The University of Oxford

Engineering Science

4th Year Project Report

Using Machine Learning to Control Software Synthesisers

Thomas O'Connor

Supervisor: Prof. Stephen Roberts, FREng

May 1, 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 that create sounds in response to musical input, typically in the form of *MIDI* (*Musical Instrument Digital Interface*) messages, and are widely used in a variety of musical contexts. Typical soft-synths have hundreds or thousands of parameters that control the sound generation algorithm, allowing the user to create sounds that suit their musical needs. This results in a high dimensional, non-linear parameter space that the user must search. Research indicates that users struggle to navigate such spaces with traditional 'knob and slider' user interfaces, 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 more efficient and 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 is created with a small dataset of presets. Principal Component Analysis (PCA) combined with Histogram Equalisation is shown to be an effective method to create a low dimensional embedding of a synthesiser 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 upom 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 novel interface, it is also tested in a different context: Image Filtering.

Acknowledgements

I would like to express my deep gratitude to Professor Stephen Roberts for his insights, attention and enthusiasm while supervising this project, allowing me to spend a year pursuing my interests. I would also like to thank my personal tutor, Professor Stephen Duncan for the great tutorials, advice and continued support throuhgout my degree.

I would like to thank the contributors to the open-source programming language *Supercollider*, without with this project would have been considerably more challenging.

I wish to thank many of my friends who helped me brainstorm ideas for this project, and gave valuable inights. In particular Joe, Toby, James, Hugo, Gethin and Henry.

Thanks to Frances, Ben, Ilari, and Alex for their detailed proofreading of this report, and their many useful suggestions.

I'd like to thanks John and Gemma for their contintued support, generosity and for inspiring me to study engineering.

Thanks to Madeleine for keeping me sane and always being there for me while I've been lost in the world of synthesisers.

Finally, I wish to thank my parents for their love and support, and for encouraging me to pursue my passions.

Contents

Chapter 1

Introduction

The invention and subsequent development of the *synthesiser* (synth) has been one of the main driving forces behind the evolution of popular and experimental music over the last century [35]. With the current ubiquity of cheap and 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 [26]. Such productions commonly use *software synthesisers* (soft synths) to produce many of the sounds.

Typical soft-synths require considerable technical knowledge to use, and have hundreds or even thousands of *parameters* that 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 parameters which produce useful 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 synths.

A conclusion can be drawn from this: more work needs to be done to make synths easier to use. To quote Roger Linn, a pioneer of musical interface design, in a 2016 keynote: [19]

> "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 detail 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 synths, 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 very useful to 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 undesirable sounds. By analysing the timbre of the presets, rather than the timbre of the entire parameter space, a simpler, and more useful control space can be identified, whilst only requiring simple statistical techniques to be used.

Humans have impressive abilities of spatial memory, [29], spatial reasoning [5], and memory of colours [10], and this can be exploited to increase the effectiveness of the interface. In particular, effort will be taken to maintain visual consistency throughout the interface.

Users are much better at giving a preferance between several options, rather that scoring individual options numerically [15, 4]. This is exploited in the Blending Interface (§5.3.1), 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 synth for this interface are identified, and a purpose built synth created to test the interface with.
- A small dataset of synth 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 upon users' preferences is developed.
- A Perfect/Imperfect user model is developed to numerically evaluate synth 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 (synths) 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 synths closely follows the evolution of electronics over the last century.

Early synths, 1900 - 1960, used the technologies of the time to create sound. 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 (Fig. 2.1). As this technology matured, many more modules were invented, such as ring-modulators, sequencers, and effects units such as reverb. Later *analogue synthesisers* were fixed architecture (or semi-modular), and much more compact and user friendly, such as the *Minimoog* and the *Prophet-5*. At this time, synths began to feauture commonly in popular music, both in the studio and in live situations [35].

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* synth.

Since the invention of personal computers, many popular synths 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 synths, many novel software synths 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 synths, as well as deatiled explanation of all aspects of sound synthesis can be found in *The Syntheser* by Mark Vail [35].

The common methods of audio synthesis are as follows [24]:

- 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 synth 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 (Fig. 2.1).

Digital synthesisers can be much more flexible in their interface, as the audio signals are not

Fig. 2.1. Analogue synthesiser - *System 35* (Moog) [22]

passing directily through the controls. Early digital synthesisers had unintuitive user interfaces consising of LCD screens and rows of buttons to alter the parameters, hidden in a series of menus (Fig. 2.2). As these interfaces lacked the hands-on control that musicians loved from analogue synths, many digital synths were less popular despite being capable of producing a much wider variety of sounds. Many users found the process of programming these synths to be incredibly frustrating [35]. To make the interfaces more user friendly, most digital synths are provided with a set of presets - parameter settings that produce desirable sounds - which demonstrate a wide range of the synth's capabilities.

Fig. 2.2. Digital synthesiser - *FS1R* (Yamaha) [36]

Soft-synths have a wide variety of interfaces. Many follow similar design patterns to analogue synths, but with additional interactive elements such as user-drawable waveforms. An example of this is the very popular *Massive* by Native Instruments [23] (Fig. 2.3). Other soft-synths offer much more visual and interactive interfaces, such as *Wavetable* by Ableton [1] (Fig. 2.4).

Fig. 2.3. Software Synthesiser - *Massive* (Native Instruments) [23]

Some soft-synths have modular designs, such as *Modular* by SoftTube [32], and some allow users to design synthesis modules from scratch, such as *Max/MSP* and *Supercollider*.

Fig. 2.4. Software Synthesiser - *Wavetable* (Ableton)[1]

There has been a wide range of reasearch in the Computer Music field in the use of multidimensional controllers, such as the *Myo*, *Leap Motion* and *Microsoft Kinect*, to control synthesis parameters in real time [14, 34], and research into advanced user interaction types, such as waveform blending (see *wavetable synthesis* [1] and *vector synthesis* [11]), preset blending [20], randomisation of presets, and many more [38, 33]. 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 users find programming synths difficult, due to the large amount of technical and creative skill necessary ([38] §4.1.1). There have been several previous attempts to use machine learning to make programming synths easier and more expressive, from both academia and commercial products ([38] §4.1.2).

The Google NSynth project [40] develops a technique for interpolating between sampled instruments, in the latent space (low-dimensional internal representation) of an *Autoencoder* rather than operating directly on the audio recordings. This project has impressive results, generating much higher quality synthetic instrument samples than can be generated by interpolating between audio samples. However, 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, as there is an exponential increase in the number of possible combinations that have to be pre-computed as more instruments are added.

Tone-matching is an active area of research in which synths are 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 synth 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 synth. The Mel Frequency Cepstrum Coefficient (MFCC) has been identified as the best metric to use, as it is based on human perception [38].

Several works have applied *interactive genetic algorithms* to synth programming [17, 27, 39]. Typically the genetic algorithm produces a variety of presets, 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 synths, allowing the synthesis architecture to mutate. The Blending Interface (§5.3.1) 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 was applied to audio recordings in the *Infinite Drum Machine* [12]. In this work, *t-Distributed Stochatic Neighbour Embedding* (tSNE) was used to create a two dimensional mapping of a large number of audio recording (samples). An interface based on this mapping was used for sample-selection in a drum machine.
The Selection Interface (§5.2) draws inspiration from this project. It would be possible to carry out the same process for a synth if first an audio recording was taken of each preset. However, the best way to select which audio recording should be taken from each preset is not clear, 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 presets of a synth. This approach has many benefits such as the speed of the mapping, and the amount of storage required.

The *Wekinator* [13] 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

Several results from the field of *Human-Computer Interaction* (HCI) give useful insights into synth interface design.

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

- 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 [16].

In [34], a number of guidelines for creative musical interfaces are presented:

1. *Low dimensionality*: Control devices often have fewer parameters than synthesis algorithms. Given the brain's limited conscious multi-tasking abilities and working memory capacity, simple controllers are preferable.
2. *Locality*, or distance preservation: Distance travelled in control space, should be reflected in the distance travelled in parameter space, and ideally perceptual distance too.
3. *Revisitability*: If a point in control space is revisited, the sound should be 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 occurs, such as a scroll, it will have a certain effect on the sound. More extreme versions of the gesture should produce more of the same effect.

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

In [34](§5.5.3), the EARS model for creative cognition is developed. It describes four modes of creative cognition which can be used to design and evaluate creative interfaces (Fig. 2.5).

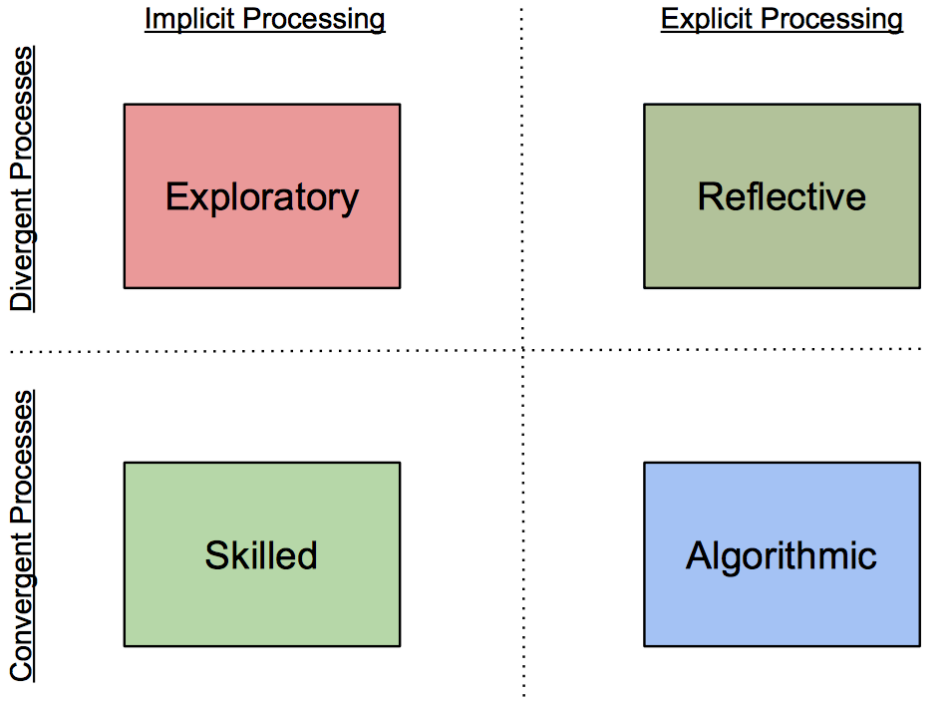


Fig. 2.5. The four quadrants formed by drawing distinctions between implicit vs. explicit thinking (left/right) and divergent vs. convergent thinking (top/bottom). Convergent processes are conceptual/parameter space traversal mechanisms that work upon improving the fitness of solutions. Divergent processes generate candidate solutions via traversal mechanisms that ignore any prediction of increasing value, e.g. creating lots of scattered points [34].

