The University of Oxford

Engineering Science

4YP Project Report

- Using Machine Learning to Control Software Synthesisers -

Candidate Number: 355136

April 21, 2018

FINAL HONOUR SCHOOL OF ENG

DECLARATION OF AUTHORSHIP

You should complete this certificate. It should be bound into your fourth year project report, immediately after your title page. Three copies of the report should be submitted to the Chairman of examiners for your Honour School, c/o Clerk of the Schools, examination Schools, High Street, Oxford.

Name (in capitals): ……

College (in capitals): …… **Supervisor:** ……

Title of project (in capitals): ……

Page count (excluding risk and COSHH assessments): ……

Please tick to confirm the following:

I have read and understood the University's disciplinary regulations concerning conduct in examinations and, in particular, the regulations on plagiarism (*The University Student Handbook. The Proctors' and Assessors' Memorandum, Section 8.8*; available at https://www.ox.ac.uk/students/academic/student-handbook) ☐

I have read and understood the Education Committee's information and guidance on academic good practice and plagiarism at https://www.ox.ac.uk/students/academic/guidance/skills. ☐

The project report I am submitting is entirely my own work except where otherwise indicated. ☐

It has not been submitted, either partially or in full, for another Honour School or qualification of this University (except where the Special Regulations for the subject permit this), or for a qualification at any other institution. ☐

I have clearly indicated the presence of all material I have quoted from other sources, including any diagrams, charts, tables or graphs. ☐

I have clearly indicated the presence of all paraphrased material with appropriate references. ☐

I have acknowledged appropriately any assistance I have received in addition to that provided by my supervisor. ☐

I have not copied from the work of any other candidate. ☐

I have not used the services of any agency providing specimen, model or ghostwritten work in the preparation of this project report. (See also section 2.4 of Statute XI on University Discipline under which members of the University are prohibited from providing material of this nature for candidates in examinations at this University or elsewhere: http://www.admin.ox.ac.uk/statutes/352-051a.shtml.) ☐

The project report does not exceed 50 pages (including all diagrams, photographs, references and appendices). ☐

I agree to retain an electronic copy of this work until the publication of my final examination result, except where submission in hand-written format is permitted. ☐

I agree to make any such electronic copy available to the examiners should it be necessary to confirm my word count or to check for plagiarism. ☐

Candidate's signature: …….. **Date:** ……..

Abstract

Software Synthesisers (Soft-Synths) are computer applications which create sounds in response to musical input typically in the form of *MIDI* (*Musical Instrument Digital Interface*) messages. They are widely used in a variety of musical contexts. Typical soft-synths have hundreds or thousands of parameters which control the sound generation algorithm, allowing the user to create sounds that suit their musical needs. This results in a high dimensional, non-linear search space which the user must navigate. Research indicates that humans are bad at navigating such interfaces with serial controls, and that there is a strong link between interface design and the level of creativity that the users experience.

This work describes a novel interface designed to help users control synthesisers in a quicker and more intuitive manner. It combines three interfaces together: a traditional 'knob and slider' interface, a search space visualisation interface, and an iterative blending interface. Suitable types of synthesiser for this interface are identified, and a purpose-built synthesiser created. A small dataset of presets is created for this synthesiser.

Principal Component Analysis (PCA) combined with Histogram Equalisation is shown to be an effective method to create a low dimensional embedding of a synth preset dataset. An algorithm is developed to eliminate an undesirable phenomenon of PCA, in which the sign of the principal components may flip unexpectedly as more data points are added to the dataset. An algorithm for iteratively generating new presets based off users' preferences is developed. A Perfect/Imperfect user model is developed to numerically evaluate syntheiser interfaces. Parameter selection order is identified as a key limitation of traditional interfaces. To verify the design of the interface, it is also tested in a different context - Image Filtering.

Contents

Chapter 1

Introduction

The invention and subsequent development of the Synthesiser has been one of the main driving forces behind the evolution of popular and experimental music over the last century. With the current ubiquity of cheap, powerful computers, the barriers of entry to music making are lower than ever. Award winning albums have been produced by teenagers, entirely on laptops and smartphones [19]. Such productions commonly use Software Synthesisers (*Soft Synths*) to produce many of the sounds.

Typical Soft-Synths require lots of technical knowledge to use, and have hundreds or even thousands of parameters which control the sound generation algorithm. Many musicians lack the expertise required to synthesise sounds from scratch and thus rely heavily on *Presets*, pre-made combinations of synth parameters which produce nice sounds. Whilst there is nothing necessarily wrong with using presets, it makes it difficult for musicians to create unique and destinctive sounds. There is even a sizeable market for buying new presets for synthesisers. A conclusion can be drawn from this: more work needs to be done to make synthesisers easier to use. To quote Roger Linn, a pioneer of synthesisers, in a 2016 keynote: [13]

> "In the early days of automobiles, to own one you had to know the technical details of how it worked, and clubs of enthusiasts were formed around details such as carburettors and spark plugs. Once cars got more reliable these clubs disappeared, as it turned out they were actually just intersted in the ability to drive. I think this is the area we're in now [with synthesisers]. People are learning every little detaim rather than focusing on making music . . . The key challenge of synth design is to find what the musician really wants, and give them controls in that space."

This project aims to tackle this problem by designing a radically new interface for controlling synthesisers, and along the way hopes to find some useful insights about high-dimensional parameter spaces in general.

1.1 Key Ideas

Presets are widely used and proved to be very useful by musicians, and so should form a key part of the interface. Furthermore, the presets of a synth contain a lot of useable information. This is because the presets have been designed to highlight a diverse range of the synth's best sounds, avoiding the numerous regions of parameter space which create inaudible, or terrible sounding sounds. By analysing the timbre of the presets, rather than the timbre of the entire parameter space, a simpler, and hopefully more useful control space can be identified, whilst only requiring simple statistical techniques to be used.

Humans have impressive abilities of spatial memory, spatial reasoning, and memory of colours, and this can be exploited to increase the effectiveness of the interface. In particular, effort will be taken to maintain visual consistency between throughout the interface.

Studies show [10], that users are much better at giving a preferance between several options, rather that scoring individual options numerically. This is exploited in the Blending Interface, where in each iteration the user selects between a range of sounds, which were generated based on their previous preferences.

1.2 Contributions

- A novel interface is designed, combining a traditional synth interface with a search space visualisation interface, and an iterative blending interface.
- Suitable types of synthesiser for this interface are identified, and a purpose built synthesiser created to test the interface with.
- A small dataset of synthesiser presets is created
- Principal Component Analysis (PCA) combined with Histogram Equalisation is shown to be an effective method to create a low dimensional embedding of a synth preset dataset.
- An algorithm is developed to eliminate an undesirable phenomenon of PCA, in which the sign of the principal components may flip unexpectedly as more data points are added.
- An algorithm for iteratively generating new presets based off users' preferences is developed.
- A Perfect/Imperfect user model is developed to numerically evaluate syntheiser interfaces.
- Parameter selection order is identified as a key limitation of traditional interfaces.
- To verify the approach, the interface is tested in a different context - Image Filtering.

Chapter 2

Literature Review

2.1 Introduction to Synthesisers

Synthesisers are devices which can create a wide variety of musical sounds in response to user input, typically in the form of notes played on a keyboard. The history of synthesisers closely follows the evolution of electronics over the last century.

Early synthesisers, from 1900 - 1960, used a variety of the technologies of the time. These included steam powered electromagnetic generators, vaccum tubes, electrostatic antennae, and a variety of electro-mechanical technologies [2].

In the 1960s and 70s, technology from early analogue computers and laboratory test equipment was used to create sound. The invention of the transistor made many types of electronic circuit cheap and practical to use, allowing the creation of *modular synthesisers*. These consisted of many modules of electronics, such as voltage-controlled oscillators, filters and amplifiers (VCOs, VCFs and VCAs) and envelope generators, which were patched together with cables (see Figure 2.1). As this technology matured, many more modules were invented, such as ring-modulators, sequencers, and effects units such as reverb. Later analogue synths were fixed architecture (or semi-modular), and much more compact and user friendly, such as the *Minimoog* and the *Prophet-5*. At this time, synthesisers began to feauture commonly in popular music, both in the studio and in live situations [25].

In the 1980s, *digital syntheisers* were invented, allowing a much wider variety of synthesis techniques than is possible with analogue electronics. In particular FM synthesis defined much of the musical sound of the 1980s, and was popularised by the Yamaha *DX7* synthesiser.

Since the invention of personal computers, many popular synthesisers have been recreated digitally as *software synthesisers*, commonly known as *plug-ins*, and are widely used in modern Digital Audio Workstations (DAWs). As well as recreatations of hardware synthesisers, many novel software synthesisers have been developed solely for use on computers, making use of

the flexibility, increased computing power, and interactive display capabitilies of computers.

A much more in depth history of synthesisers, as well as deatiled explanation of all aspects of sound synthesis can be found in *The Syntheser* by Mark Vail [25].

The common methods of audio synthesis are as follows: [17]

- Wavetable Synthesis
- Synthesis using Oscillators
- Additive Synthesis
- Frequency Modulation (FM) Synthesis
- Subtractive Synthesis
- Granular Synthesis
- Physical Modelling

In addition, many synthesisers include the use of audio samples. Hybrid approaches of the above techniques are also common.

2.2 Introduction to Synthesiser Interfaces

Analogue synthesiser interfaces, whether modular or fixed architecture, usually consist of a number of modules, such as VCOs and VCAs, controlled with potentiometers, switches, and control signals from elsewhere in the synth, see Figure 2.1.

Digital synthesisers can be much more flexible in their interface, as the audio signals are

Figure 2.1: Analogue synthesiser - *System 35* (Moog)

not passing directily through the controls. Early digital synthesisers had famously un-user friendly interfaces consising of LCD screens and rows of buttons to alter the parameters, hidden in a series of menus, see Figure 2.2. As these interfaces lacked the hands-on control that musicians loved from analogue synths, many digital synths were less popular, despite being much more capable of producing a wide variety sounds. Many users found the process of programming these synths to be incredibly frustrating [25]. To make the interfaces more user friendly, most digital synths are provided with a set of presets - parameter settings that produce nice sounds - which demonstrate a wide range of the synthesiser's sounds.

Figure 2.2: Digital synthesiser - *FS1R* (Yamaha) [26]

Software synthsesiers have a wide variety of interfaces. Many follow similar design patterns to analogue synths, but with additional interactive elements such as user-drawable envelopes. An example of this is the very popular *Massive* by Native Instruments [16], shown in Figure 2.3. Other software synthesisers offer much more visual and interactive interfaces, such as the *Wavetable* synthesiser by Ableton [1], shown in Figure 2.4.

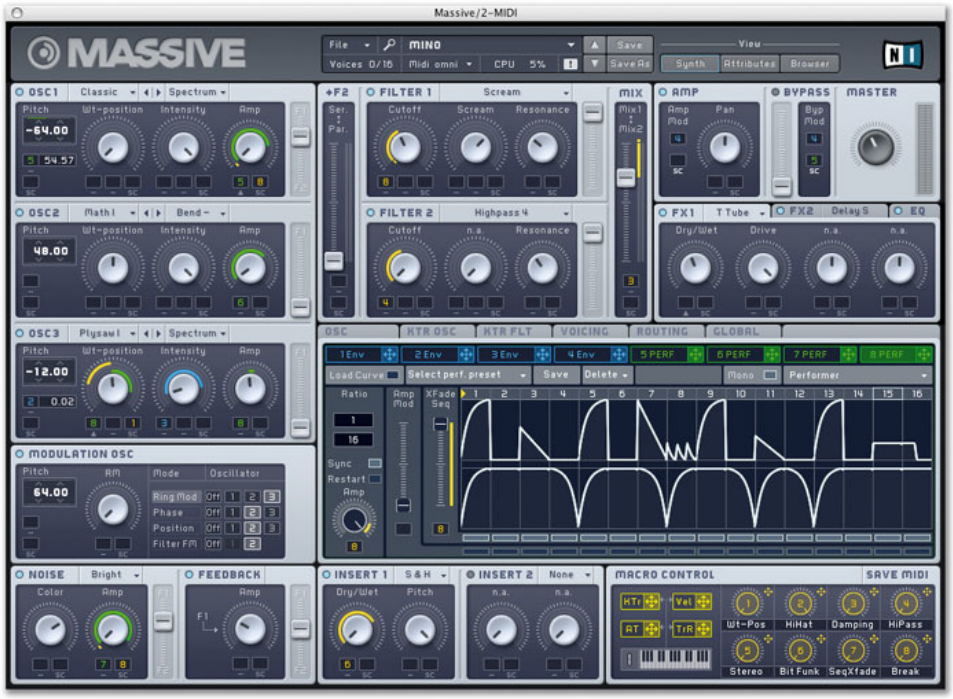


Figure 2.3: Software Synthesiser - *Massive* (Native Instruments) [16]

Figure 2.4: Software Synthesiser - *Wavetable* (Ableton)[1]

Some software synthesisers have modular designs, such as *Modular* by SoftTube [22], and some even allow users to design synthesis modules from scratch, such as *Max/MSP* and *Supercollider*.

There has been a wide range of reasearch in the Computer Music field in the use of multideminsional controllers, such as the *Myo*, *Leap Motion* and *Microsoft Kinect*, to control synthesis parameters in real time [9, 24], and research into advanced user interaction types, such as waveform blending (see *wavetable synthesis*[1] and *vector synthesis* [6]), preset blending [14], randomisation of presets, and many more [27, 23]. Several studies have applied a variety of machine learning techniques to synthesiser control, as described in the next section.

2.3 Using Machine Learning to Control Synthesisers

Many user have difficulties programming synthesisers, due to large amount of technical and creative skill necessary. There have been several previous attempts to use machine learning to make programming synthesisers easier and more expressive. In Yee King's thesis [27] Sections 4.1.1, and 4.1.2 describes the difficulty of programming synthesisers, and several of these attempts, from both academia and in commercial products.

The Google NSynth (Neural Synthesiser) project [29], develops a technique to do preset blending of sampled instruments, in a latent space of a WaveNet autoencoder rather than

operating directly on the audio recordings. This project has impressive results, but a downside is that it relies upon pre-computing a large database of audio recordings to densely sample the latent space, which are then interpolated for real time playback. Furthermore only 16 instruments were sampled.

Tone Matching is an active area of research in which a synthesiser is automatically programmed to match a reference audio recording. Common techniques involve using global optimisation techniques, such as genetic algorithms, feed-forward neural networks and gradient methods. This either requires a system which continuously monitors the output of the synthesiser, and calculates a sound-based metric to compare to the reference recording's metric, or requires a densely sampled database of metric values over the parameter space of the synthesiser. The Mel Frequency Cepstrum Coefficient (MFCC) has been identified as the best metric to use, as it is based on human perception [27].

Several works have applied interactive genetic algorithms to synthesiser programming [3, 20, 28]. Typically the genetic algorithm produces a variety of synth settings, and the user acts as the fitness function, selecting the best of these to be used for the next iteration. An advantage of using genetic algorithms, is that they can be applied to modular synthesisers, allowing the synthesise architecture to mutate. The Blending Interface has a similar iterative approach, but has been designed so that each iteration can be carried out a lot faster by the user.

