreate acoustically produced sounds which can be heard from instruments, objects or landscapes in real life, or create completely new, artificial sounds & effects from the imagination and beyond. There are many methods of synthesis, such as subtractive, granular, wavetable, FM and so on, but the principle remains the same across all methods: waves in the audible frequency range are manipulated in the time and/or frequency domains to achieve a particular sound, of finite or infinite duration.

The goal of this project is to investigate the optimal implementation of a deep learning algorithm for independent sound re-synthesis by an AI, and to produce a corresponding model for demonstration. This includes identifying the most suitable type of neural network for the task, a means of parameterising or representing audio/sound for the model's input & output, and a programming interface through which it can output the results of its re-synthesis attempts. Hence the product would be a software program that accepts a digital audio sample within given constraints, processes this sample according to the embedded deep learning algorithm, and interfaces with an application to output a generated sound, which is the re-synthesis of the original sample. The original sample and re-synthesised output would be compared by means of spectral and temporal analysis tools, such as in MATLAB or Python.

Stretch goals for this project would include:

- automated, adaptive filtering of complex audio productions to isolate tracks (i.e. separate sounds/timbres) as samples for input
- extension of the application interface to allow users to select, play and modify re-synthesised sounds via user controls & MIDI interfacing

Appendix

Plan/Schedule:

- Complete Andrew Ng's Coursera courses on Machine Learning & Deep Learning; hands-on learning with TensorFlow
    - *(1 month, now – Friday 13th November 2020)*
- Set up Final Project Report outline in LaTeX and begin drafting
    - *(1 week, Friday 23rd October – Friday 30th October 2020)*
- Literature review: research sound/music in machine/deep learning contexts to inform audio parameterisation
    - *(3 weeks, Friday 30th October – Friday 20th November 2020)*
- Research existing APIs for sound synthesis OR research audio programming & parameter interfacing for software synthesis in C/Python
    - *(2 weeks, Friday 13th November – Friday 27th November 2020)*
- Complete specification for deep learning algorithm, audio parameterisation and programming interface
    - *(1 week, Friday 27th November 2020 – Friday 4th December 2020)*
- **Progress report**
    - ***(due Tuesday 8th December 2020)***
- **Semester 1 exams**

- Initial implementation of deep learning model
    - *(1 month, Wednesday 3rd February 2021 – Wednesday 3rd March 2021)*
- Source and compile training & testing datasets for algorithm
    - *(1 month, Wednesday 10th February 2021 – Wednesday 10th March 2021)*
- Initial implementation of application and interface for sound re-synthesis
    - *(2 weeks, Wednesday 3rd March 2021 – Wednesday 17th March 2021)*
- Hands-on learning with signal processing in Python/MATLAB for model evaluation
    - *(2 weeks, Wednesday 10th March 2021 – Wednesday 24th March 2021)*
- Program testing
    - *(1 week, Wednesday 24th March 2021 – Wednesday 31st March 2021)*
- Model fine-tuning and retesting
    - *(2 weeks, Wednesday 31st March 2021 – Wednesday 14th April 2021)*
- Final model evaluation
    - *(1 week, Wednesday 14th April 2021 – Wednesday 21st April 2021)*
- **Final project report**
    - ***(due Tuesday 27th April 2021)***