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 is evaluated using this model in Section 6.5.

Chapter 3

Description of Synthesiser Algorithm

3.1 Description

It was necessary to choose a synth to use, in order to develop the interface. To remove some of the complexities of communicating with commercial synths, a purpose built synth was made in the programming language *Supercollider*. A *six operator phase modulation* (PM) synth was created,[1] loosely based on the Yamaha *DX7*, and was designed to be representative of typical commercial soft-synths.

The synthesis algorithm uses the *FM7 UGen* for Supercolldier [18]. This implements six *operators*[2] with independent amplitudes and frequencies, whose outputs can be used to modulate each other's phase. This is described by the following discrete-time equation [38]:

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

where T is the sampling interval, k is the sample number, $y_i[k]$ is the output of operator i at time kT, 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 synthesis architecture (Fig. 3.1) 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 $\{0.5, 1, 2, 3, \dots\}$, allowing the operators to set to different harmonics of the base frequency. The fine

[1] PM synthesis is very closely related to FM synthesis [6], and the names are often used interchangeably.

[2] A general term for waveform generators in PM synthesis. In this case, sine-wave generators are used.

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 are then continuously *amplitude-modulated* by two *low frequency oscillators* (LFOs), and a *modulation envelope*, as described by the discrete-time equation:

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

where $a_i[k]$ is the amplitude of operator i, $\mathrm{LFO}^a[k]$ and $\mathrm{LFO}^b[k]$ are the LFOs' output values, $\mathrm{ENV}[k]$ is the modulation envelope's output value, all at time kT. lfo_i^a, lfo_i^b, and env_i^{mod} are parameters which determine how much operator i is modulated by the respective signals.
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 envelope generator*[3], which is triggered by the MIDI Note ON message.

After amplitude modulation, the operators amplitudes and frequencies are used as inputs to the previously described phase modulation algorithm. The six outputs from this algorithm are then mixed together, and multiplied by the *amplitude envelope* (another ADSR envelope generator) to form the output signal: $Y[t] = \mathrm{ENV}^{amp}[k] * \sum_{i=1}^{6} y_i[k] * a_i^{output}$, where a_i^{output} is the Output Level parameter for operator i.

The synth has 96 parameters, as described in Table 3.1. All of the parameters can be easily adjusted in real time, by sending messages over *Open Sound Control (OSC)* [25], a UDP based protocol for inter-application communication.

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

[3] A commonly used type of envelope generator with the parameters Attack, Decay, Sustain and Release.

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

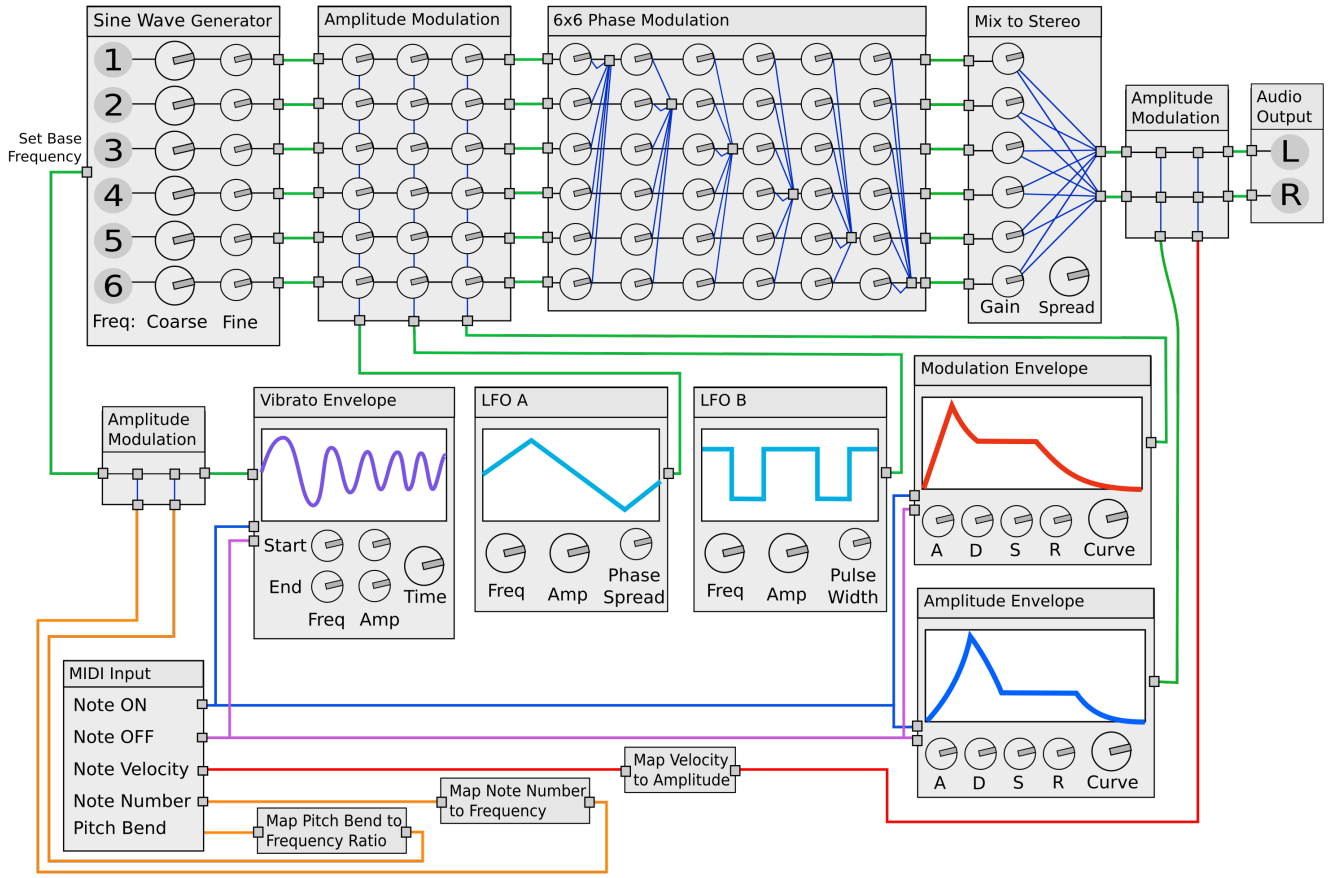


Fig. 3.1. Software Synthesiser Schematic - a 6 operator Phase Modulation synthesiser, with two LFOs, and seperate envelope generators for modulation, amplitude and vibrato. Implemented in *Supercollider* with 96 parameters controlable in real time over the *OSC* protocol.

3.2 Design choices and justification

Typical commercial synths have a large number of parameters[4], which can be discrete or continuous, and are usually constrained within a range. This synth features a combination of continuous and discrete parameters, all of which are bounded. Commonly used bounds are: $[0, 1]$, $[0, k]$ and $[-k, k]$, where $k \in \mathbb{R}^+$ (See Table 3.1). Some synths have *modular* architectures, which allow for considerable flexibility in sound generation. However, due to the *combinatorial explosion* of potential parameter combinations, and difficulties blending between presets, modular synths are not addressed in this work. Some synths 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

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

work. Both of these types of synths would be interesting to pursue as a follow-up work.

This work is aimed at analogue-style synths, in which there are a medium number of parameters with a fixed internal routing. Effects modules, such as reverb, delay and chorus have not been included in the synth, as these can easily be added to the signal chain by users. Effects modules also tend to be much easier to control than synths, 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 function, with no covariance between the parameters.

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

The synth 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 the DX7 does, only two 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 parameters in most presets for this synth. 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 synths, and presents a challenge when trying to learn useful information from the parameters.

In Yee King's thesis [38], 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 tone-matching goal 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.

As this synth has six 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 (§5.4) as two presets can have very different parameters despite having the same sound.

Chapter 4

Description of Full Interface

The full interface developed in this project 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 designed to use as little information about the synth 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). This chapter gives an overview of each interface, with a detailed description in the next chapter.

4.1 Traditional Interface

The *Traditional Interface* is designed to be representative of a typical soft-synth interface. It has a knob or slider for each synthesis parameter, 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 Fig. 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.