Dimensionality reduction has been applied to a set of audio recordings in the 'Infinite Drum Machine'. In this work, feature vectors were calculated for a large number of real world sound recordings, and t-Distributed Stochatic Neighbour Embedding (tSNE) was used to map these sounds to 2 dimensions. This mapping was used to select samples for use in a drum machine. The Selection Interface draws inspiration from this project. It would be possible to carry about the same process for a synthesiser if first an audio recording was taken of each preset of the synthesiser. However, it is not clear exactly the best way to select which audio recording should be taken from each preset, as different presets respond very differently to the *MIDI Note*, *Velocity*, and *Note Length*. Therefore the best sampling method, to accurately characterise a synthesiser preset, in as few audio recordings as possible is an open research question.
In this work a parameter-based approach is taken, to see if similar quality results can be obtained through a statistical analysis of the parameters of the presets of a synthesiser. If

this approach is sucessful, it also has many benefits in terms of the speed of the mapping, and the amount of storage required.

The *Wekinator* [8] is an infuential application of *interactive machine learning* (also known as *active learning*) in music, in which expressive control mappings can be generated for many different types of user interaction based on a small number of user supplied training examples. This approach could be used to allow the interface designed in this report to be controlled more expressively, and make it better suited to live performance.

2.4 HCI design principles for Creative Musical Interfaces

There are four main types of parameter mapping in synthesier interfaces:[11]

- One-to-one: each control dimension is mapped to a single synthesis parameter.
- One-to-many: one control dimension is mapped to many synthesis parameters.
- Many-to-one: many control parameters affect one synth parameter.
- Many-to-many: a combination of the above (known as complex mappings).

Complex many-to-many mappings appear to be the most effective for many users, as well as being more fun and inspiring higher levels of creativity. It appears that they allow users to intuitively learn to navigate the control space. One-to-one interfaces on the other hand require the user to break the creative process into a number of subtasks, and are reported to be confusing and frustrating to use.

In [24], number of guidelines for creative musical interfaces are presented:

1. *Low dimensionality*: Control devices often have fewer parameters than synthesis engines. Given the brains limited conscious multi-tasking abilities and working memory capacity, simple controllers are preferable.
2. *Locality*, or distance preservation: Having travelled a certain distance in control space, we want that to be reflected in the distance travelled in parameter space, and ideally perceptual distance too.
3. *Revisitability*: If we return to the same point, we wish it to sound the same. The location of preset points should be stable.
4. *Continuity*: If a point is adjacent to another point on the low-dimensional surface, they should be adjacent in the high-dimensional space.
5. *Smoothness*: Continuous higher derivatives are desirable to eliminate sudden changes

in direction, this has relevance to the predictability of a control.

6. *Linearity*: When a gesture, such as a scroll, occurs it will have a certain effect on that sound, more extreme versions of this gesture should produce more of the same effect.

The Blending Interface, described in Section 5.3.1, follows all of these design guidelines, and the Selection Interface, described in Section 5.2, follows all except the *Smoothness* condition (due to the sudden jumps between presets). Both are complex many-to many interfaces.

In [24], the EARS model for creative cognition is developed, described in Figure 2.5. Detailed decriptions of the four quadrants of the model can be found in the thesis, Section 5.5.3.

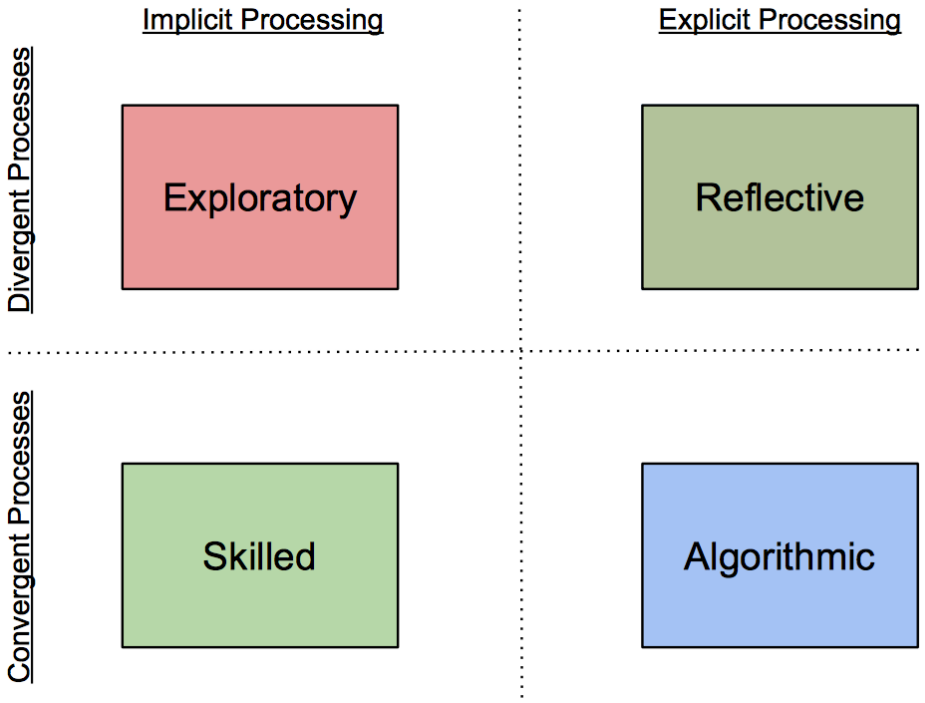


Figure 2.5: The four quadrants formed by drawing distinctions between implicit vs. explicit thinking (left/right) and divergent vs. convergent thinking (top/bottom). [24]

For different creative tasks, interfaces suited to an appropriate mode of creative cognition (Exploratory, Reflective, Skilled or Algorithmic) will be more effective, and it is important for users to be able to easily switch between these different modes. This project's interface will be evaluatied using this model in Section 6.5.

Chapter 3

Description of Synthesiser Algorithm

3.1 Description

In order to develop the synthesiser controller, it was necessary to choose a synthesiser to work with. To remove some of the complexities of interfacing with many commercial synthesisers, a purpose built synth was made to make it as easy as possible to work with the interface, whilst being representative of typical soft-synths available. The programming language *Supercollider* was used to create a 6 operator Phase Modulation (PM) synthesiser, loosely based on the Yamaha DX7, a very famous synthesiser from 1983. PM synthesis is very closely related to FM synthesis [4], and the names are often used interchangeably.

The synthesis algorithm is based around the FM7 UGen for Supercolldier [12], which implements 6 operators with independent amplitudes and frequencies, whose outputs can be used to modulate the phase of any of the operators. The implementation can be simply described by the following discrete time equation:

$$y_i[t+1] = a_i \sin(2\pi T f_i + \sum_{j=1}^{6} y_j[t] m_{ij}) \tag{3.1}$$

where T is the sampling interval, $y_i[t]$ is the output of operator i at time t, a_i and f_i are the amplitude and frequency of operator i and m_{ij} is a scalar parameter determining the level of phase modulation from operator j to operator i. This is a very flexible sound generation algorithm which can create a large number of different sounds.

The full synthesiser architecture is shown in Figure 3.1 and can be described as follows.

A *MIDI Note ON* message is received with a musical note number N and a velocity V. This triggers the sound generations process for this particular note. the note number N is converted into a base frequency F. The frequency of each operator is set using the equation:

$$f_i = F f_i^{coarse}(1 + f_i^{fine}) \tag{3.2}$$

The coarse frequency parameter f_i^{coarse} for operator i can take discrete values {0.5, 1, 2,

3, ...}, allowing the operators to set to different harmonics of the base frequency. The fine frequency parameter f_i^{coarse} can take any value in the range [0,1], and is used to detune operators away from the perfect harmonic ratios. This is the approach usually taken in typical synthesisers when setting frequencies, as the fine frequency values are typically very small.
The base frequency F can be continually modulated by the *MIDI PitchBend* control, and by a vibrato envelope (an exponential ramp from a start frequency and amplitude to an end frequency and amplitude, over a time period in the range [0,20] seconds).
The operators' amplitudes, a_i, are then continuously modulated by two low frequency oscillators (LFOs), and a modulation envelope. LFO A is a zero mean triangle wave oscillator with a frequency in the range [0, 20Hz], amplitude in the range [0, 1], and phase spread in the range [0, 1] (a parameter which allows the LFO's initial phase to be randomly varied). LFO B is a zero mean square wave oscillator with a frequency in the range [0, 20Hz], amplitude in the range [0, 1], and pulse width in the range [0, 1]. The modulation envelope is an ADSR (Attack-Decay-Sustain-Release) style envelope generator, which is triggered by the MIDI Note ON message. This can be described by the equation:

$$a_i[t] = (1 + \mathrm{LFO}^a[t] * \mathrm{lfo}_i^a) * (1 + \mathrm{LFO}^b[t] * \mathrm{lfo}_i^b) * (\mathrm{ENV}^{mod}[t] * \mathrm{env}_i^{mod} + (1 - \mathrm{env}_i^{mod})) \quad (3.3)$$

where $\mathrm{LFO}^a[t]$ and $\mathrm{LFO}^B[t]$ are the LFOs' output values at time t, $\mathrm{ENV}[t]$ is the modulation envelope's output value at time t, and lfo_i^a, lfo_i^b, and env_i^{mod} are parameters which determine how much operator i is affected by the respective signals.
After amplitude modulation with the LFOs and the modulation envelope, the operators are fed into the previously described phase modulation algorithm. The 6 outputs from this algorithm are then mixed together, and multiplied by the Amplitude Envelope to form the output signal: $Y[t] = \mathrm{ENV}^{amp}[t] * \sum_{i=1}^{6} y_i[t] * a_i^{output}$, where a_i^{output} is the Output Level parameter for operator i, and the Amplitude Envelope is another Envelope generator, similar to the Modulation Envelope.

The synthesiser has 96 parameters, as described in Table 3.1. All of the synth's parameters can be easily adjusted in real time, by sending messages through the Open Sound Control (OSC) protocol [18], a UDP based protocol for inter-application communication.

A selection of 35 presets were created for this synth, to give a selection of all the different kinds of nice sounds the synth can make, and to provide data to design the interface with.

Parameter Group	Parameter	Domain	Time/Timbre	Weighting
Phase Modulation	36 x m_{ij}	$[0, 10]$	Timbre	3
Coarse Frequency	6 x f_i^{coarse}	$\{0.5, 1, 2, 3, 4, ...\}$	Timbre	10
Fine Frequency	6 x f_i^{fine}	$[0, 1]$	Timbre	10
Output Level	6 x a_i^{output}	$[0, 1]$	Timbre	3
Modulation Envelope Amount	6 x ENV_i^{mod}	$[0, 1]$	Timbre	1
LFO A Depth	6 x LFO_i^a	$[0, 1]$	Timbre	1
LFO B Depth	6 x LFO_i^b	$[0, 1]$	Timbre	1
LFO A	Rate Amplitude Phase Spread	$[0, 20]$ Hz $[0, 1]$ $[0, 1]$	Time	3
LFO B	Rate Amplitude Pulse Width	$[0, 20]$ Hz $[0, 1]$ $[0, 1]$	Time	3
Amplitude Envelope	Attack (A) Decay (D) Sustain (S) Release (R) Curve	$[0, 10]$ seconds $[0, 10]$ seconds $[0, 1]$ $[0, 10]$ seconds $[-5, 5]$	Time	10
Modulation Envelope	Same as previous		Time	10
Miscellaneous	Vibrato Start Amt. Vibrato End Amt. Vibrato Start Freq. Vibrato End Freq. Vibrato Time Stereo Spread	$[0, 1]$ $[0, 1]$ $[0, 20]$ Hz $[0, 20]$ Hz $[0, 20]$ seconds $[0, 1]$	Time	1

Table 3.1: Synthesiser Parameters

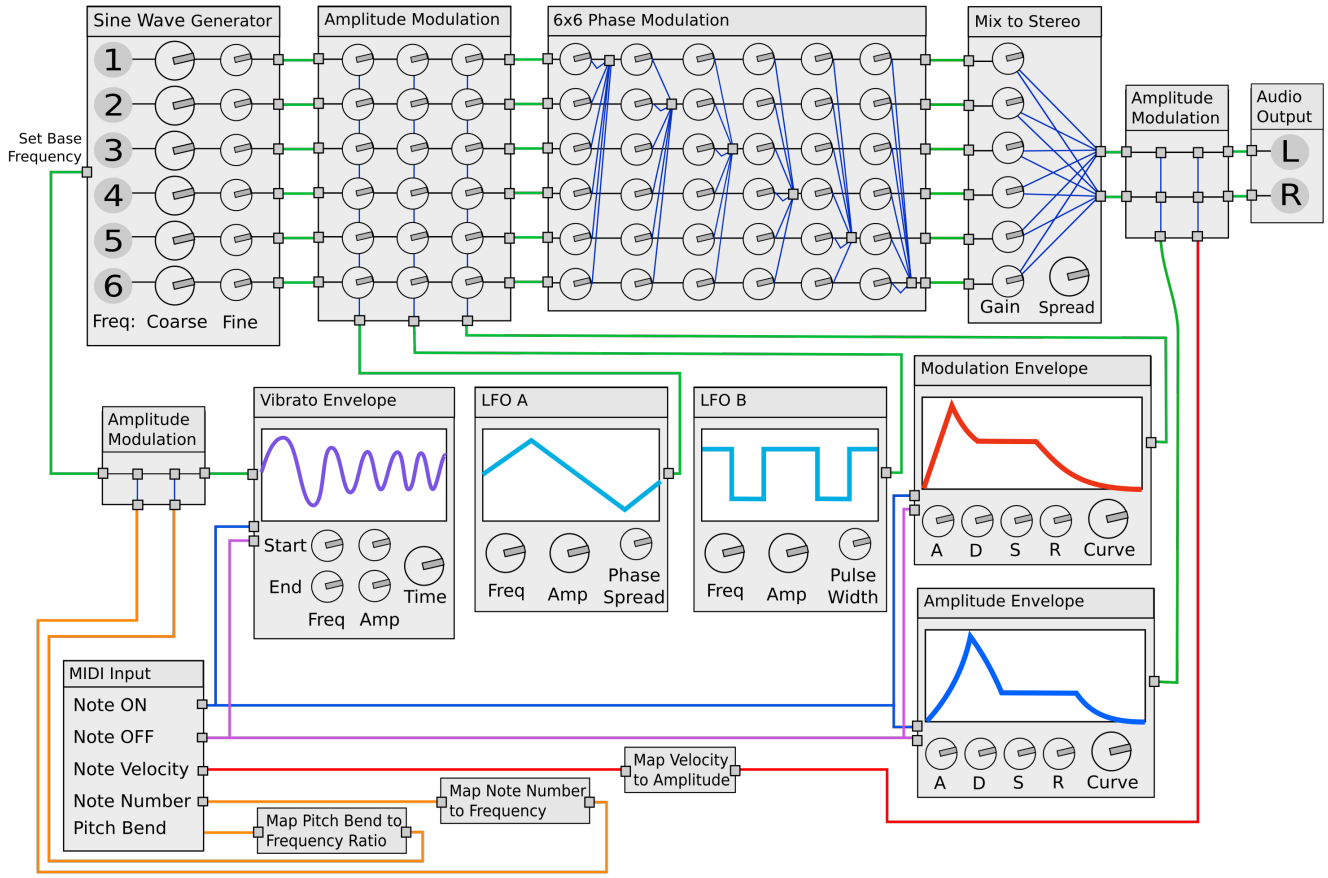


Figure 3.1: Software Synthesiser Schematic - a 6 operator Phase Modulation synthesiser, with two LFOs, and seperate envelope generators for modulation, amplitude and vibrato. Implemented in *Supercollider* with 96 parameters controlable in real time over the *OSC* protocol.