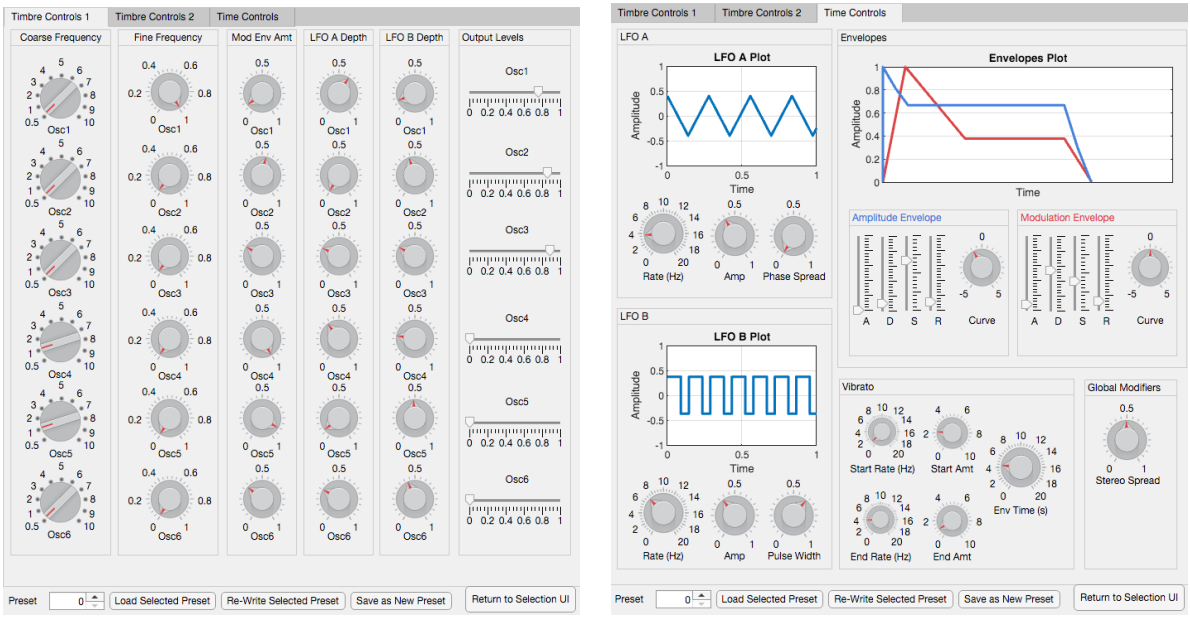


Fig. 4.1. Traditional Interface - 'Timbre Controls 1' and 'Time Controls' pages

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 of *Macro Controls* (usually eight), each of which vary one or more parameters, and aim to give the user a quick way to fine-tune the presets to their needs.

The *Selection Interface* is a new approach to displaying presets, arranging the presets as cells of a *2D Voronoi diagram* [9], 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 (Fig. 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 combined with Histogram Equalisation.

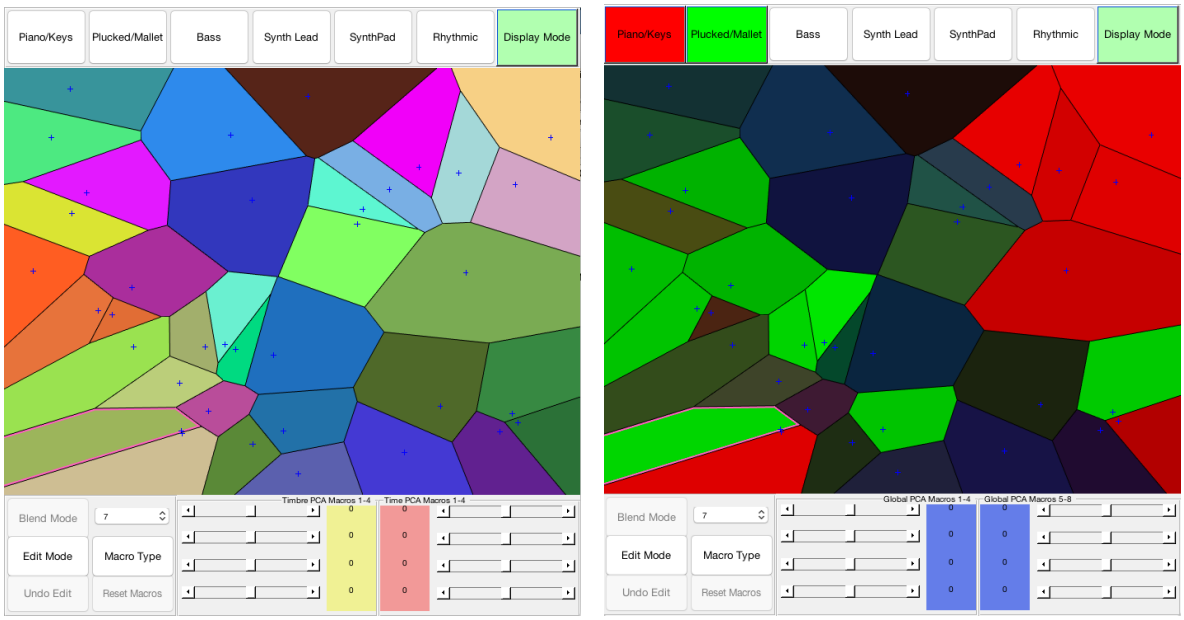


Fig. 4.2. Selection Interface - Normal View, and Category Highlight View

4.3 Blending Interface

Preset Blending is a 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 (Fig. 4.3). 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 (§5.3.2).

This process can repeat 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 return 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 (Fig. 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.

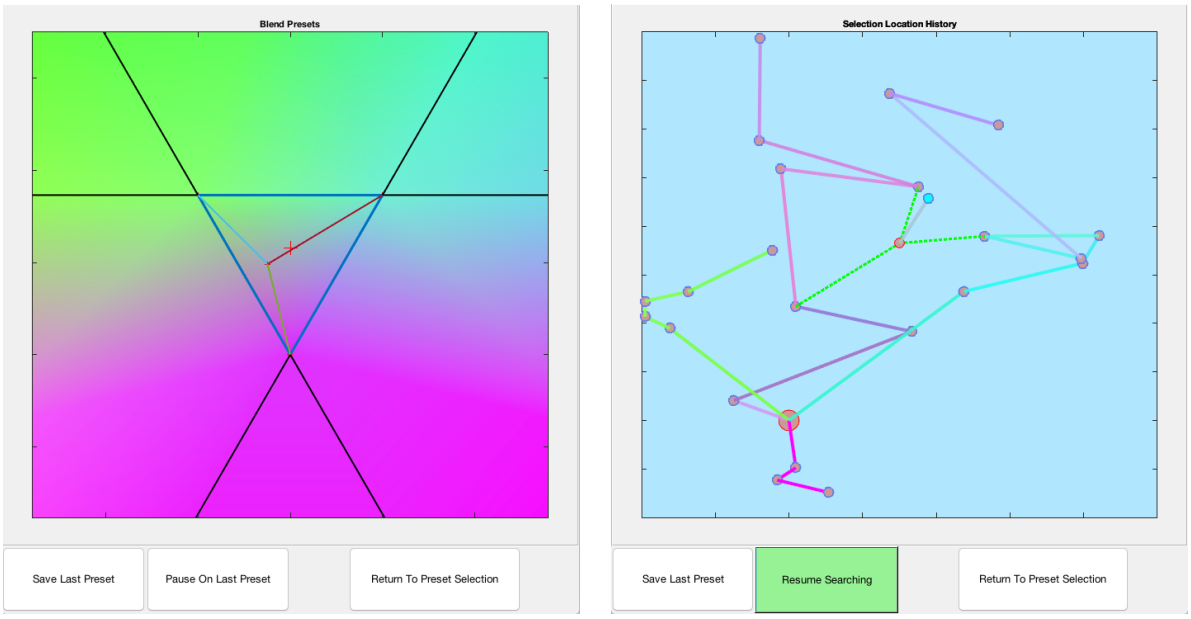


Fig. 4.3. Blending Interface - Normal View, and Selection History View

[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 (Fig. 4.4). To undo the edits made to the preset the user can click the ‘Undo Edit’ button. In this way the user can move back and forth between the Selection and Traditional 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 three 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, and can be edited with the Macro Controls or the Traditional Interface.

Parameter visualisations are displayed alongside all of the interfaces to give the user more feedback of the changes they are making with the interface, and to aid in understanding the synthesis algorithm. These are shown on the left and bottom sides of the interface in Fig. 4.4.

The following chapter gives a detailed description of each of the interfaces, and describes some of the technical challenges which had to be overcome to make the interfaces work effectively

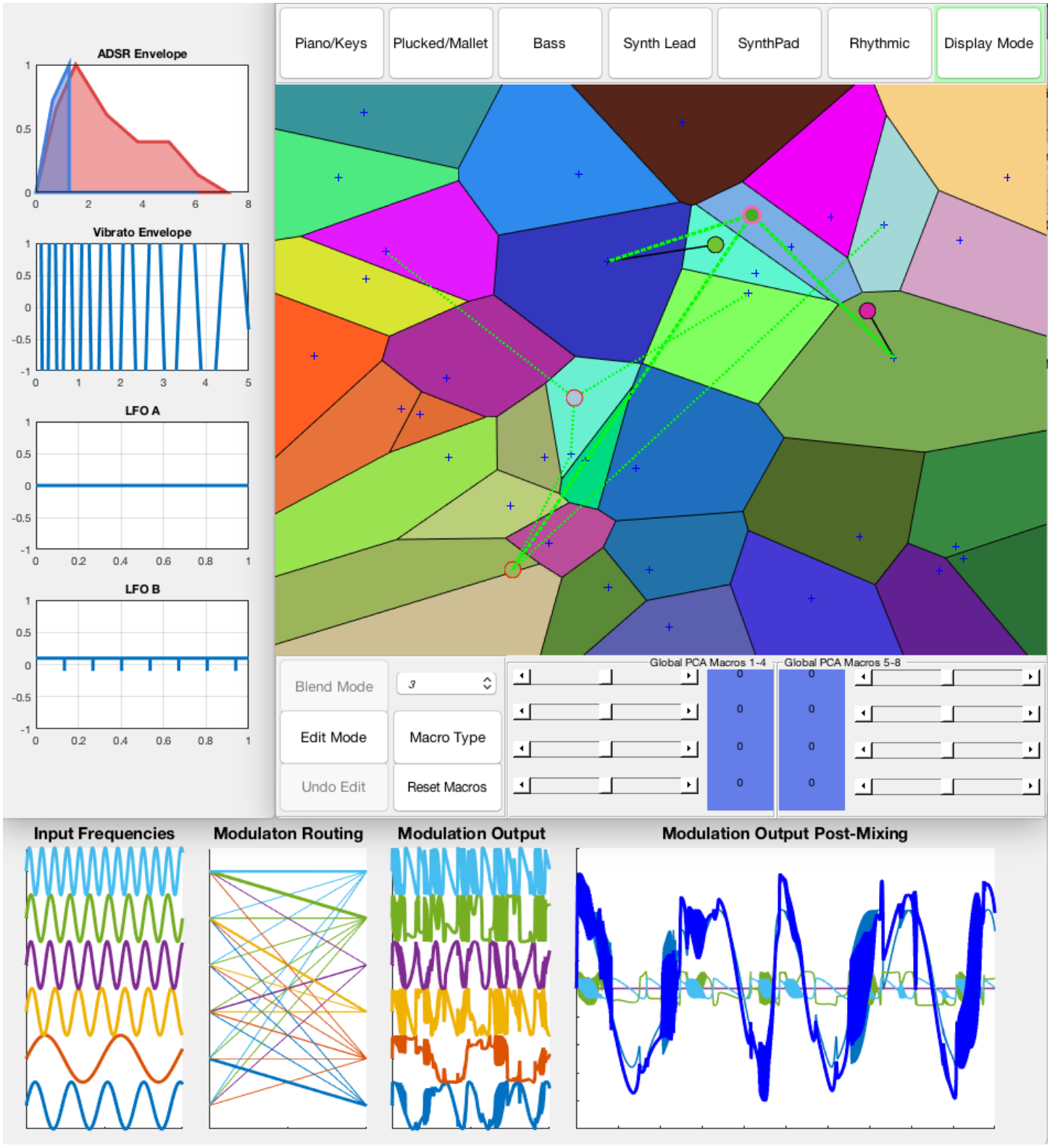


Fig. 4.4. Combined Interface - PCA view. The top right corner of the interface can be swapped for the Traditional Interface and the Blending Interface. The parameter visualisations on the left and bottom sides are common to all three interfaces. The combined preset markers are shown on the Selection 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 synth.

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. Especially effective for more intuitive sections of the synth architecture, such as the LFOs and envelopes.

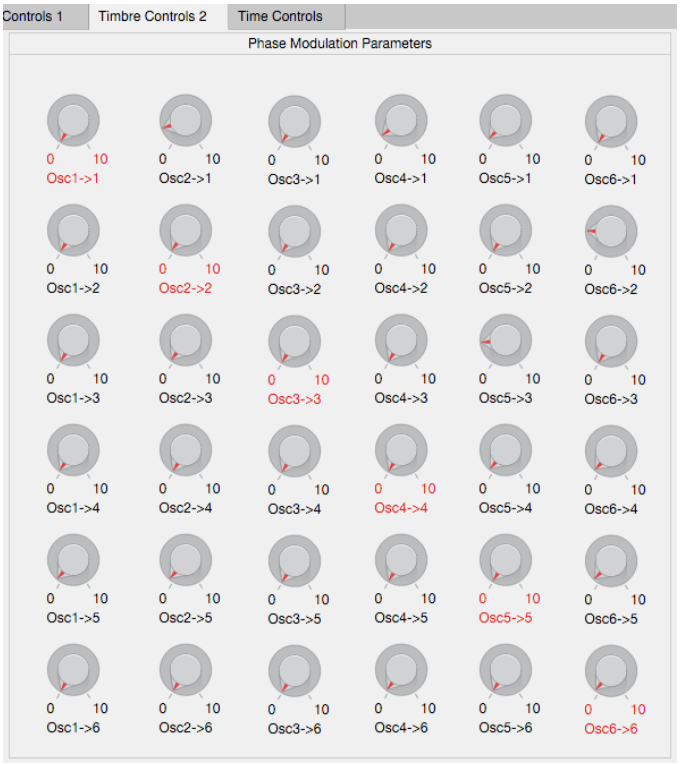


Fig. 5.1. Trad. Interface - 'Timbre Controls 2'

5.1.2 Weaknesses

Even if the user knows exactly what each parameter should be set to, it still takes 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 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.[1]

[1] For detailed evaluation of similar interfaces, see [16, 34, 38].

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 [12]), *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 $\mathbf{\Sigma}_x = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T$, where $\mathbf{X} = [(\mathbf{P}_1 - \mu), (\mathbf{P}_2 - \mu), \dots (\mathbf{P}_n - \mu)]^T$, $\mathbf{P}_i$ is the ith preset, μ is the mean of the presets, and n is the number of presets. The eigenvalues of $\mathbf{\Sigma}_x$ 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 [31]. 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 Macro Controls that only vary a subset of the parameters may be more intuitive to users.

5.2.2 Global PCA vs Time/Timbre PCA

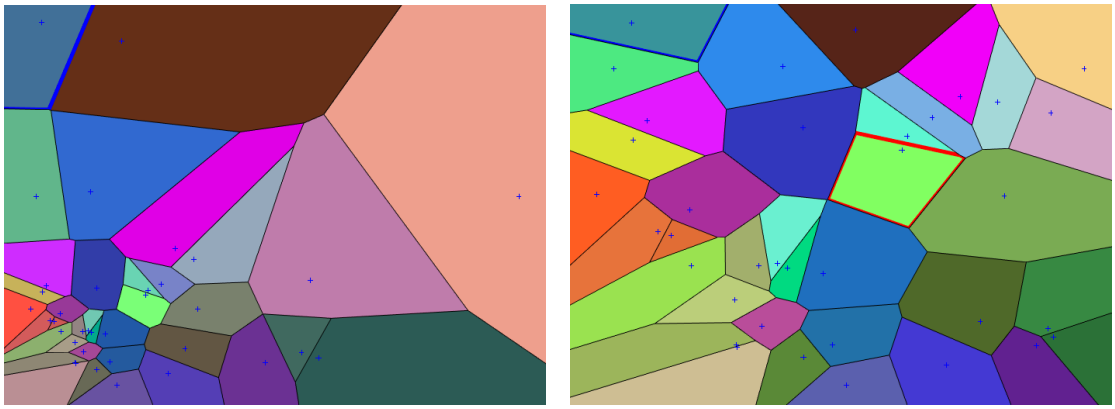
The PCA is calculated in two different ways for different sections of the interface. In *Global PCA*, each of the 36 presets' 96 parameters are arranged in a 36×96 array. PCA is carried out on this array to produce the Global principal components.
In *Time/Timbre PCA*, the 96 parameters are partitioned 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 of 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, §5.3.1), and automatically recalculate the Macro Control mapping 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 (Fig. 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

Fig. 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* [41] was used, resulting in the mapping shown in Fig. 5.2b.

Histogram equalisation in one dimension (Fig. 5.3) is described below.

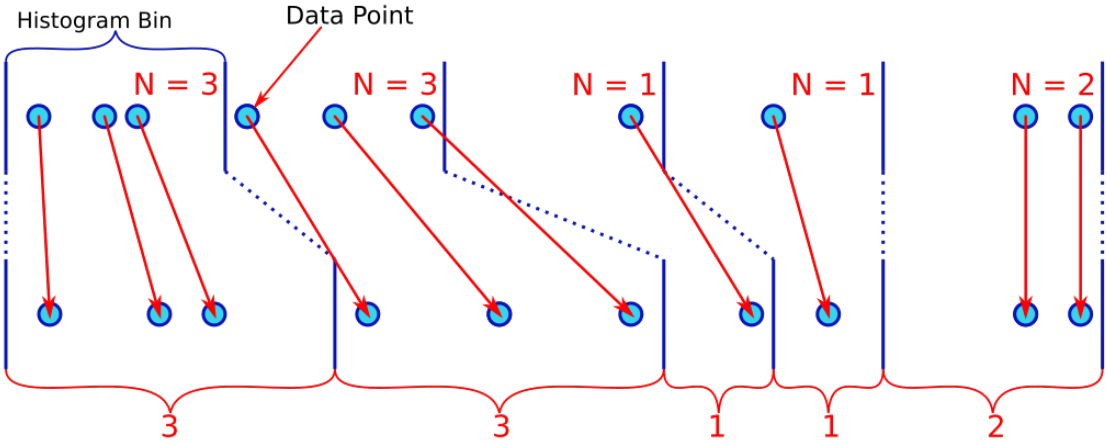


Fig. 5.3. Histogram Equalisation in one dimension

A histogram is created by splitting the data into a number of eqaully spaced bins (six bins was 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 five 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 three principal components.