3.2 Design choices and justification

Typical commercial synthesisers have a large number of parameters[1], which can be discrete or continuous, and are usually constrained within a range. Some synthesisers have Modular or Semi-Modular architectures, which allow for a lot of flexibility, but due to the combinational explosion of number of parameter combinations, and difficulty blending between presets, this type of architecture is not addressed in the work. Some synthesisers use short audio recordings, known as samples, as part of their sound creation algorithm. This creates difficulties blending between presets, so this type of architecture will not be considered in this work. (However both of these types of synthesisers would be interesting to try as a follow up work).

This work is aimed at analogue-style synths, in which there are a medium number of parameters

[1] For example, the *Yamaha DX7* has 126 parameters, and *Omnisphere 2* has several thousand parameters

with a fixed internal routing. Effects modules, such as reverb, delay and chorus have not been included in the synthesiser, as these can easily be added to the signal chain by users. Effects modules also tend to be much easier to control than synthesisers, as they typically have fewer, and more intuitive parameters. Effects modules are typically chained together in series, with no feedback, so each effect module is a seperate transfer funtion, with no covariance between the parameters.

The synthesiser features a combination of continuous and discrete parameters, all of which are bounded. Common bounds are: $[0, 1]$, $[0, k]$ and $[-k, k]$, where $k \in \mathbb{R}^+$ (See Table 3.1).

The DX7 was chosen to as a base synth, as although it is a very powerful, it is notoriously difficult to use due to the phase modulation algorithm being very un-intuitive. An improved interface has the ability to open up FM synthesiser programming to non-expert users.

The synthesiser was designed to have access to as wide a range of sounds as possible with as few parameters as possible. For example instead of having a separate envelope generator for each oscillator (as in the Yamaha DX7), only 2 envelopes were included, with an adjustable amount per operator. The synth has 96 total parameters, compared to the 126 of the DX7, but it can replicate most of the useful sounds of a DX7. Despite aims to reduce parameter count, there are still many redundant parmaeters in most presets for this synthesiser. For example, if the amplitude of one of the LFOs is set to zero, all of the other parameters to do with the LFO will have no effect on the sound. This is a common feature with synthesisers, and presents a challenge when trying to learn useful information from the parameters.

In Yee King's thesis [27], a similar FM7 Ugen based Supercollider synthesiser was used, but no envelopes or LFOs were included, so that it was limited to producing sounds with a static timbre. Whilst this had advantages for the timbre mapping approach of the work, it resulted in a synthesiser that was not very representative of ones typically used, as envelopes are such an important part of designing sounds.

An interesting property of this synth design is that due to the 6 identical operators which can be configured in any combination, there is lots of possible permutation ambiguity: if two operators have all of their parameters switched, the sound will remain identical. This presents a challenge as two presets can have very different parameters despite having the exact same sound.

Chapter 4

Description of Full Interface

The full interface developed in this report is a combination of three interfaces. Each has the ability to vary the entire set of parameters, and produce a wide variety of sounds, but they are designed to complement each other as well as possible. The user can easily move back and forth between the different interfaces. The interface is desinged to use as little information of the synthesiser as possible, so that it can be easily applied to other synths. A video of the full interface can be viewed at (LINK), and a working demo can be downloaded from (LINK).

4.1 Traditional Interface

The Traditional Interface is designed to be representative of a typical software and hardware synthesisers. It has a knob or slider for each parameter of the synthesiser, and has interactive visualisations for some parts of the architecture (namely the envelopes and LFOs). The interface is arranged in three seperate pages, two of which are shown in Figure 4.1. If the full interface were to be extended to work with any VST, the traditional interface would be replaced by the VST's user interface.

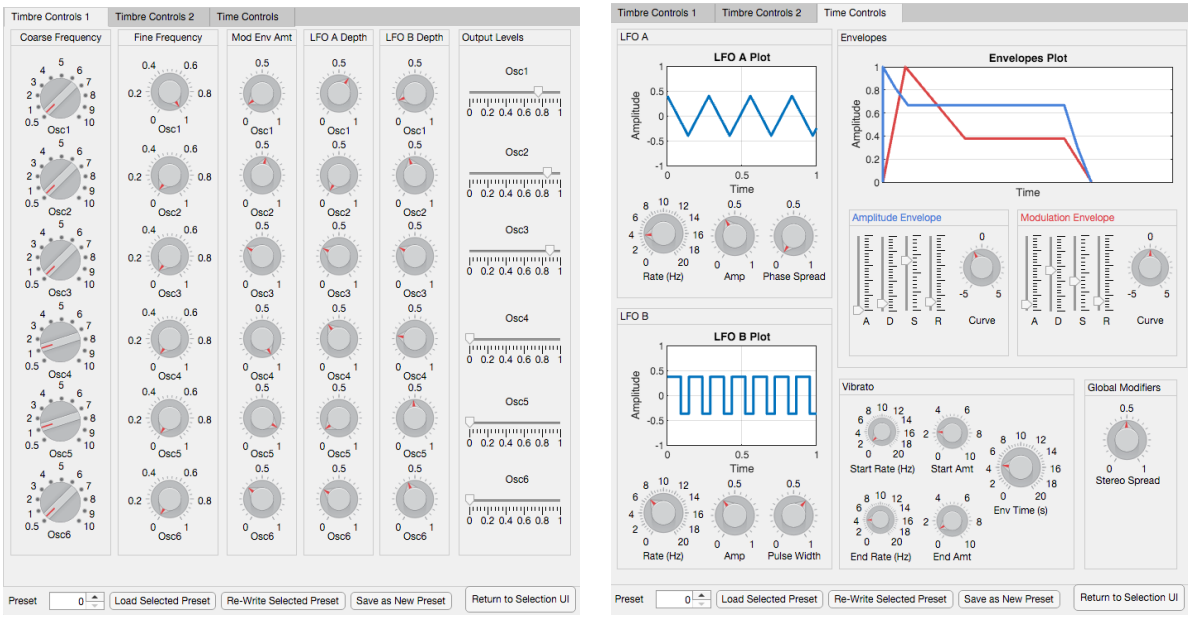


Figure 4.1: Traditional Interface - 'Timbre Controls 1' and 'Time Controls' pages

4.2 Selection Interface

Typical Soft-Synths are supplied with a large number of presets, which are usually named and arranged into categories. They are usually displayed in a list which can be searched by name, or split into category. In many soft-synths, each preset is given a set (usually 8) of *Macro Controls*, which each vary single parameter or combinations of parameters, and aim to give the user a quick way to tweak each preset.

The Selection Interface is a new approach to displaying presets, arranging the presets as cells of a 2D Voronoi diagram [5], such that similar presets are close to each other and coloured similarly. The presets can quickly be compared by moving the mouse around the diagram, and presets are selected by clicking on the cells. Once a preset is selected it can be edited by Macro Controls. The presets have been divided into a number of categories. Clicking the category buttons at the top of the interface highlights the selected category, as shown in Figure 4.2. By clicking the 'Display Mode' button, it is also possible to limit the presets displayed to just the selected category(s), displayed in a recalculated Voronoi diagram.

Both the 2D spacing of the presets, and the mapping of the macro controls are calculated automatically from the set of presets' parameter values using Principal Component Analysis.

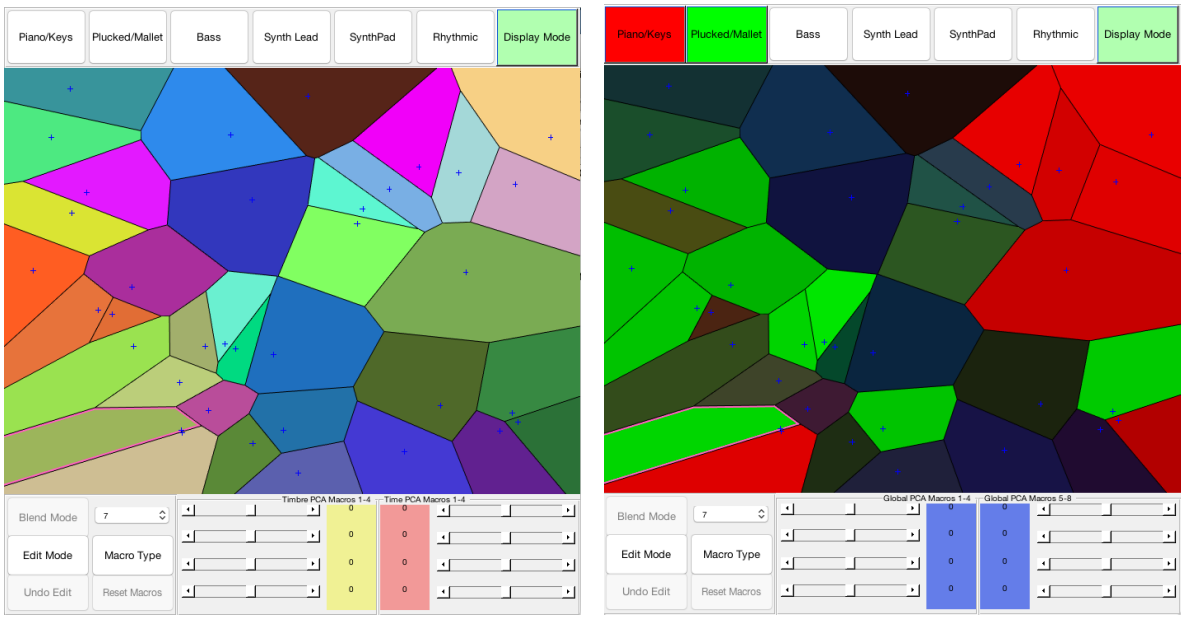


Figure 4.2: PCA Interface - Normal View, and Category Highlight View

4.3 Blending Interface

Preset Blending is a somewhat popular approach in synth interface design, where new presets are created by a linear combination of presets.[1] The Blending Interface extends this approach.

Three initial presets (A, B, C) are selected, and are placed on the corners of a triangle at the centre of the interface. As the cursor moves over the interface, a new preset is created as a weighted sum of the three presets, based on proximity to the corners. Once the user finds the optimal preset in the space, the user clicks. The preset clicked on then becomes the preset A (i.e is placed on the bottom corner of the triangle), and a new preset B and C are generated with a novel algorithm, described in Section 5.3.2.

This process is repeated as many times as desired, allowing the user to keep searching through the parameter space.

If the user clicks on the Pause on Selected Preset button, they are shown an interactive display of their past preset choices, which they can use to go back to previously selected settings and resume searching. It is also possible to select three of the previous presets, and assign them to preset A, B and C. The user has the option to freeze sections of the parameter space (see Figure 5.13) which helps the search be more fine tuned to their needs. The colours of the Blending Interface are calculated to be consistent with the Selection Interface.

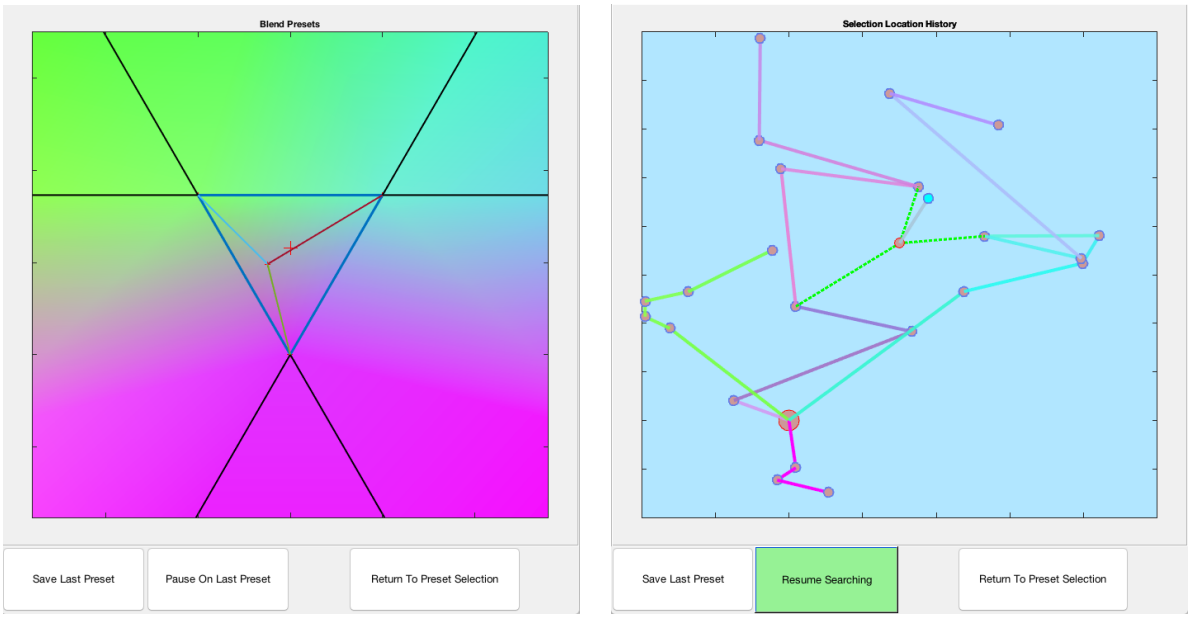


Figure 4.3: Blending Interface - Normal View, and Selection History View

[1] For example, the *Alchemy* and *FM8* soft-synths allow preset blending, and the *Nodes* object in *Max/MSP* is a commonly used implementation of preset blending.

4.4 How the Interfaces are Combined

When the application is started, the Selection Interface is displayed first. Once a preset has been selected and possibly modified with the Macro Controls, the user has the option to click the ‘Edit Mode’ button, which opens the Traditional Interface, allowing the currently selected preset to be further modified. The user can move back to the Selection Interface by clicking the ‘Return to Selection UI’ button. The varied preset is displayed on the Selection Interface with a marker, attatched to the original preset, which is positioned and coloured to be consistent with the PCA mapping (as shown in Figure 4.4). If the user wants to undo the edits made to the preset they can click the ‘Undo Edit’ button. In this way the user can move back and forth between the Blending and PCA interfaces as often as desired.

Once three presets have been selected, the user has the option of clicking the ‘Blend Mode’ button which opens the Blending Interface, and assigns the three selected presets to A, B and C. The user can then use the Blending Interface for as long as necessary, and then click the ‘Return to Preset Selection’ button. This time another marker is created, with dotted lines to the 3 initially selected presets. This marker can then be used in the same way as the cells of the voronoi diagram, i.e it can be previewed by clicking, selected by double clicking, can be edited with the Macro Controls or the Traditional Interface.

There are a set of parameter visualisations that are displayed alongside all of the presets to give the user more feedback into what changes they are making, and to aid in understanding how the synthesis algorithm works. These are shown on the left and bottom sides of the interface in Figure 4.4.

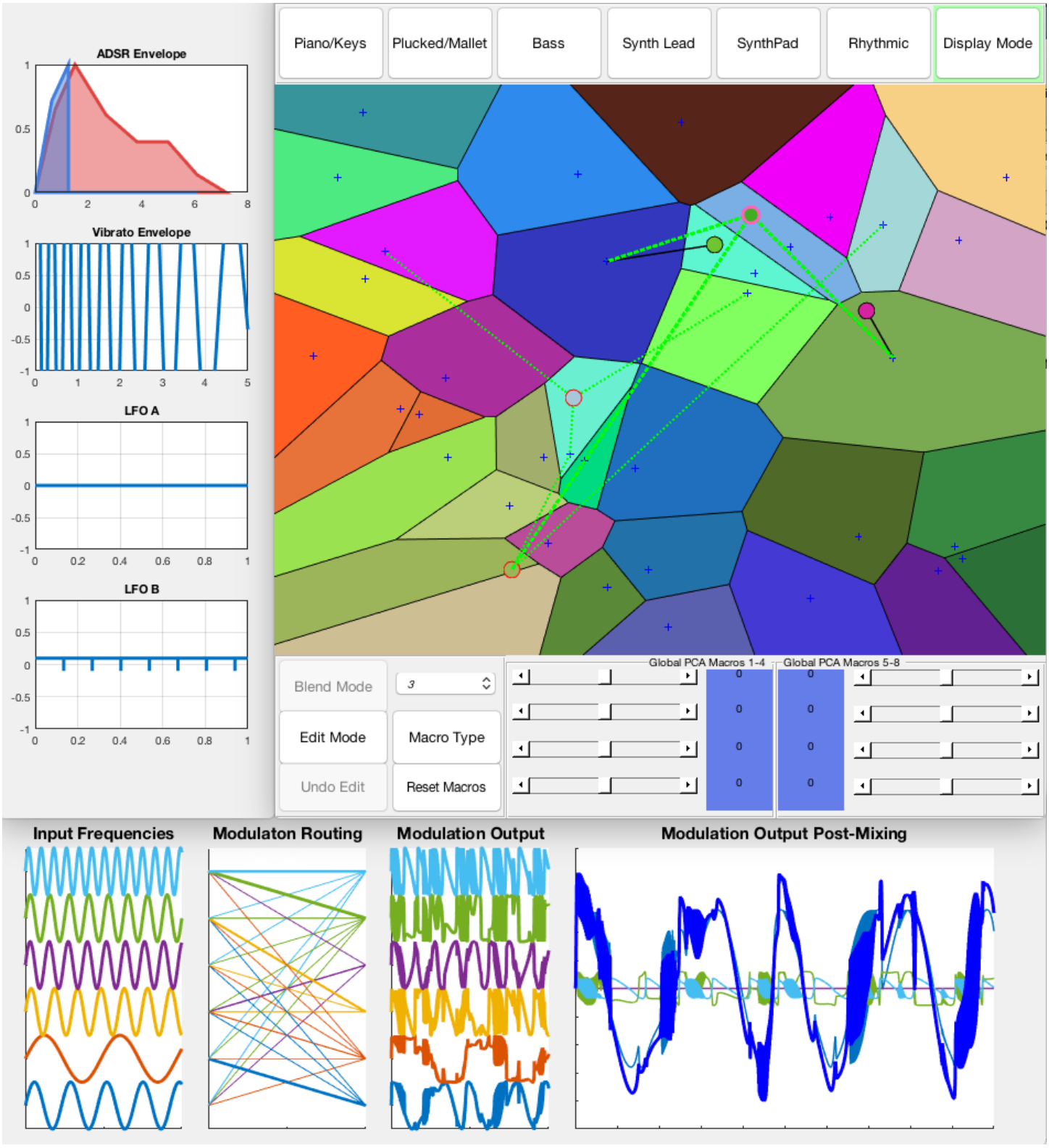


Figure 4.4: Combined Interface - PCA view. The top right corner of the interface can be swapped for the Traditional Interface and the Blending Interface. The parameter visualisations on the left and bottom sides are common to all three interfaces. The combined preset markers are shown on the Selection interface, as circles connected to dotted lines.

Chapter 5

Design and Evaluation of Interfaces

5.1 Traditional Interface

This interface was created in the MATLAB App Designer toolbox. As the parameters are varied, they are sent in real time over OSC to the synthesiser.

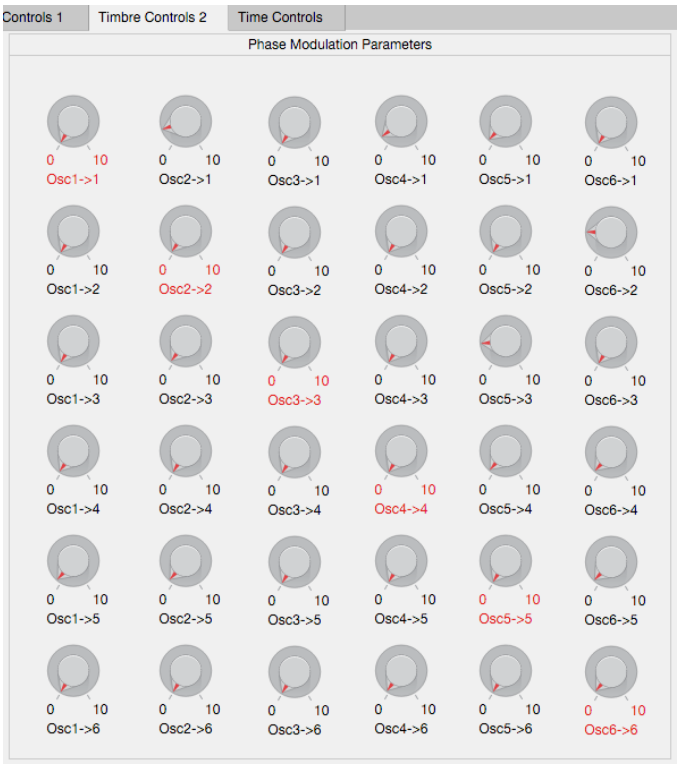


Figure 5.1: Trad. Interface - 'Timbre Controls 2'

5.1.1 Strengths

A skilled user can use knowledge of synthesis to construct any preset they can conceive of. The entire parameter space can be searched. For sectional of the synthesiser archececture that are more intuitive, such as the LFOs and envelopes, this works very well.

5.1.2 Weaknesses

Even if the user knows exactly what each parameter should be set to, it will still take several minutes to go through all of the 96 knobs and set them to the correct value. This is studied in detail in Section 6.3.1, which shows that the order at which the parameters are visited is absolutely crucial to the speed of the interface: if they are visited in a random order, the interface is very slow. Parts of the synthesis architecture are non-intuitive and fiddly to use, in particular the 36 phase modulation routing parameters shown in Figure 5.1. If the user doesn't know exactly what they want, or how to acheive a particular sound, this interface can be extremely inefficient and frustrating.

5.2 Selection Interface

This interface was created as a Matlab figure-based application. A key technical detail of this interface is the choice of mapping function from the high dimensional parameter space to the 5-dimensional XY-RGB space of the interface. Many dimensionality reduction techniques could be used for this, such as *tSNE* (used in [7]), *Linear Discriminant Analysis*, and *Autoencoders*. The simplest dimensionality reduction technique was chosen to try first: *Principal Component Analysis* (*PCA*). It is linear, non-parametric and has a closed-form solution, making it reliable and simple to implement. Other techniques could be investigated as a follow-up work.

5.2.1 Principal Component Analysis

PCA finds an optimal linear transformation of a dataset onto a set of decorrelated orthogonal basis vectors known as *principal components*. This transformation is optimal in the sense that it diagonalises the covariance matrix of the transformed dataset. The principal components are calculated as the eigenvectors of the covariance matrix $\mathbf{C}_x = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T$, where $\mathbf{X} = [\mathbf{P}_1, \mathbf{P}_2, \dots \mathbf{P}_n]^T$, $\mathbf{P}_i$ are the individual presets, and n is the number of presets.

The eigenvalues of $\mathbf{C}_x$ are the variances of each component, and the principal components are arranged in order of decreasing variance. Most of the variance of a dataset typically lies in the first few components, and hence by only using these components, dimensionality reduction is acheived [21]. The *PCA score* for a preset is calculated by projecting the preset onto a particular principal component.

PCA makes the assumption that directions with large variance correspond to directions of interest. In the case of a synthesiser preset dataset, this assumption is reasonable, as the data has high signal to noise ratio (although this may be affected by permutation ambiguity).

PCA is affected by the relative scaling of the parameters in the dataset. This can be accounted for by inversely weighting each parameter by its variance before applying PCA. However, no conclusive evidence was found to show that this is worth doing for our particular dataset.

There are several extensions to PCA including *kernel PCA* which generalises PCA to the non-linear case. *Sparse PCA* is another of such extensions, which aims to find principal components which have as few non-zero variables as possible. This is of special interest to the Macro Controls of the Selection Interface, as if each macro control only varies a subset of the parameters, it may make the Macro Controls more intuitive to users.

5.2.2 Global PCA vs Time/Timbre PCA

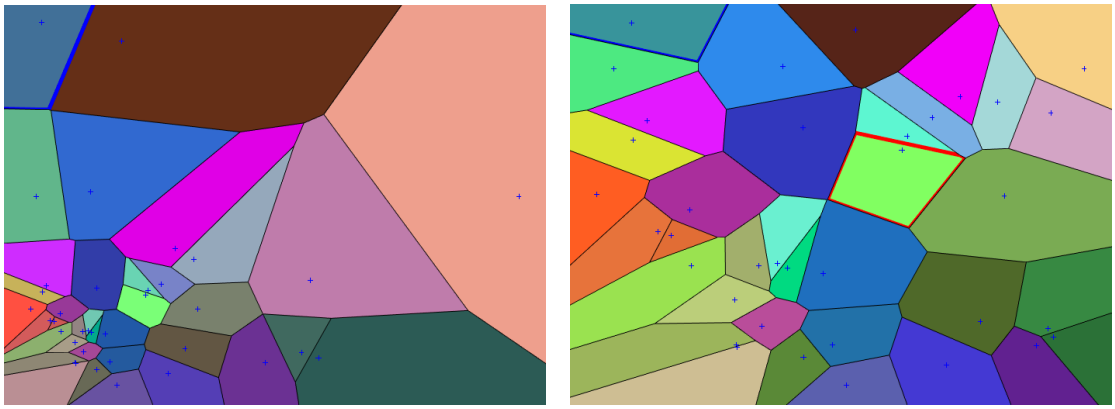
The PCA is calculated in two different ways for different sections of the interface. In *Global PCA*, each of the 36 presets' 96 parameters arranged in a 36×96 array. PCA is carried out on this array to produce the Global principal components.
In *Time/Timbre PCA*, the 96 parameters are partitioned into two sets: 72 which affect timbre, and 24 which affect variation of the sound over time (I.e all of the envelope and LFO parameters). PCA is then carried out seperately on the resulting 36×72, and 36×24 arrays to produce the Time principal components, and the Timbre principal components.
The Global principal components are used for the XY-RGB mapping of the Voronoi diagram, the Global PCA Macros, and for colouring the Blending Interface. The Time/Timbre PCA principal components are just used for the Time/Timbre Macro Controls.

The user is given the option to switch between the Global, and Time/Timbre Macro Controls. The Global controls allow a greater amount to the space to be searched, but the Time/Timbre controls can be more understandable. See Section 6.3.2 for an evaluation of the two types.
The Macro controls are the same for all presets, but centered on the currently selected preset. i.e. they allow a relative change of parameters not an absolute parameter change. A possible extension is to create a unique set of macro controls for each preset, which may make macro controls more useful, but at the sacrifice of consistency between presets.

Another possible extension to this approach is to allow parameter freezing, as in the Blending Interface (see Section 5.3.1), and have the PCA Macros automatically be recalculated based on the currently unfrozen parameters.

5.2.3 PCA + Histogram Equalisation Description

When using the Global PCA scores to calculate the XY-RGB position of each preset in the Voronoi diagram, there are some undesirable characteristics of the mapping produced. The sizes of the cells varies dramatically, and the majority of the presets usually get compressed into one of the corners, see Figure 5.2a. This is because PCA is linear, and so outliers will skew the diagram. After rescaling to $[-1, 1]$ to fit in the diagram, the inliers will be compressed to a small range. A similar effect occurs with the colour mapping, causing many of the colours to be similar to each other, minimising the dynamic range of the diagram. Research in the field of



(a) PCA alone. (b) PCA + Histogram equalisation

Figure 5.2: Selection Interface before and after Histogram Equalisation

user interface design shows that users usually associate larger icons with greater importance (REFERENCE), therefore it would be preferable for all the presets to be of a similar size, so as not to give uninteded meaning to particular presets.

To fix this issue, a variant of an image processing method known as *histogram equalisation*[30] was used, resulting in the mapping shown in Figure 5.2b.

Histogram equalisation in one dimension is shown in Figure 5.3, and described below.

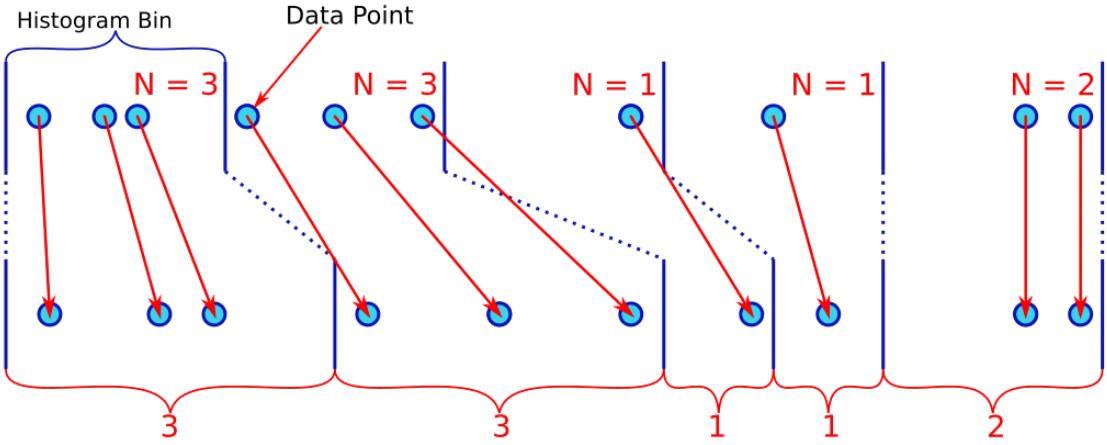


Figure 5.3: Histogram Equalisation in one dimension

A histogram is created by splitting the data into a number of eqully spaced bins (6 bins was empirically found to work well for our dataset). The bins are then rescaled, such that the bin width is equal to the number of data points in the bin).

This process is carried out for the presets in all 5 of the XY-RGB dimensions, and then each dimension is mapped to the range $[-0.95, 0.95]$, such that it will fit inside the axes of $[-1, 1]$. Figures 5.5 and 5.6, show this technique applied to the first 3 principal components.

This approach, although simple, effectively redistributes the presets. As long as none of the bins are empty, the mapping in each dimension is peicewise-linear and continuous. This means that the Combined Preset Markers can be positioned correctly on the graph by first calculating the PCA scores, then passing these scores through the mapping function. Due to the continuity of the mapping function, the Macro Controls will cause the preset markers to move smootly across the graph.