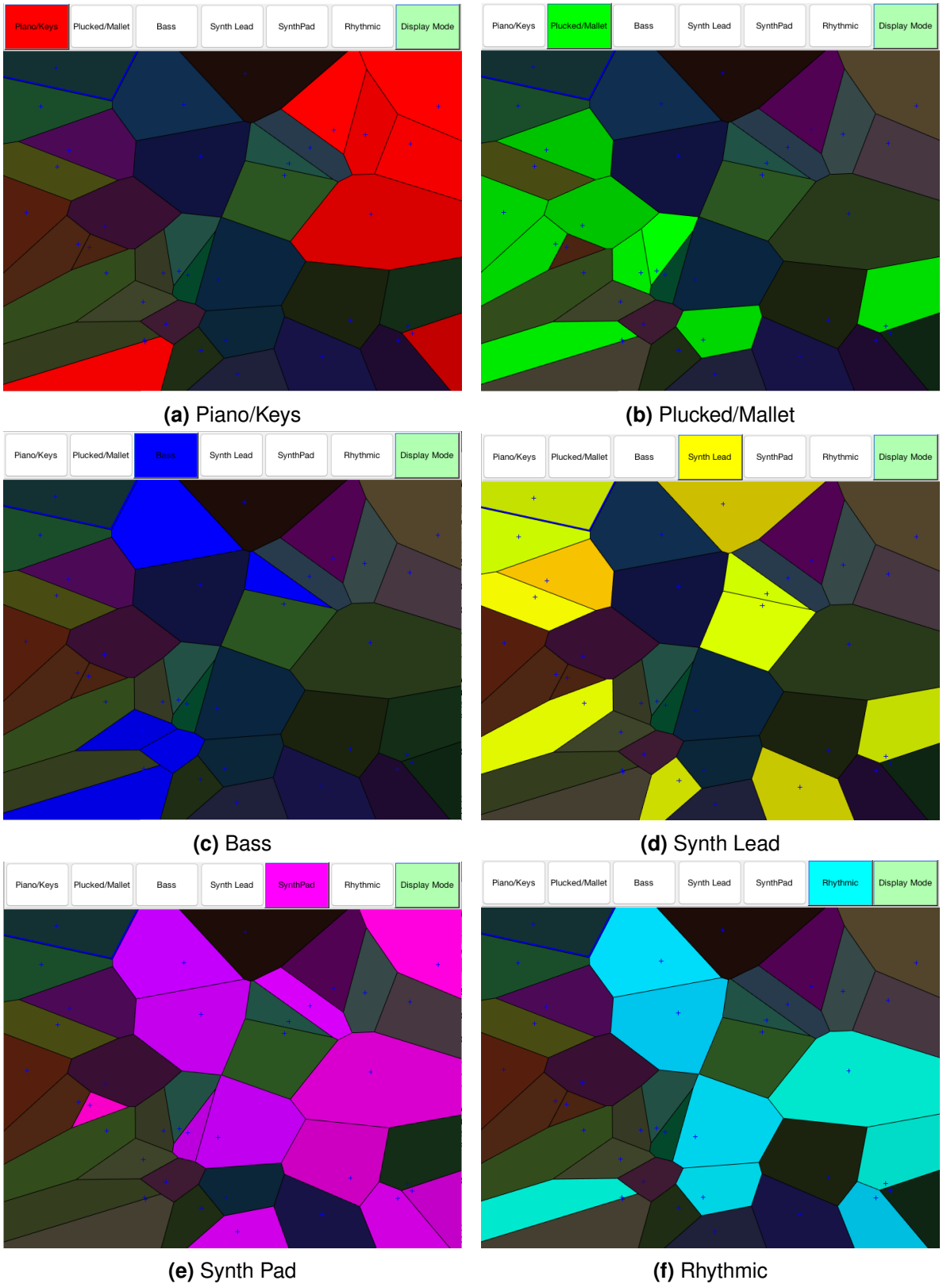
This approach, although simple, effectively redistributes the presets. As long as none of the bins are empty, the mapping in each dimension is peicewise-linear and continuous. This means that the Combined Preset Markers can be positioned correctly on the graph by first calculating the PCA scores, then 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 smoothly 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 (Fig. 5.4). This demonstrates that **PCA clusters 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.
Key limitations of the above analysis are 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 parameters than if they'd been made from scratch by a person. To validate this approach more thoroughly, sets of presets from several pre-existing commercial synths should be tested. Once this larger dataset has been collected, a comparision of different dimensionality reduction techniques should be carried out.

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*.
Fig. 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 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.



(a) Piano/Keys **(b)** Plucked/Mallet

(c) Bass **(d)** Synth Lead

(e) Synth Pad **(f)** Rhythmic

Fig. 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.

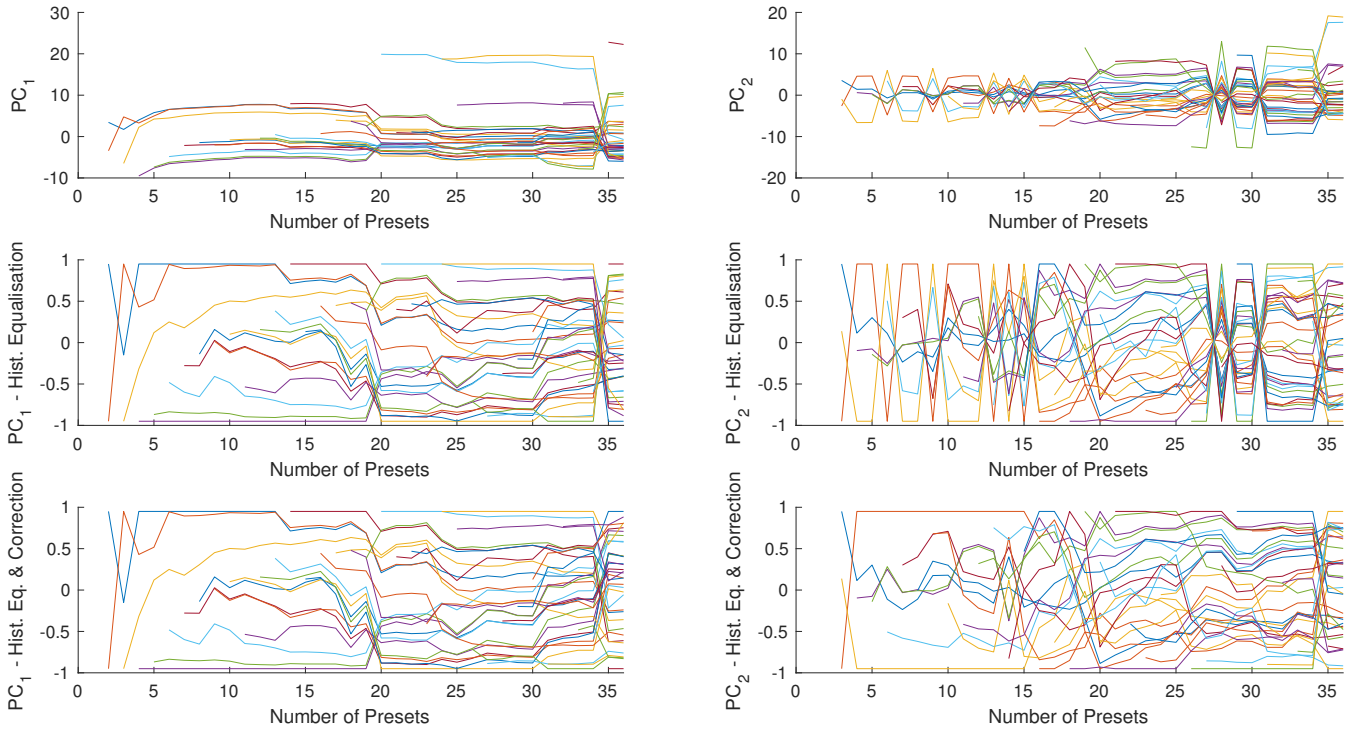


Fig. 5.5. PCA stability with number of presets. These plots show the positions of the indiviual presets when mapped onto the first and second principal components, with and without Histogram Equalisation. The mapping is recalculated each time a new preset is added. The positions are relatively stable 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 Fig. 5.6.

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 PC_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 $\|PC_i^{k+1} - PC_i^k)\|_1 > \|PC_i^{k+1} - (-PC_i^k))\|_1$, flip the sign of all remaining presets.

This algorithm is used on PC_3 in Fig. 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 is necessary to do is keep track of which components have been flipped, and each time a new preset is added, check if $\|PC_i^{k+1} - PC_i^k)\|_1 > \|PC_i^{k+1} - (-PC_i^k))\|_1$ to determine whether to flip component i, and update the record of flipped components.

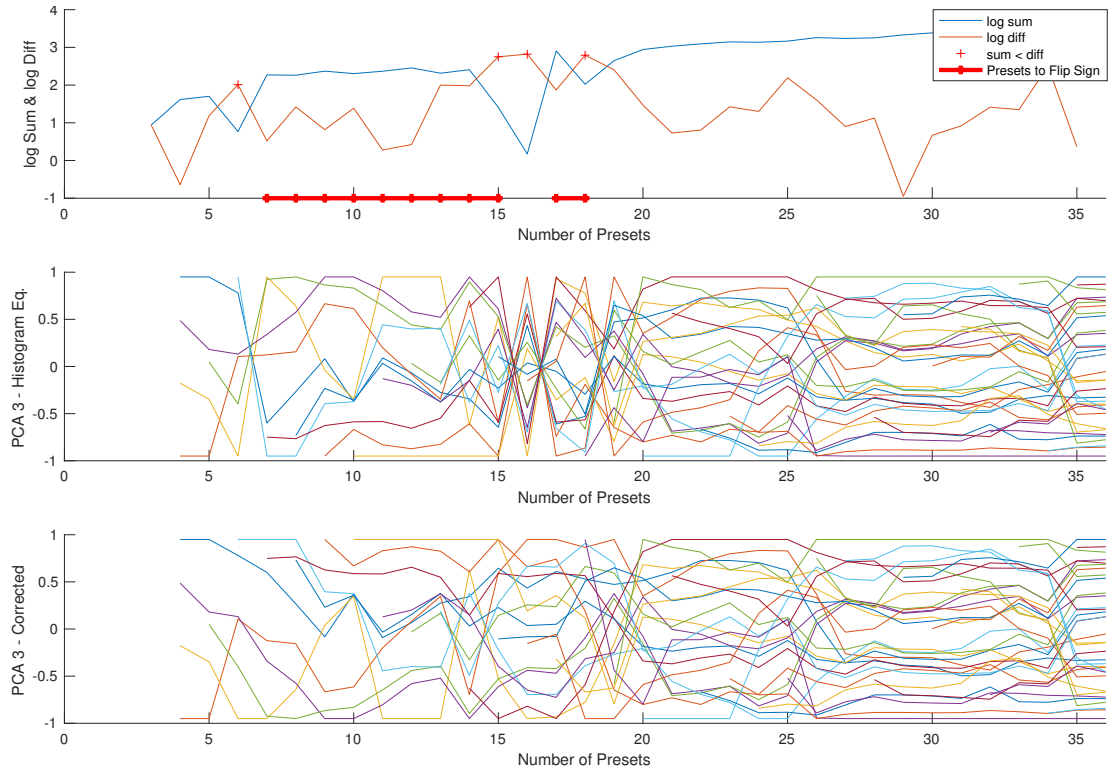
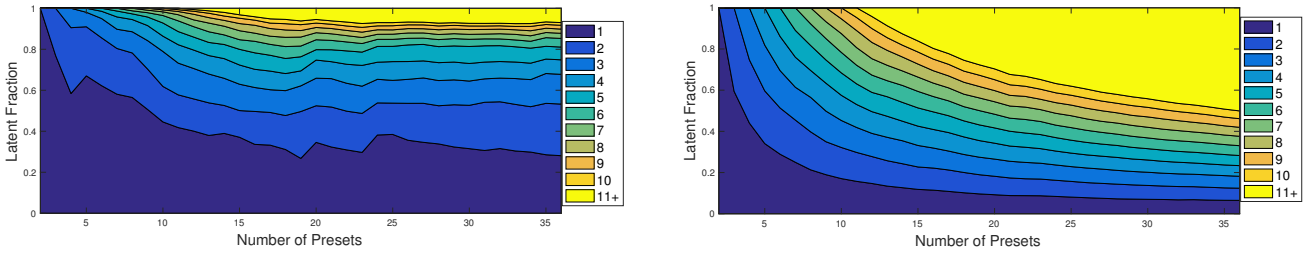


Fig. 5.6. PCA Sign Flip Correction Algorithm.