5.2.4 Demonstrations of Preset Group Clustering

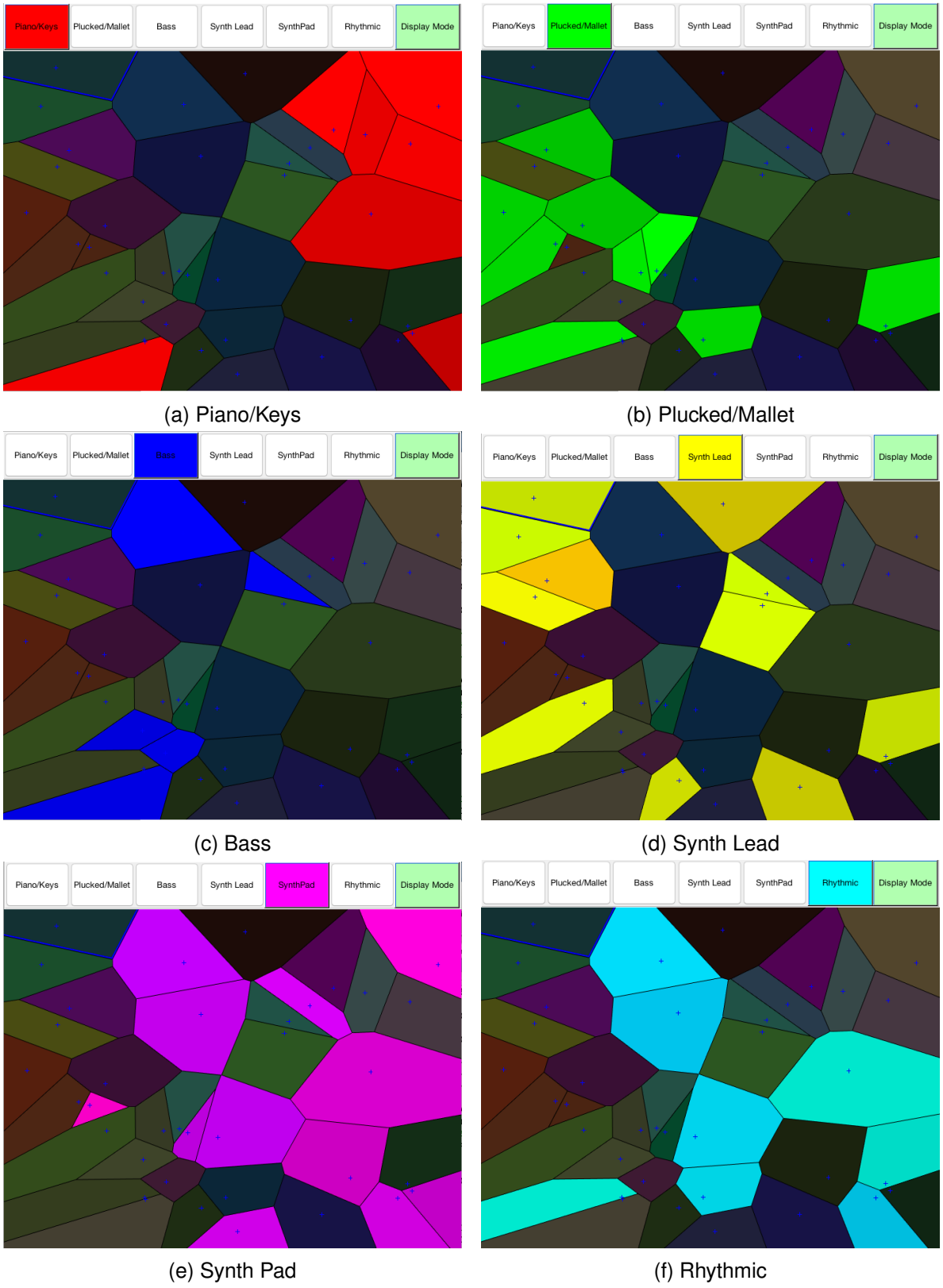
Having divided the presets into a set of (non-exlusive) categories, the clustering nature of the PCA process can be tested by viewing where the categories are placed in the diagram, see Figure 5.4. This demonstrates that the PCA process does indeed cluster the categories together, although some are better clustered than others. Part of this is due to the fact that the categories are somewhat subjective, for example the decision between Synth Lead and Synth Pad, or between Piano/Keys and Plucked/Mallet. Some of the categories also include a wide range of sounds, especially Rhythmic and Synth Lead, and so have a less consistent pattern in their parameters.

It is hard to draw many conclusions about the clustering performance, due to the small sample size of presets, and the fact that many of the presets were made by using the Blending Interface, so potentially have overly similar paramaters than if they'd been made from scratch by a person. To validate this approach more thoroughly, it would need to be applied to many real world sets of presets of pre-existing commercial synthesisers, but several technical challenges prevented this being possible during the sort timescale of this project.

5.2.5 How the PCA Mapping Scales with Number of Presets

The consistency of the PCA mapping as more presets are added is of importance to the usefulness of the interface. If the mapping radically changes every time a new preset is added, then any intuitition the user has learned about what the different components represent will be lost. This is one of the reasons for choosing PCA, as the linearity of the technique should lead to less surprising results than stochastic, non-linear techniques such as *tSNE*.

Figure 5.5 shows the variation of the preset positions for the first two principal components as the presets are added one at a time in the order of creation. After Histogram Equalisation has been applied the components are relatively stable, except at certain points where the



(a) Piano/Keys

(b) Plucked/Mallet

(c) Bass

(d) Synth Lead

(e) Synth Pad

(f) Rhythmic

Figure 5.4: Preset Categories shown on Selection Interface - For ceratin categories, such as Piano/Keys and Plucked/Mallet, the PCA + Histogram Equalisation mapping has effectively clustered the presets together. Other categories, such as Synth Lead, are less clustered.

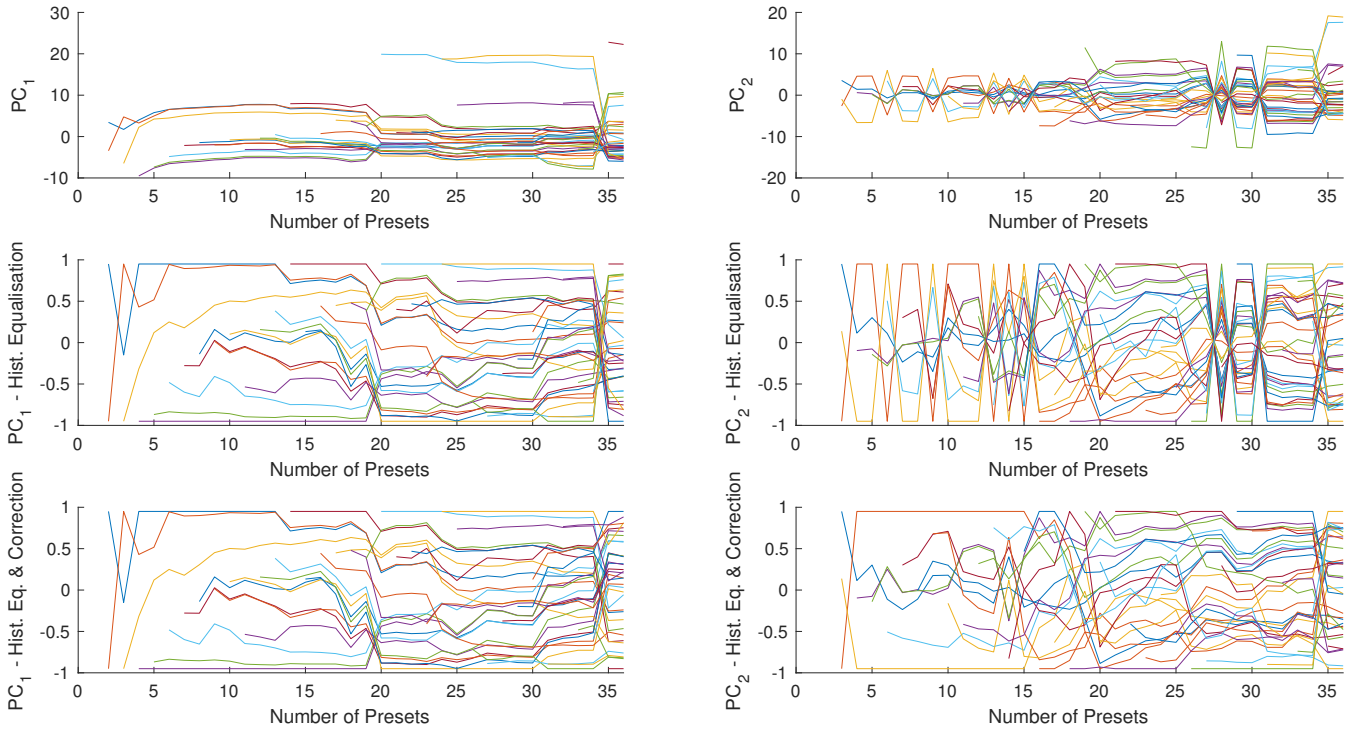


Figure 5.5: PCA stability with number of presets. These plots show the positions of the indiviual presets when mapped onto the first and second principal components, with and without Histogram Equalisation. The mapping is recalculated each time a new preset is added. The positions are relatively stable except at discrete points where the component flips. These flips can be automatically corrected, as shown in the thrid row, with details of the correction process shown in Figure 5.6.

component flips sign. This can be seen at preset 34 for PC_1, and presets 6, 9, 13, 15, 28, 31:36 for PC_2. The bottom row of the figure shows the components after this sign flipping has been corrected. This makes a dramatic improvement in the stability of the mapping, especially for PC_2. This phenomenon also occurs in $PC_{3,4,5}$ used for the RGB mapping of the presets.

As PCA is very quick to compute, the necessary sign reversals to correct the components can easily be computed using dynamic programming.

Defining PCA_i^k to be the ith principal component computed with k presets.

- Iterate from $k = 1$, to $k = N - 1$, where N is the number of presets.
- If $\|PCA_i^{k+1} - PCA_i^k)\|_1 > \|PCA_i^{k+1} - (-PCA_i^k))\|_1$, flip the sign of all remaining presets.

This algorithm is used on PC_3 in Figure 5.6. In order for this algorithm to work well, the preset mapping needs to be Histogram Normalised and centered on zero. To implement the algorithm in the interface, all that it necessary to do is keep track of which components have been flipped, and each time a new preset is added, check if $\|PCA_i^{k+1} - PCA_i^k)\|_1 > \|PCA_i^{k+1} - (-PCA_i^k))\|_1$ to determine whether to flip component i, and update the record of flipped components.

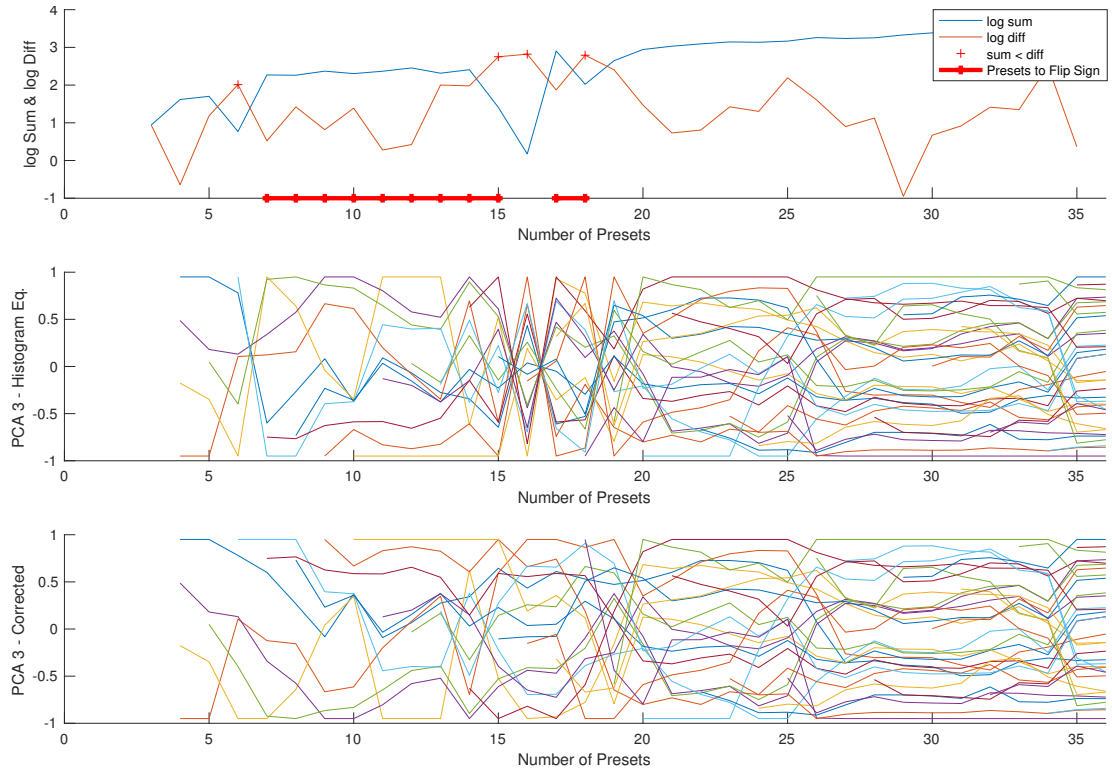
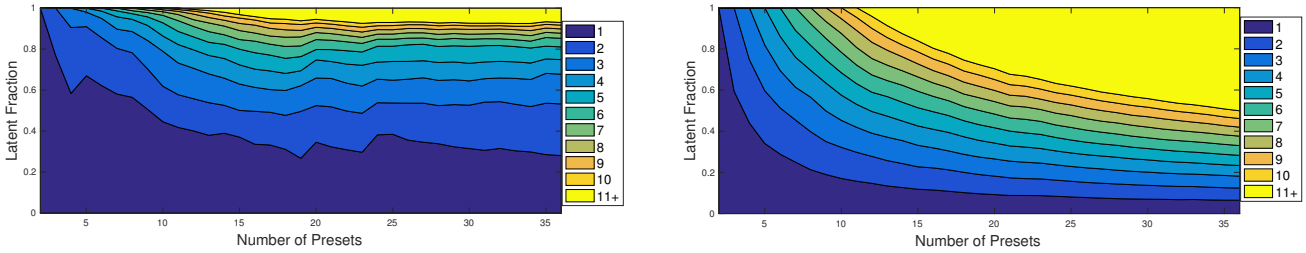


Figure 5.6: PCA Sign Flip Correction Algorithm.

The variance of each principal component can be used to measure the effectiveness of the PCA mapping. If the first few components account for most of the variance of the dataset, then the dimensionality reduction has been successful. Figure 5.7a shows how the fraction of the variance associated with each principal component changes as more presets are added. Over 50% of the variance can be accounted for by $PC_{1,2}$, over 90% by $PC_{1:10}$, and the graph assymtotes to a relatively stable state. Figure 5.7b is the same plot for a random dataset the same size as the original dataset. Each of the fractions is rougly exponentially decreasing. When all 36 presets are included, only 50% of the variance can be accounted for by $PC_{1:10}$.
It appears that synth dataset has some non-random internal structure which PCA is successfully managing to find. This suggests that PCA is an appropriate technique to use on syntheiser datasets, however testing on more synthesiser datasets is necessary to validate this result.



(a) Original Presets

(b) Randomised Presets

Figure 5.7: Variance Fraction - Colour denotes Principal Component number

5.3 Blending Interface

5.3.1 Detailed Description

This interface was created as a Matlab figure-based application. Figure 5.8 shows the the key steps of the Blending Interface's algorithm.

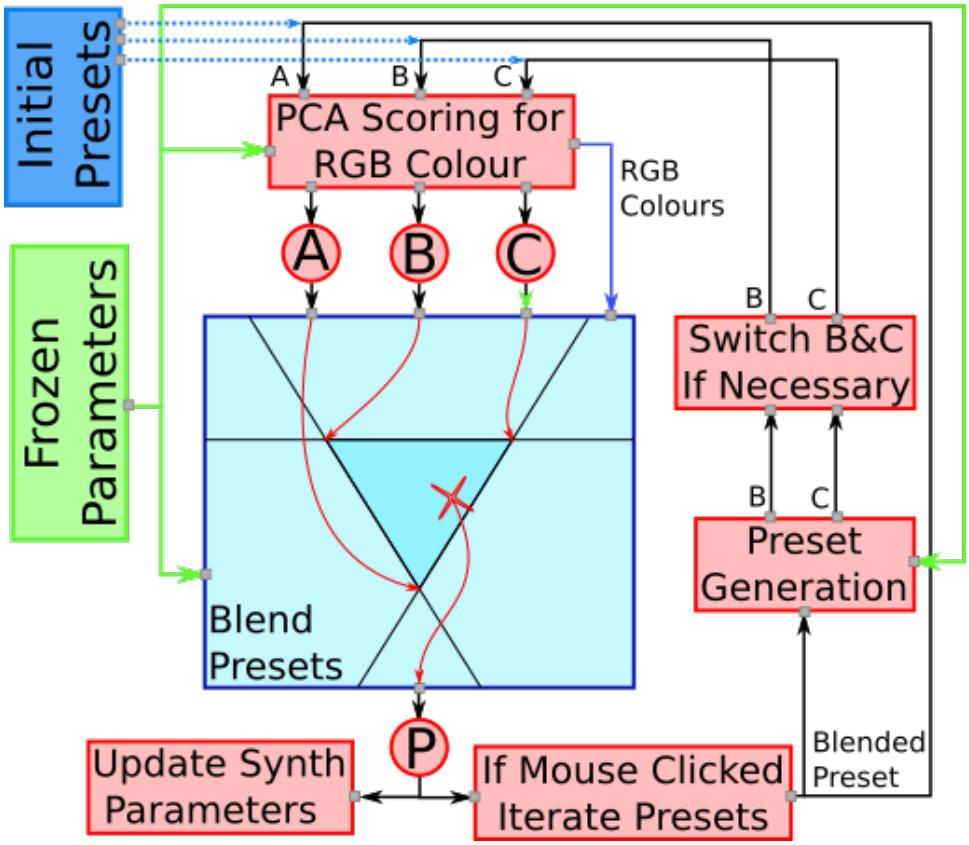


Figure 5.8: Blending Algorithm flow chart

The blending of presets is calculated as a non orthogonal vetor decomposition: [15]

$$\mathbf{P} = f(\alpha\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C}) \tag{5.1}$$

Where $\mathbf{P}$ is the blended preset, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are presets A, B and C, f is a function which applies parameter constraints, and α, β and γ are calculated from the following matrix equation:

$$[\beta, \gamma]^T = (\mathbf{M}^T\mathbf{M})\mathbf{M}^T[x, y], \qquad \alpha = 1 - (\beta + \gamma) \tag{5.2}$$

where x and y are the x and y coordinates of the current cursor position, and $\mathbf{M} = [\mathbf{b}, \mathbf{c}]$, where $\mathbf{b}$, $\mathbf{c}$ are vectors from preset location A to the B and C locations, as shown in Figure 5.9. There are two reasons for blending the presets in this way. Firstly, on an m-dimensional control surface, it is possible to linearly blend between all possible combinations of $m + 1$ presets, so

on a 2D screen, three presets should be used.[1] Secondly, this arrangment allows for locations both inside and outside of the central triangle to be used. Inside the triangle, α, β and γ are all in the range [0, 1], giving an *interpolation* between the presets. Outside of the triangle α, β and γ can be less than zero and greater than one, allowing for *extrapolation*.

The parameter constraint function, f, applies the relevant constraints to each parameter. Most parameters are continuous with an upper and lower bound, $[l, u]$. For these parameters: $f_i(P_i) = \min(\max(P_i, l), u)$, where P_i is the ith parameter of blended preset $\mathbf{P}$. For the Coarse frequency parameters, the continuous blended value is discretised back into the set $0.5, 1, 2, 3, ...$, which the desicion boundary between neighbouring number falling equidistant to the numbers, as shown in Figure 5.10.

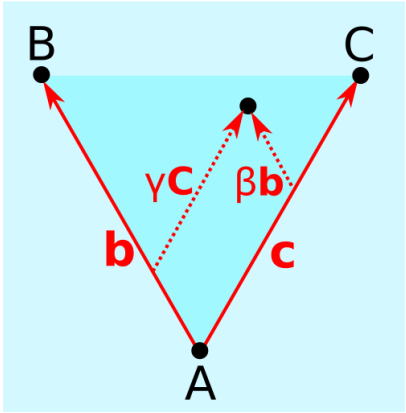


Figure 5.9: Non-orthogonal vector decomposition

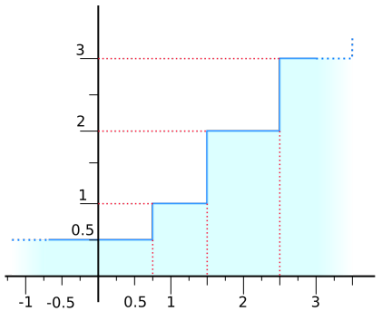


Figure 5.10: Mapping from continuous to coarse frequency

To make the iterative blending process more intuitive, and to maintain consistency between the interfaces, the PCA based RGB colouring from the Selection Interface was used. This is done by using Equation 5.1 to calculate the blended preset at each of the points marked on Figure 5.11, then calculating the PCA Scores for each of these points. $PC_{3,4,5}$ are mapped to RGB colour, and colour is linearly interpolated between the points. A more detailed colour space can be acheived by using more points, but there is an associated performance tradeoff.

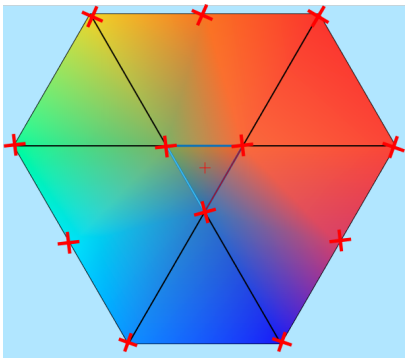


Figure 5.11: $PC_{3,4,5}$ applied to RGB colour of Blending Interface

Each time the presets are iterated and a new B and C are generated the first principal component, PC_1 is calculated for B and C, and if $PC_1^B > PC_1^C$, then presets B and C are swapped. This makes the x direction in the Blending Interface correspond to a direction of increasing PC_1. This is done to further maintain consistency between the two interfaces, as PC_1 is also used in the

[1] Some implementations of preset blending in commercial synthesisers use four presets located on the corners of a square, however in this arrangement it is not possible to reach all possible combinations.

Selection Interface for the x coordinate of each preset.

The Selection History display allows the user to browse their previously selected presets, by clicking on nodes of the graph. Each time presets are iterated, the vector from A to the selelected coordinates, $\mathbf{p}_i$ is recorded. The graph is constucted by joining these vectors head-to-tail, as shown in Figure 5.12. The colours of each edge of the graph are chosen based on $PC_{3,4,5}$ of the presets. See Figure, 4.3 for a typical example of the Selection History plot.

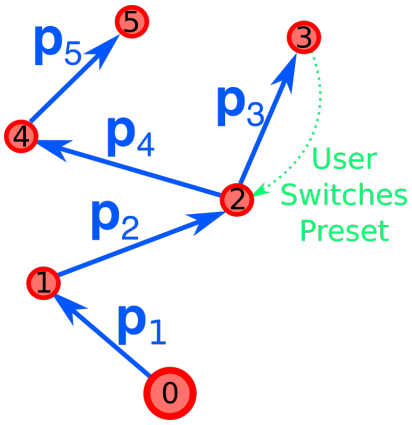


Figure 5.12: Selection History plot construction

It may be possible to use some of the information contained in the graph (if the data from many users is collected), as a way to refine the preset generation algorithm. If the generation algorithm is producing good presets, then most of the choices should be somewhere between A, B and C on the interface. If the algorithm is producing bad presets, most of the choices will be very close to, or below A.

As part of the Blending Interface, the user has the option to freeze sections of the parameter space. The parameters are divided into Time and Timbre parameters, and then subdivided into a total of 12 smaller categories, see Figure 5.13. This makes the Blending Interface more useful, as it allows users with knowledge of synthesisers to fine-tune the blending process to their needs. This tends to lead to more useful generated presets, as it reduces the dimensionality of the search space.

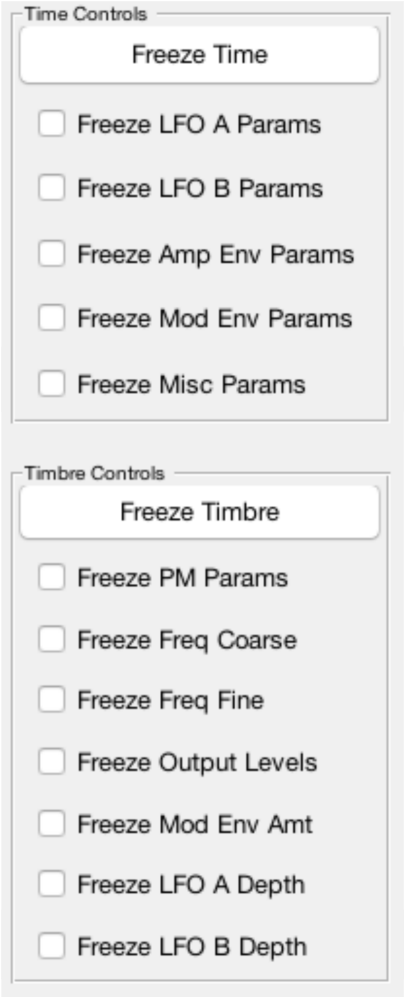


Figure 5.13: Parameter Freezing

5.3.2 Development / Verification of Preset Generation Algorithm

THIS SECTION NEEDS WRITING! Give background to Active Learning. Mention possibility of using Bayesian Optimisation, but aiming for simpler approach.

Bayesian Optimisation. - Discuss how this is a promising technique in this field, for example preference gallery interface, but due to large amount of training data neccesary, and bad performance in high dimensions it wont be used.

Mention preferance gallery study, and how users are much better at giving comparisons than scoring individual presets. Hence Blending Interface, as allows comparison between 3 presets, and a plane of settings in-between them.

5.4 Investigion of Permutation Ambiguity

Due to the equivilance of the 6 operators of the synth arcitecture, there is the possibility of permutation ambiguity in the dataset, as described in Section 3.2. However it may be possible to correct for this using knowledge of the synth architecture. Typically the operators with the loudest Output Level parameters act as *modulators* and the operators with lower output levels act as *carriers* (as defined in [27], Section 3.1.1). Therefore the dataset can be permutation-alligned by permuting each preset, such that the Output Level parameteres are in decreasing order.

To determine the sucess of this approach, the original dataset was compared to the permuted dataset, and a randomly permuted dataset , using the following cost function:

$$C = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathrm{error}(\mathbf{P}_i, \mathbf{P}_j) \tag{5.3}$$

where N is the number of presets, and error is the error metric defined in Section 6.2 with only the Timbre parameters included (As only these exhibit permutation ambiguity).

The results of this evaluluation is shown in Table 5.1. The permutation-alligned dataset has a similar cost to the original dataset, and a 15% lower cost than the randomly permuted cost. It is worth noting that when creating the original dataset, most presets were designed to minimise permutation ambiguity, and so exhibit lower levels than a typical synth preset dataset might. The interface was tested after permutation-alligning the presets, which seemed to improve the usefulness of the macro controls, and made the blending interface create more useful sounds. Furthermore the category clustering perfomance wasn't degraded. Based on the results, permutation-allignment seems to be a worthwhile endevour, as it gives a sigificant improvement over randomly permutated dataset, and gives potential improvements to the usefulness of the interface. More work needs to be done to thoroughly evaluate this technique, and to determine

if more knowledge of the synth architecture can make the process more effective.

	Original Dataset	Permutation-Alligned Dataset	Randomly Permuted Dataset
Cost $\times 10^4$	2.427	2.452	2.819 (averaged over 10 runs)

Table 5.1: Permutation Ambiguity Evaluation Results

Chapter 6

Numerical Evaluation of Interfaces

Due to the subjectivity of interface design, and as the three interfaces have quite different purposes, there are not many obvious metrics to use to compare the interfaces with each other. However, the key challenge is to make the combined interface better than the traditional interface alone. MORE WRITING - PRIOR WORK? MEASURE OF REGRET¿?

6.1 Perfect/Imperfect User Model

A way of evaluating the performance of the interface is to simulate a Perfect User, and an Imperfect User carying out a range of tasks. These simulated users must use the interface to move from initial presets to goal presets. This analysis won't account for any creativity based goals of the interface, but will help evaluate some of the practical considerations of the interface. (BAD WORDING¡ WORK ON THIS SECTION).

The Perfect User has perfect knowledge of the synth and the interface, so can always choose the optimum position for a particular knob, or choose the optimum blend of presets in the Blending Interface.

The Imperfect User has imperfect knowledge of the synth and the interface, so chooses interactions similarly to the Perfect User, but misses by a certain amount: $\Delta P_i = \Delta P_i^0 * (1 + \varepsilon)$, where ΔP_i^0 is the Perfect parameter change, and ε is a zero-mean Gaussian random variable with variance σ^2. Various values of σ will be considered to account for the varying skill levels of users.

A key consideration for the Traditional Interface is the order which parameters are varied. In the Perfect Order, parameters are visited in decreasing order of importance. In the Random Order, parameters are visited in a completely random order. Real users most likely operate somewhere in the middle of these regimes. (REFERENCE FOR THIS¿???)

6.2 Error Metric

A parameter based error metric is used for these tasks due to the complexity of using a sound based error metric (AS DESCRIBED IN SECTION - MAYBE DESCRIBE MORE HERE). For each parameter, p_i, a weighted L_1 norm is used. The parameters are weighted based on an approximate measure of importance, and normalised by their parameter ranges. The total error is calculated as:

$$E = \sum_{i=1}^{N} \frac{w_i}{r_i} \|p_i - p_i^{goal}\|_1 \tag{6.1}$$

where w_i is the scalar importance weighting for the ith parameter, and r_i is the range of values that parameter i can take, see Table 3.1.

The L_1 norm was chosen over the L_2 norm so that a few large parameter errors don't dominate the error metric and as no gradients are necessary. This is by no means a perfect metric for comparing synth settings, but it is good enough for our purposes.

6.3 Comparison of Isolated Interfaces

6.3.1 Traditional Interface

Perfect/Imperfect user tests were carrried out on the Blending Interface, using each of the 36 presets as a goal preset, and setting the starting preset to the closest preset to the goal preset. The results of these tests are shown in Figure 6.1. In each of the plots, the cyan lines are individual instances of the Perfect Order test. The blue lines are individual instances of the Random Order test. The solid red line is the mean of the Perfect Order tests, and the red dotted line is the mean of the Random Order tests.