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. Fig. 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 asymptotes to a relatively stable state. Fig. 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 the 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 synth datasets**, however testing on other synth datasets is necessary to validate this result.



(a) Original Presets **(b)** Randomised Presets

Fig. 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. Fig. 5.8 shows the the key steps of the Blending Interface's algorithm, and Fig. 5.14 shows one iteration of the Blending Interface applied to a simple 6-dimesional parameter space.

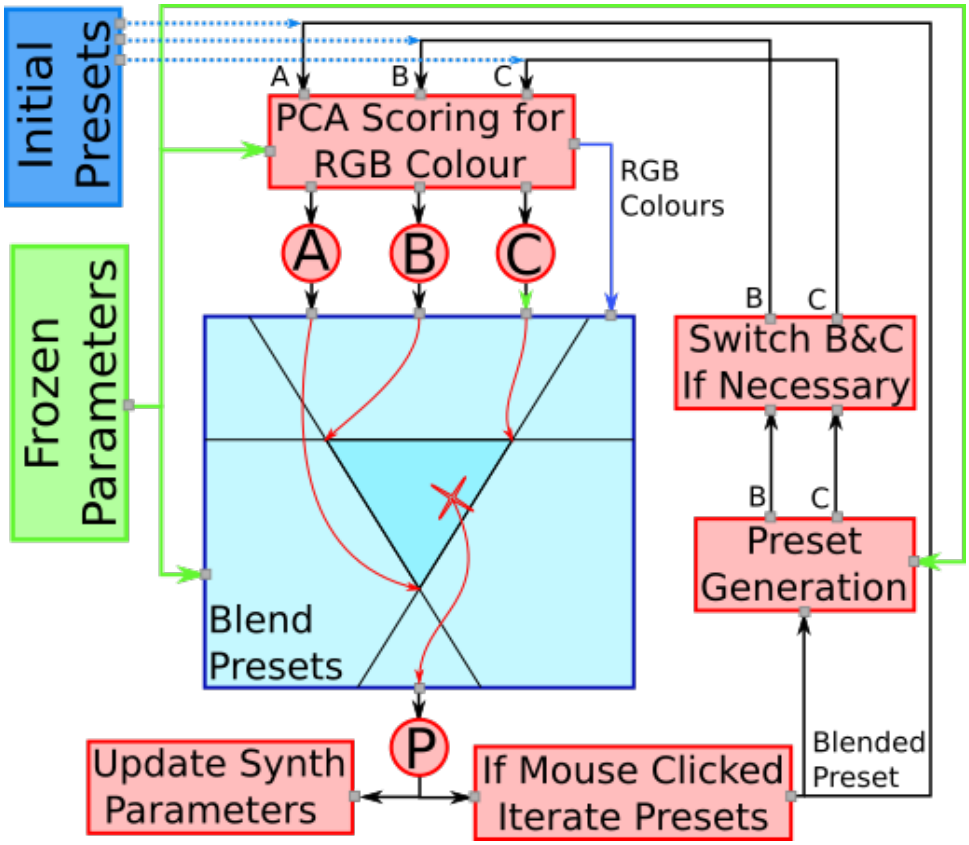


Fig. 5.8. Blending Algorithm flow chart

The blending of presets is calculated as a non orthogonal vetor decomposition: [21]

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

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 (Fig. 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 interpolate between all possible combinations of $m+1$ presets, so on a 2D screen, three presets should be used.[2] 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* (See Fig. 5.14).

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 to the set $\{0.5, 1, 2, 3, ...\}$, with the desicion boundary equidistant from neighbouring numbers (Fig. 5.10).

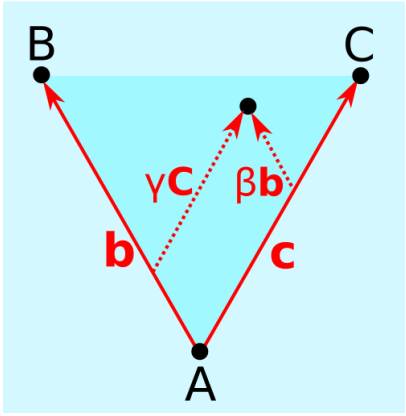


Fig. 5.9. Non-orthogonal vector decomposition

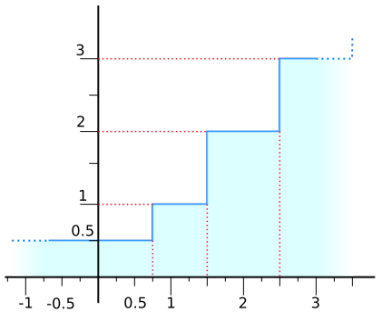


Fig. 5.10. Mapping from continuous to coarse frequency

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. Equation 5.1 is used to calculate the blended preset at each of the points marked on Fig. 5.11, and the PCA Scores are evaluated for each point. $PC_{3,4,5}$ 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.

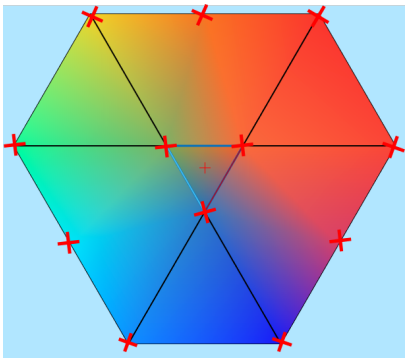


Fig. 5.11. $PC_{3,4,5}$ applied to RGB colour of Blending Interface

Each time the presets are iterated, the first principal component, PC_1 is calculated for the generated presets B and C. 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 maintain consistency between the two interfaces,

[2] 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.

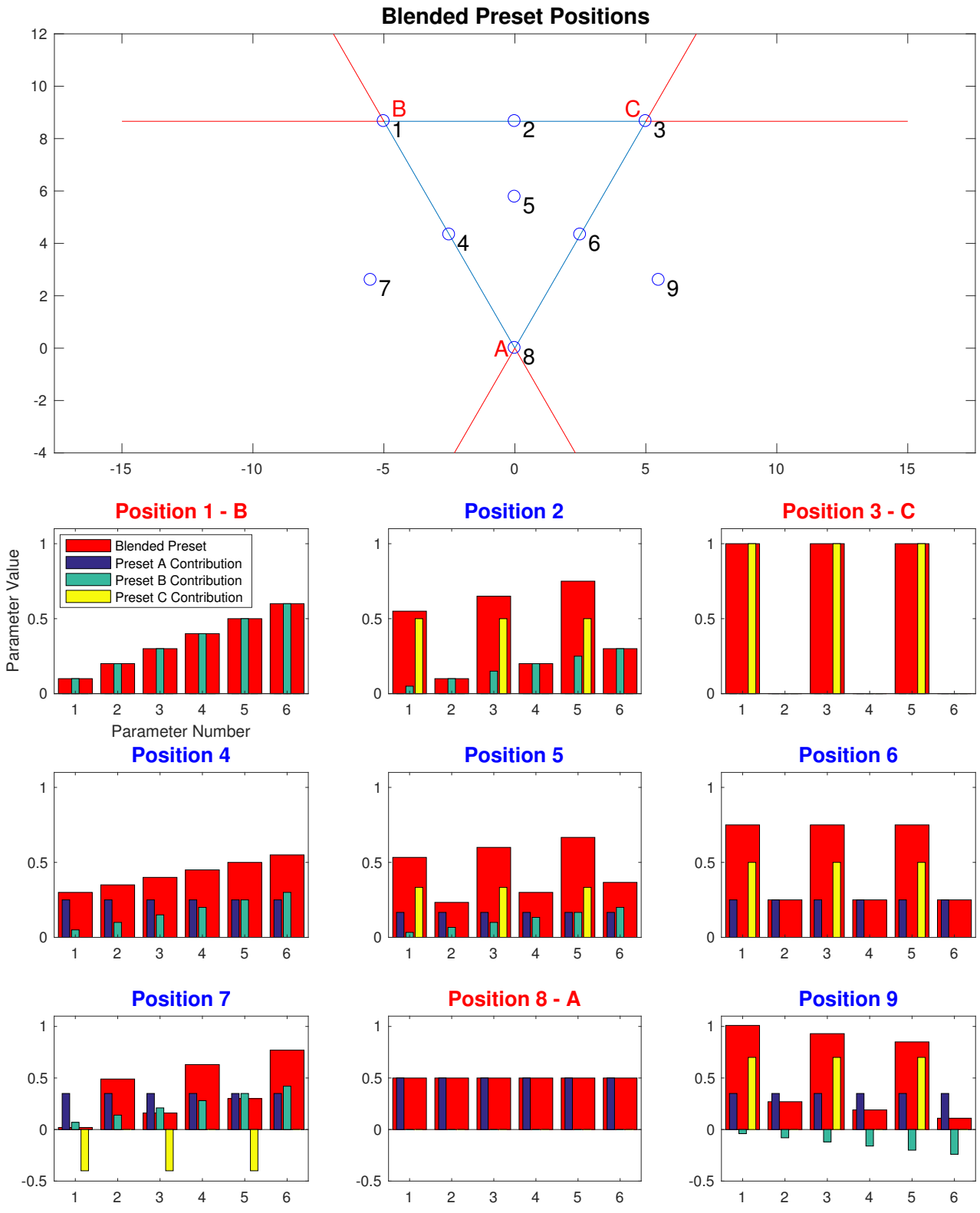


Fig. 5.14. Preset blending applied to three presets (A, B, C) for a simple 6 dimensional parameter space. Each bar graph displays the blended preset produced by its corresponding location on the 'Blending Preset Positions' plot. Positions 8, 1 and 3 display presets A, B and C, and all other positions are a linear combination of these presets (§5.3.1). Each of the graphs' thick bars correspond to a particular parameter of the blended preset. The thin bars represent the contributions of presets A, B and C, which are summed to give the blended 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 (Fig. 5.12). The colours of each edge of the graph are chosen based on $PC_{3,4,5}$ of the presets. See Fig. 4.3 for a typical Selection History plot.

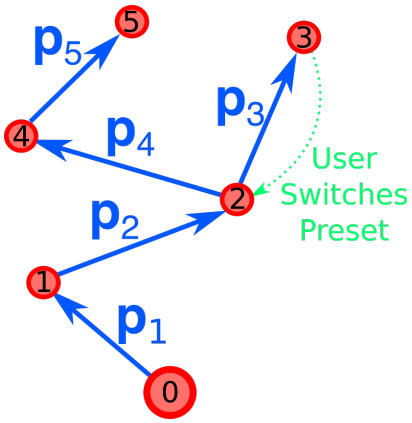


Fig. 5.12. Selection History plot construction

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.

The user has the option to freeze sections of the parameter space (Fig. 5.13). The parameters are divided into Time and Timbre parameters, and then subdivided into a total of 12 smaller categories (Table 3.1). 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.

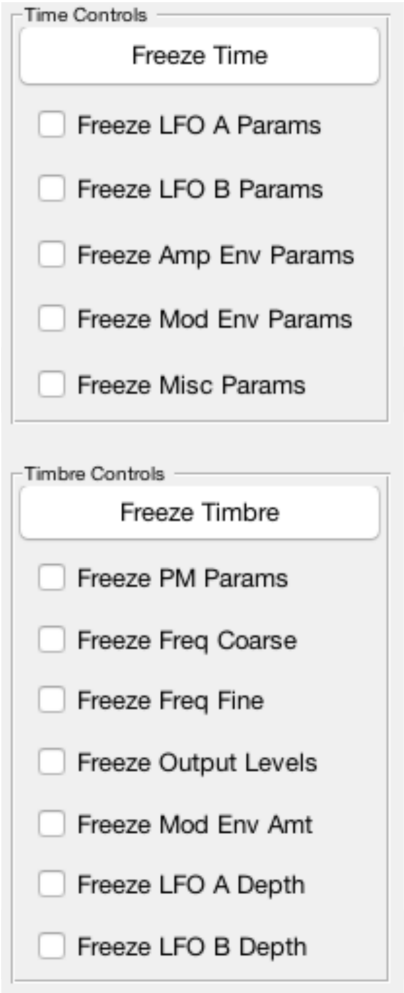


Fig. 5.13. Parameter Freezing

5.3.2 Preset Generation Algorithm

The aim of the Preset Generation Algorithm is to suggest useful new presets, based on the user's previous selections. In the field of *Active Learning* [30], several similar problems have been adressed, the most relevant of which is a preference-gallery based approach to fine-tuning parametric procedural animations [4]. In this work a user is iteratively presented with four generated aminations, and selects their favourite. In each iteration, *Bayesian optimisation* is used to learn from the previous selections, and generate four new animations. This procedure

was shown to be superior to both a slider-based interface, and an interface in which the user rates each animation on a numeric scale. The gallery interface was improved by allowing the user to manually set and freeze parameters, especially for expert-users. Bayesian optimisation is well suited to this task, as it explicitely deals with the *exploration-exploitation* tradeoff [42] and is designed to minimise the number of user queries. However, the main limitation with this work is that traditional Bayesian Optimisation can only be used for problems with low (< 20) dimensionality due to an exponential increase in necessary user queries. The animation model only had 12 parameters, and this still took a considerable ammount of training data to fine-tune the hyperparameters of the learning algorithm. Adaptations of bayesian optimisation have managed to solve certain higher-dimensional problems [37], but it is currently non-trivial to apply these results to general problems. Therefore an alternative algorithm was developed.

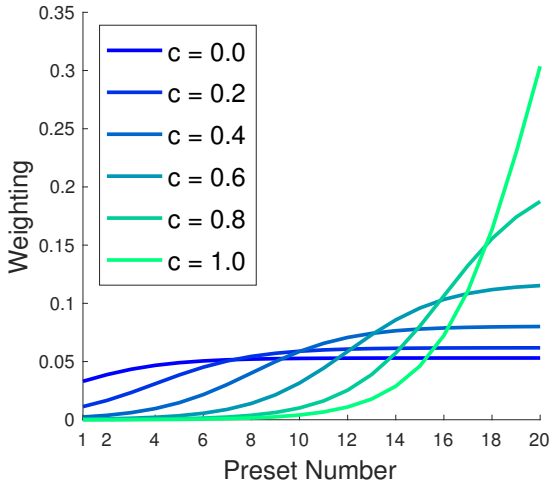


Fig. 5.15. Sigmoid weighting function

The new presets are a *stochastic weighted sum* of the k previously selected presets:

$$\{\mathbf{B}^{k+1}, \mathbf{C}^{k+1}\} = f(\eta \mathbf{P}^k + (1-\eta)\mathbf{w}^T\mathbf{H} + \varepsilon) \tag{5.3}$$

where $\mathbf{H} = [\mathbf{P}_1, \mathbf{P}_2, \dots \mathbf{P}_k]^T$ where $\mathbf{P}_i$ is the ith previously selected preset, $\eta \in [0,1]$ varies the weight of the last selected preset, and f as defined in §5.3.1. The weight vector $\mathbf{w}$ is generated by a *sigmoid weighting function*:

$$w_i = \frac{a}{1 + \exp\left(-\frac{10}{k}(i - ck)\right)} \tag{5.4}$$

where a is a normalising constant such that $\sum_{i=1}^{k} w_i = 1$, and $c \in [0,1]$ is the *center ratio* parameter, used to vary the weighting of recently selected presets (Fig. 5.15).

Zero-mean Gaussian noise ε is added to the weighted mean with a covariance defined by:

$$\Sigma = b \times (\nu\Sigma_P + (1-\nu)\Sigma_H) \times \xi^k \tag{5.5}$$

where $b \in \mathbb{R}^+$, Σ_P is the covariance (§5.2.1) of the preset dataset, Σ_H is the covariance of the previously selected presets[3], $\nu \in [0,1]$ sets the ratio of the covariances, and $\xi \in [0,1]$ causes the magnitude of the covariance to decay as k increases. The covariance was defined in this way so that the correlations from both the preset dataset and the previously selected

[3]When $k = 1$, $\Sigma_H = \mathbf{0}$.

presets are incorporated into the algorithm. If a user makes consistent decisions with regards to a particular parameter, its variance will decrease and eventually the parameter will become fixed. This algorithm has several hyperparameters which were set using an optimisation based approach, described in Section 6.6, then fine-tuned empirically until the algorithm gave sensible suggestions ($b = 0.4, c = 0.5, \eta = 0.5, \nu = 0.3, \xi = 0.9$).

5.4 Investigion of Permutation Ambiguity

Due to the equivilance of the six operators of the synth architecture, there is the possibility of permutation ambiguity in the dataset (§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 [38], Section 3.1.1). Therefore the dataset can be permutation-alligned by permuting each preset, such that the Output Level parameters 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:

$$C = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathrm{error}(\mathbf{P}_i, \mathbf{P}_j) \tag{5.6}$$

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 data. 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 incorporating more knowledge of the synth architecture can make the process more effective.

	Original Dataset	Permutation-Alligned	Randomly Permuted
Cost $\times 10^4$	2.427	2.452	2.8264 (mean of 50 runs, $\sigma = 0.0098$)

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 no obvious metrics to use to compare the interfaces with each other. *Fitt's Law* and the *Index of Search Space Reduction* [34] has been used for evaluating the throughput of multidimensional controllers, however these metrics are aimed at analysing users' interactions with a controller rather than the control mapping. The key question to answer is, are the Blending Interface and the Macro Controls better at editing presets than the Traditional Interface? The *Perfect/Imperfect User Model* is developed to answer this question, by analysing simulated users carrying out tasks on the interfaces. This analysis won't account for any of the creativity-based goals of the interface, but will help evaluate some of the practical considerations of the interface.

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 move from initial presets to goal presets using one of the interfaces.

The Perfect User has perfect knowledge of the synth and the interface, so can always choose the optimum position for a particular parmeter, 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 standard deviation σ (which can be varied to account for the different 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 operate somewhere in the middle of these regimes depending on their skill level.

6.2 Error Metric

A parameter-based error metric is used for these tasks due to the complexity of using a sound based comparisons of presets (§2.3). 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. It may be possible to incorporate more knowledge of the synth into the error metric to explicitely remove the effect of permutation ambiguity, and to better approximate the results of a sound-based error metric.

6.3 Evaluation 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 starting from the closest preset to the goal preset. The results of these tests are shown in Fig. 6.1. The cyan lines are individual instances of the Perfect Order test. The dark blue lines are individual instances of the Random Order test. The solid red lines are the mean of the Perfect Order tests (solid), and the Random Order tests (dotted).

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 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 Fig. 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**.