For low values of σ, the Perfect Order tests folllows a smooth, approximately exponentially decreasing, curve. For high values of σ, and when the Random Order is used, the error decreases in a much less

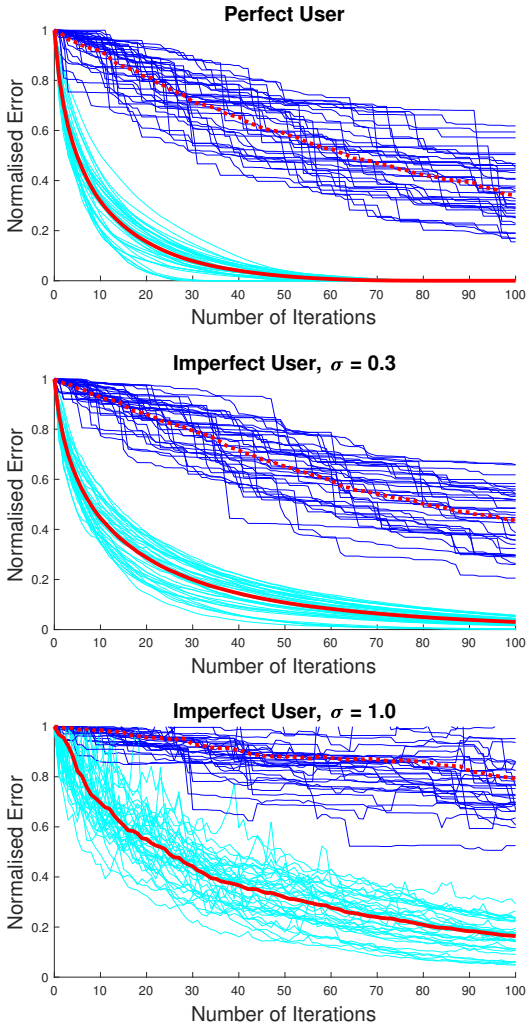


Figure 6.1: Perfect/Imperfect user tests for Traditional Interface

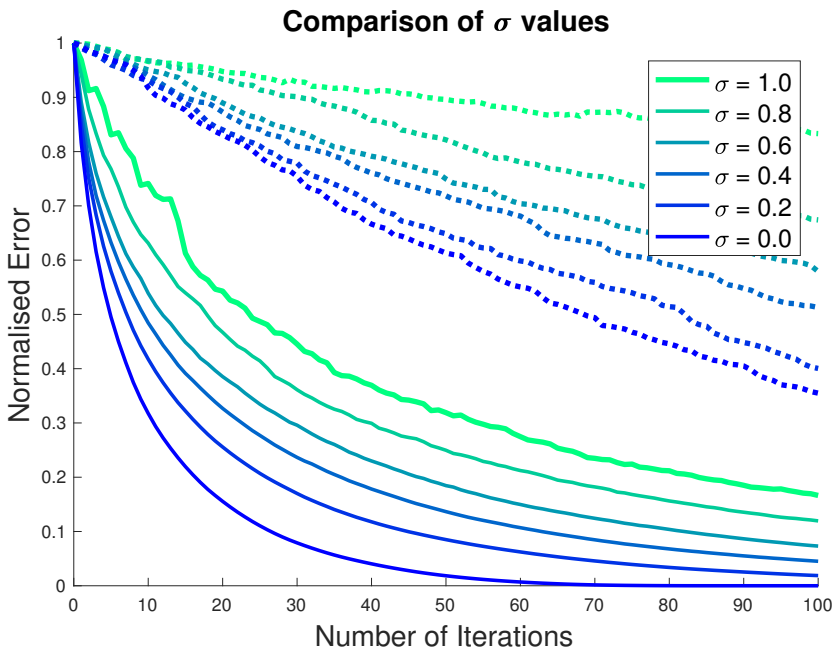


Figure 6.2: Perfect/Imperfect user tests for Traditional Interface. Perfect Order shown with solid lines, Random Order shown with dotted lines. The Parameter order has a much more significant effect than the Imperfectness of the user.

smooth manner. In all cases the Perfect Order gives a substantially faster convergence time. A comparison of the mean error for various values of σ is shown in Figure 6.2. These results suggest that the main limitation when using a Traditional Interface is not the accuracy with which the parameters are set, but the choice of parameter.

6.3.2 Selection Interface

A similar evaluation was carried out with the selection interface. Again all 36 presets were used as goal presets, with the closest preset used each time as the starting point. For each test, the PCA values were calculated for the remaining 35 presets, simulating the situation where the PCA Macros were being used to find a brand new preset. As before, two parameter orders were tested: the Macro Controls were visited cyclically in order of PCA number, or visited in a random order. For each iteration of the test, the MATLAB 'Patternsearch' search algorithm was used to minimise the cost function for a particular Macro Control. (WHY WAS THIS USED?)
A key question with the interface is whether the 'Global' or 'Time/Timbre' version of the PCA Macro Controls should be used. The results for a test with the combination of both sets of macros is shown in Figure 6.3, and a comparison of the mean results for the different macro configurations is shown in Figure 6.4.

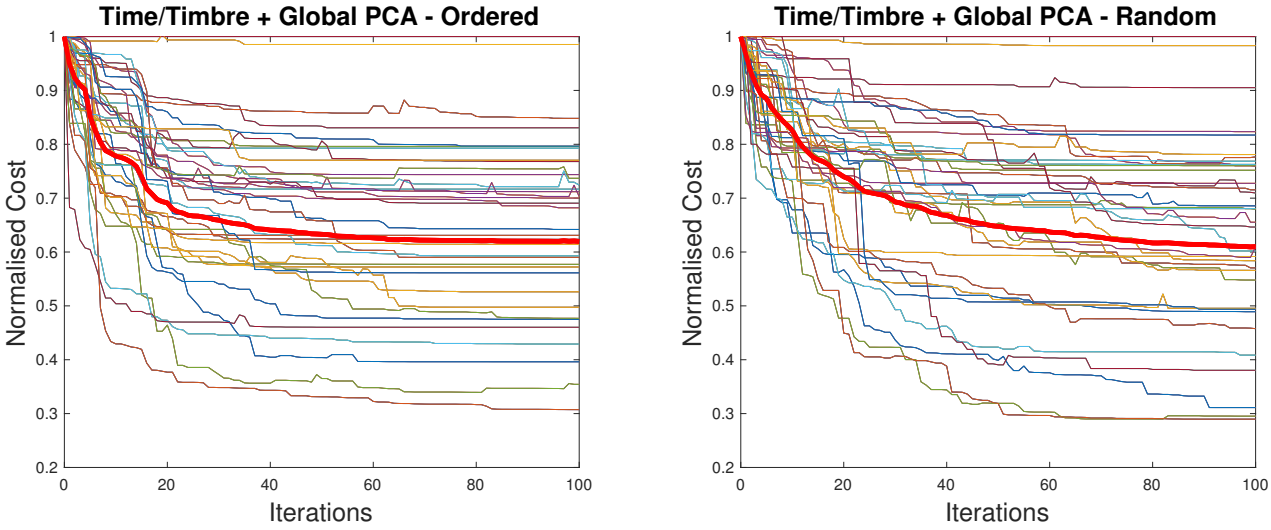


Figure 6.3: Ordered/Random user tests for Selection Interface

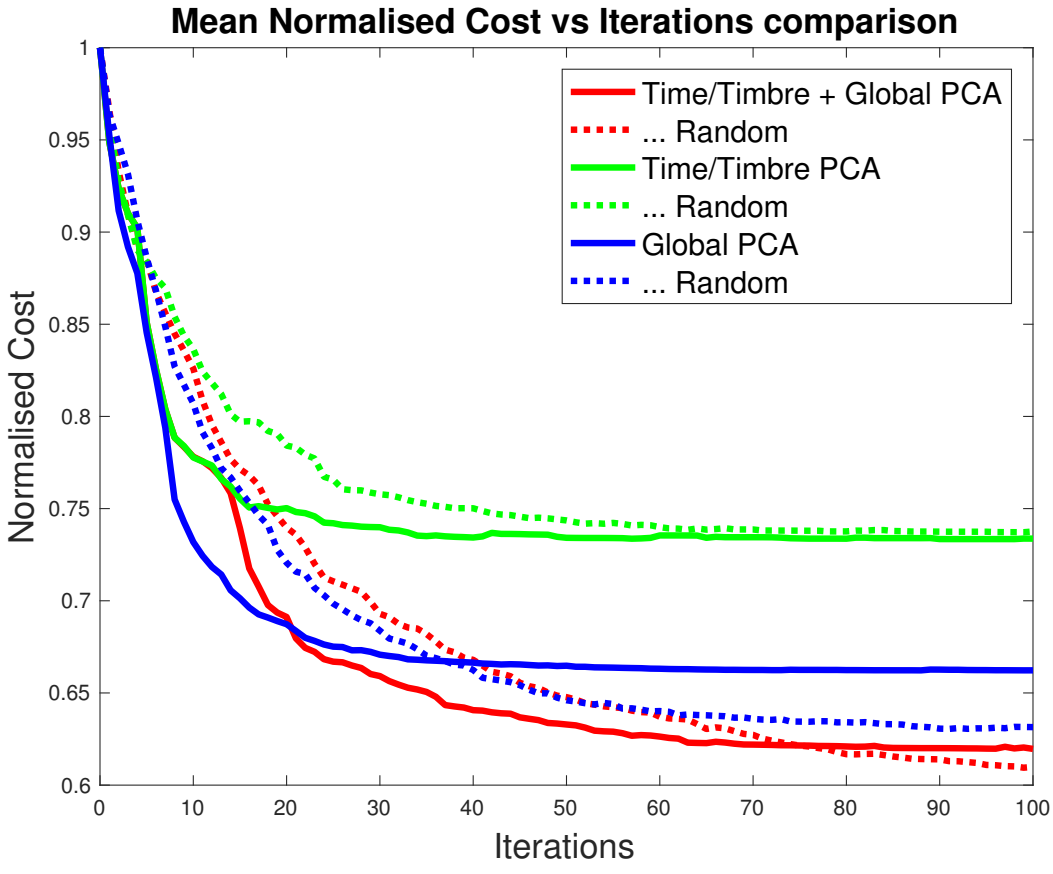


Figure 6.4: Comparison of Macro Types - All of the Macro Controls have a similar initial rate of cost reduction, but the Global Macro Controls have a better steady state value than the Time/Timbre Macro Controls.

Some points to note from these results:

As shown in Figure 6.3, for different preset goals, the effectiveness of the PCA Controls varies significantly. In the best case the error drops very rapidly, falling 50% in the first 10 iterations. In the worst case the error doesn't decrease at all. This suggests that, for this particlar task at least, having the same Macro Controls for each preset may not be ideal, and so a new approach which in some way optimises the Macro Controls per preset would be worth pursuing. However, as previously stated in Section 5.2.2, this comes at the cost of the user being less familiar with what the Macro Controls are actually doing.

In all cases the mean error converges to a non-zero steady state value. This is due to the limited degrees of freedom when using the Macro Controls. This means that the Macro Controls alone cannot be used to perfectly match a goal preset.

When the Global Macro Controls are being used, after a large number of iterations, the random order manages to decrease the error past the cyclic order. It is likely that this is due to the cyclic order getting stuck in a local optimum due to the naive 'one parameter at a time' (BETTER NAME FOR THIS) search algorithm.

Finally, each of the combinations of controls has a similar initial rate of error reduction, but the 'Global' Macro Controls appear to have a significant advantage over the 'Time/Timbre' Macro Controls over larger numbers of iterations. This is likely due to the fact that the Global Macro Controls have access to principal components 1 to 8, whereas the 'Time/Timbre' Macro Controls on have access to two sets of principal components 1 to 4. The combination of both sets of Macro Controls gives the lowest mean error overall, which is unsurprising as it has the most degrees of freedom, and it has a comparable error reduction rate as the 'Global' Controls.

Based on this it is concluded that it is worth keeping both sets of macro controls for the user to be able to select between, especially as the Time/Timbre Macro Controls may lead to more semantically meaningful controls due to the seperation of Time and Timbre. (JUSTIFY THIS MORE)

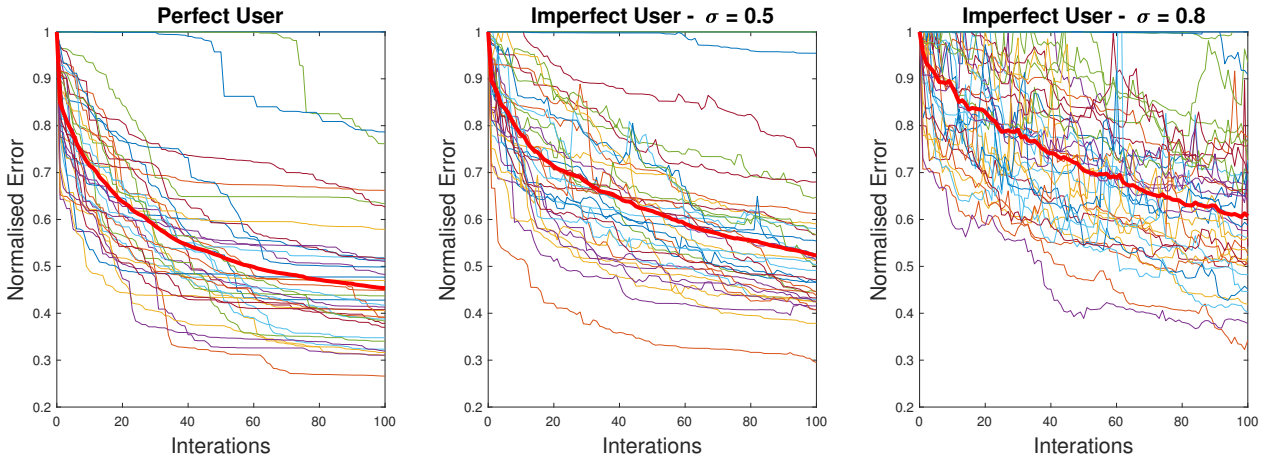


Figure 6.5: Perfect/Imperfect User tests for Blending Interface

6.3.3 Blending Interface

The Blending interface was evaluated in a similar manner. For each of the 36 possible goal presets, the closest three presets to it were found, and these used as preset A, B and C in the blending algorithm. MATLAB's Patternsearch was again used in each iteration to optimise the position of the mouse in the Blending Interface to minimise the parameter based error metric.

Results of this test are shown in Figure 6.5 On average, there is n rapid decrease in error during the first iteration, followed by an approximately expenential decrease in error. As in the Selection Interface, there are some tests in which the error barely decreases at all, and some in which the error decreases very quickly, however the variance of the different tests is less than in the PCA interface. For the Perfect User, error is monotoically decreasing, but this is not always the case for the imperfect user, and for real users of the system(EVIDENCE?). This reinforces the need for the Selection History Plot (See Section 5.3.1), which allows users to select several of the local minima in the error plot to resume blending with. (MAYBE MORE DETAIL)

6.4 Comparison of Interfaces

A comparison of the test results from all three interfaces is shown in Figure 6.6. COULD DO A TABLE OF INITIAL RATE OF ERROR REDUCTION

The fastest converging test is the Perfect User with Perfect Order on the Traditional Interface, and the slowest converging test is the Imperfect User with Random Order on the Tradtional

interface. Interestingly, the order at which the parameters are visited has a much more significant affect than the perfectness of the user. This is due to the Imperfect user model having a zero mean error, so after visiting a parameter several times, the value will converge to the correct value. This result demonstrates that the Traditional Interface can simultaneously be the best possible, and worst possible interface to control a synth with, depending on the experience level of the user, and the intuitiveness of the parameters.

The initial error reduction rate of the Blending Interface is very fast, comparable with the Perfect User of the Traditional interface. The main reason for the initial speed of the Blending Interface is that in the first iteration, it blends between 3 presets, all close to the goal preset, allowing the first iteration to be a lot less random than subsequent steps. (EXPLAIN BETTER)
A key advantage of the Blending Interface is that it removes the need for the user to choose which order to alter the parameters which, as previously described, is the main factor in speed of the Traditional Interface. (MORE DETAIL?)
Over a larger number of iterations, Blending Interface is supassed by the Traditinonal interface with random order. This suggests that the preset generation algoithm is not optimal, as after a while it becomes faster to just sequentially offer the user a random parameter. However, as the user has the option to freeze parameters, and the option to switch to the other interfaces at any point, this may not be an issue.
Before converging to their steady state value, the macro controls have quite a fast rate of error reduction - halfway betwen the Perfect User with Perfect or Random Order. This can be partially explained as the Macro Contols allow all of the synthesiser parameters to be varied at once. This also suggests that, as the PCA was carried out on a set of presets, it is likely that the principal components are pointed in a more useful direction than if they'd been selected randomly. (WRITE BETTER)
These results suggest that a good workflow for editing synthesis parameters is as follows:

- Find closest 3 presets to desired sound
- Use Macro Controls to refine these presets further ($\approx$ 5 iterations per preset)
- Use Blending Interface to combine these presets together and further refine ($\approx$ 10 iterations)
- Use Traditional interface for final refining of preset. ($\approx$ 20 iterations)

The full Interface allows such a process to be carried out, and allows the user to swicth back and forth between the interfaces as often as necessary.

As these tests have only been carried out on one set of presets for one particular synthesiser, an important follow-up work to this project is to validate these results on other synthesisers. As these tests are purely parameter based, the tests could potentially be carried out before doing a full integration of the interface with other synthesisers, as only the set of presets for each synth are needed.

6.5 EARS-model based evaluation of interface

In Table 6.1, the three interfaces are evaluated in terms of the EARS model from Section 2.4 [24]. A conclusion which can be drawn is that the interfaces enables Exploratory, Algorithmic, and Reflective process well, but is not suited to skilled interactions. One of the interfaces could be redisigned with this in mind, or a fourth interface could be introduced. It should be designed for use with multidimensional controllers, and involve moving through a parameter space consisting solely of the nice sounds of the synthesiser. An Interactive Machine Learning system such as the *Wekinator* [8] would be well suited to this task, however more work needs to be done to integrate this approach with the other 3 interfaces.

Interface	Exploratory	Algorithmic	Reflective	Skilled	Comments
Traditional	Maybe	Yes	No	No	Can be used for exploratory and algorithmic depending on the skill level of the user
Selection	Yes	No	Yes	No	Exploration by browsing and varying presets. Reflective as easy comparison of presets with combined and varied presets.
Blending	Yes	Maybe	Yes	Maybe	Well suited to exploratory, due to simple interface and preset generation. May be used algorithmically due to parameter freezing. The Selection History graph enables reflective comparison and combination. May be used as a skilled strategy, but is not optimised for this approach. Too low dimensional and unpredictable.

Table 6.1: EARS Model evaluation of synthesiser interfaces

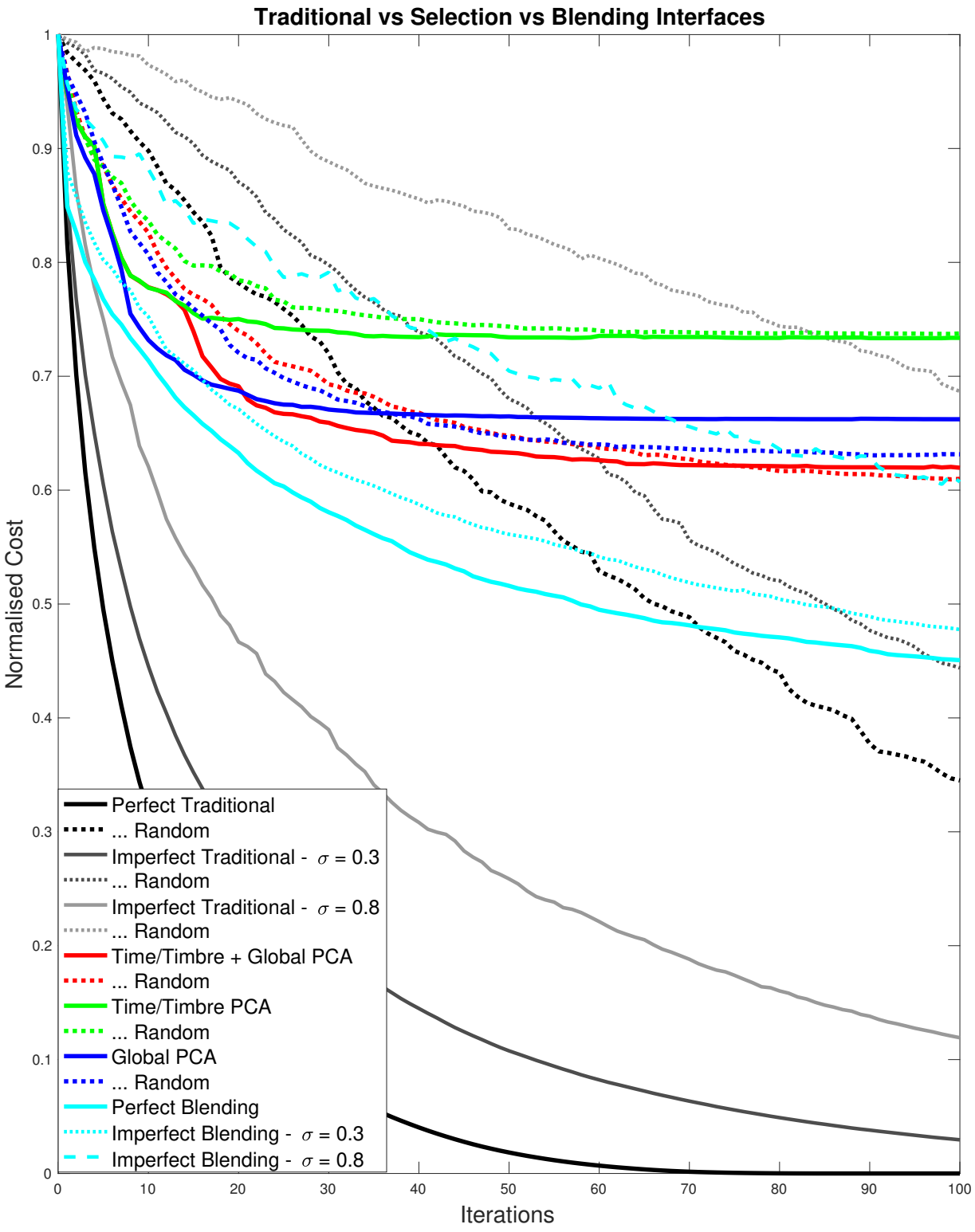


Figure 6.6: Comparison of Interfaces - The Blending Interface, and Selection Interface both have initial rates of error reduction faster than the traditional interface with random order (WRITE MORE DESCRIPTION HERE)

Chapter 7

Conclusion

The interface proposed in this project has many benefits over a traditional synthesiser interface, as has been designed following design heuristics from the fields of Human Computer Interaction and Creative Cognition. Based on simulated user studies, the interface has at least as good performance as a traditional interface when carrying out search based tasks, and based on real user feedback it has many advantages in terms of creativity (If don't have interviews CANT INCLUDE).

Synthesiser preset datasets are not random, and information can definitely be learned from them. PCA combined with Histogram Equalisation has been identified as a useful dimensionality reduction tool for synthesiser parameters, and several preset generation algorithms have been developed and tested.

The interface has also been tested in a different domain, Image Filtering, with promising results. It could easily be applied to many other use cases, such as stage lighting design, or parametric 3D design, as each has a high dimensional parameter set which can be readily controlled over interfaces such as OSC.

7.1 Further Work

The evaluation of this interface was only carried out on a single synthesiser, due to the project's time constraints. An immediate goal is to validate the conclusions from this work on several other synthesisers. The next synthesisers to test should be chosen to include other categories of synthesis, such as Subtractive, Additive and Physical Modelling syntheses. It is non-trivial to extend this work to modular synthesisers, as there is neither a fixed number of parameters, or a fixed achitecture, so different presets may be uncomparable with any common error metric.

The interface has been designed to be as general purpose as possible, allowing it to control

arbitrary synths over the OSC protocol, but due to a lack of standardisation between soft synths, and lack of implementation time, more work needs to be done to create a truly general purpose soft synth controller. As many soft-synths are in the Virtual Studio Technology (VST) format, making a version of the interface with acts as a VST Host, and uses the parameter retrieval and preset storage functionality of VSTs is a good next goal if this project is continued in the future.

The interface has been designed with touch-screen contollers in mind, so it would be interesting to implement the interface on a touch screen. Either a dedicated app could be made, or the very customisable OSC controllers Lemur (REFERENCE) or Mira (REFERENCE) could be used. The interface could be further extended by taking into consideration the Multi-Touch capabilities of modern touchscreens.

When the Blending Interface is used, the user carries out a large number of interactions, all of which show a users preference over hyperplanes of parameter settings. If recorded, it may be possible to use these interactions to form a training set to train more sophistocated models of user preference in the parameter space. This can be seen as an extension of the Bayesian Optimisation approach to preference galleries shown in **??**. As this technique relies on fitting a Guassian process to the high dimensionsal parameter space, large amount of training data is necessary, and these interactions may be a way to find such data. As each comparison is choosing the optimum in a hyperplane, as opposed to a binary choice, the interactions have the potentially to be more information dense. Such models of user preference could be used to devise a more sophistocated preset generation algorithm.

Aside from the initial confiuration of presets A, B and C, the Blending Interface currently does not use the preset dataset in its preset generation algorithm. Due to the effectiveness of using this dataset in the Selection Interface, the inclusion of this data is a worthwhile area of further research.

In typical synthesisers, the parameters have a hierarchical nature, as the synthesis algorithm is usally made up of several modules, each with their own parameters and/or sub modules. Many synthesisers store their presets in the JSON (Javascript Object Notation) format, and use such hierarchies to make clean data structures. It may be possible to make use of this hierarchical parameter structure to deduce covariances between parameters, as parameters in the same

module are likely to be much more co-dependant than those in seperate modules. This could be used to refine the preset generation and comparison algorithms, and help to detect and correct cases of permutation ambiguity. However this approach has several challenges, most importantly there is a lack of standardisation of preset storage hierarchies.

Bibliography

[1] Ableton.com. (2018). *Wavetable — Ableton*. [online] Available at: https://www.ableton.com/en/packs/wavetable/ [Accessed 18 Apr. 2018].

[2] Documentation.apple.com. (2009). *A Brief History of the Synthesiser*. [online] Available at: https://documentation.apple.com/en/logicstudio/instruments/index.html#chapter=A%26section=5%26tasks=true [Accessed 20 Apr. 2018].

[3] Jónsson B et al. 2015. *Interactively Evolving Compositional Sound Synthesis Networks*. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15), New York. DOI: http://dx.doi.org/10.1145/2739480.2754796

[4] Chowming, J.M. (1977). *The Synthesis of Complex Audio Spectra by Means of Frequency Modulation*.

[5] Dobrin, A. (2005). *A review of properties and variations of Voronoi diagrams*.

[6] Encyclotronic. (2018). *Yamaha SY22 Vector Synthesizer*. [online] Available at: https://encyclotronic.com/synthesizers/yamaha/sy22-r345/ [Accessed 18 Apr. 2018].

[7] Experiments.withgoogle.com. (2017). *The Infinite Drum Machine by Manny Tan & Kyle McDonald* - AI Experiments. [online] Available at: https://experiments.withgoogle.com/ai/drum-machine [Accessed 18 Apr. 2018].

[8] Fiebrink, R. et al. (2010). *The Wekinator: A System for Real-time, Interactive Machine Learning in Music*. Proceedings of The Eleventh International Society for Music Information Retrieval Conference (ISMIR 2010).

[9] Hantrakul, L. and Kaczmarek, K. (2014). *Implementations of the Leap Motion in sound synthesis, effects modulation and assistive performance tools*. In: International Computer Music Conference. [online] Available at: http://hdl.handle.net/2027/spo.bbp2372.2014.100 [Accessed 18 Apr. 2018].

[10] Harzing, A et al. 2009. *Rating versus ranking: What is the best way to reduce response and language bias in cross-national research?*. International Business Review, ISSN 0969-5931

[11] Hunt, A. & Kirk, R. 2000. *Mapping Strategies for Musical Performance*. Trends in Gestural Control of Music

[12] Kersten, S. *skUG SuperCollider UGen library*, 2008. http://space.k-hornz.de/software/skug/.

[13] Linn, R. *Keynote*. Audio Developer Conference, 2016. https://www.youtube.com/watch?v=3bHGeSv37rU [at 19:00]

[14] Marier, M. (2012). *Designing Mappings for Musical Interfaces Using Preset Interpolation*.

NIME.

[15] Miguel (https://math.stackexchange.com/users/95625/miguel), *Equation for non-orthogonal projection of a point onto two vectors representing the isometric axis?*, (version: 2017-03-07): `https://math.stackexchange.com/q/1123586`

[16] Native-instruments.com. (2007). *MASSIVE*. [online] Available at: `https://www.native-instruments.com/en/products/komplete/synths/massive/` [Accessed 18 Apr. 2018].

[17] Nicol, C. (2005). *Development and Exploration of a Timbre Space Representation of Audio*. Ph.D. University of Glasgow. References

[18] Opensoundcontrol.org. (2002). *Introduction to OSC* [online] Available at: `http://opensoundcontrol.org/introduction-osc` [Accessed 18 Apr. 2018].

[19] Pierce, D. (2017). *The hot new hip-hop producer who does everything on his iPhone*. [online] WIRED. Available at: `https://www.wired.com/2017/04/steve-lacy-iphone-producer/` [Accessed 20 Apr. 2018].

[20] Rice, David. (2015). *GenSynth: Collaboratively Evolving Novel Synthetic Musical Instruments*. 10.13140/RG.2.1.4691.6001.

[21] Shlens, D. (2014). *A Tutorial on Principal Component Analysis*. arXiv:1404.1100 [cs.LG]

[22] Softube.com. (2016). Softube - Modular. [online] Available at: `https://www.softube.com/modular` [Accessed 18 Apr. 2018].

[23] Smith B.D. (2017) *Play it Again: Evolved Audio Effects and Synthesizer Programming*. In: Correia J., Ciesielski V., Liapis A. (eds) Computational Intelligence in Music, Sound, Art and Design. EvoMUSART 2017. Lecture Notes in Computer Science, vol 10198. Springer, Cham

[24] Tubb, R. (2016). *Creativity, Exploration and Control in Musical Parameter Spaces*. Ph.D. Queen Mary University of London.

[25] Vail, M. (2014). *The Synthesizer*. New York, NY: Oxford University Press.

[26] Vintagesynth.com. (2012). *Yamaha FS1R — Vintage Synth Explorer*. [online] Available at: `http://www.vintagesynth.com/yamaha/fs1r.php` [Accessed 18 Apr. 2018].

[27] Yee-King, M. (2011). *Automatic Sound Synthesizer Programming: Techniques and Applications*. Ph.D. University of Sussex.

[28] Yee-King, M J. 2016. *The Use of Interactive Genetic Algorithms in Sound Design: A Comparison Study*. Computers in Entertainmnent, 14(3), [Article]

[29] *Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders*. Engel, J., Resnick, C., Roberts, A., et al. 2017, arXiv:1704.01279

[30] Singh, P. et al. (2015). *Histogram Equalization: A Strong Technique for Image Enhancement*. International Journal of Signal Processing, Image Processing and Pattern Recognition.