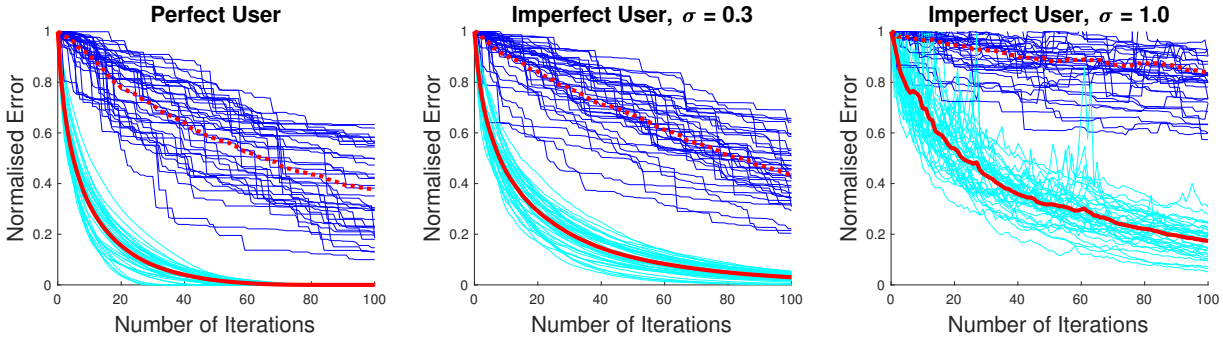


Fig. 6.1. Perfect/Imperfect user tests for Traditional Interface

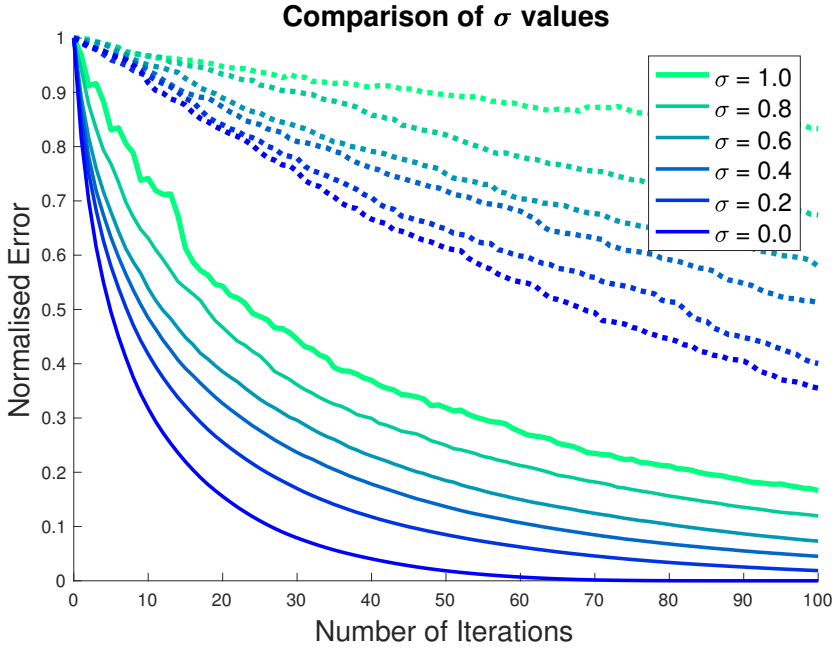


Fig. 6.2. Perfect/Imperfect user tests for Traditional Interface averaged over all 36 goal presets. Perfect Order shown with solid lines, Random Order shown with dotted lines. The parameter order has a much more significant effect than the imperfections of the user.

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* algorithm[1] was used to minimise the error metric for a particular Macro Control, to simulate a Perfect User.

[1] A global search algorithm. For these one-off tests the only goal is to find the minimum, with the convergence speed being irrelevant. The cost function is smooth and only a single parameter is being altered at a time, so *Patternsearch* is appropriate.

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 Fig. 6.3a, and a comparison of the mean results for the different macro configurations is shown in Fig. 6.3b.

Some points to note from these results:

For different preset goals, the effectiveness of the Macro Controls varies significantly (Fig. 6.3a). 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 that a new approach that calculates the Macro Controls per preset is worth pursuing. However, this comes at the cost of the user being less familiar with what the Macro Controls are actually doing (§5.2.2).

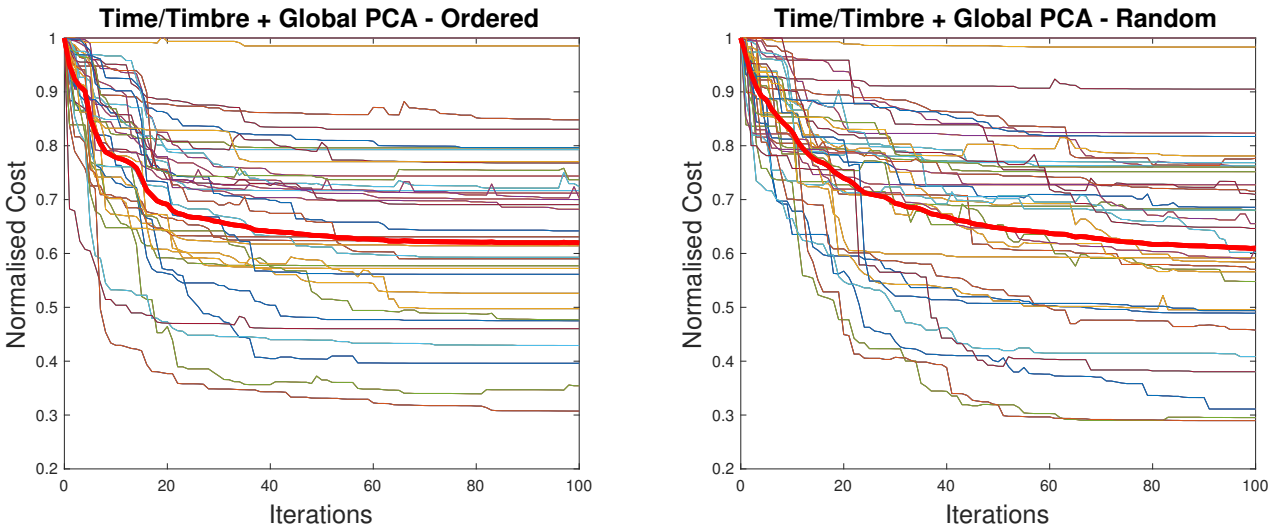
In all cases the mean error converges to a non-zero steady state value, due to the limited degrees of freedom when using the Macro Controls. This means that the Macro Controls alone cannot be used to perfectly match a goal preset.

For numbers of iterations less than 40, the cyclic order decreases the error faster than the random order.[2] Therefore ordering the Macro controls based on their variance is a sensible choice, so that users can easily use this cyclic order.

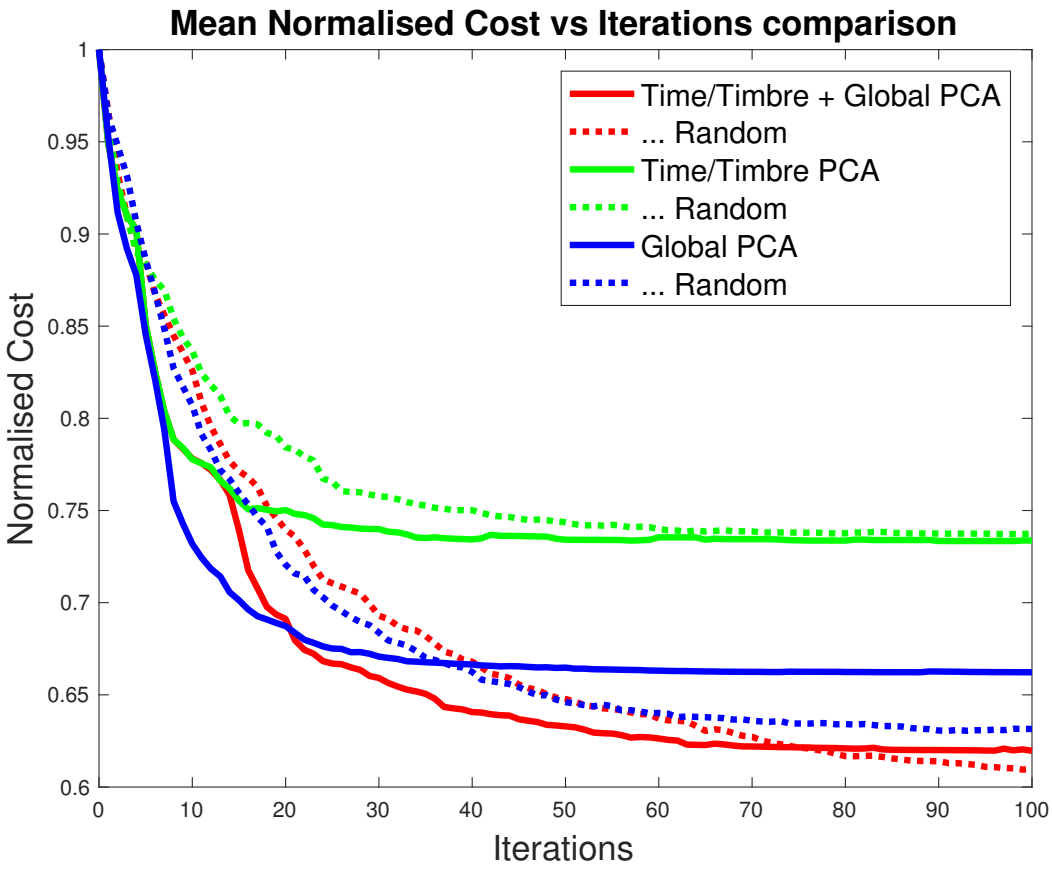
Finally, each of the sets of controls has a similar initial rate of error reduction, but the Global Macro Controls appear to have a significant advantage over the Time/Timbre Macro Controls over larger numbers of iterations. This is likely due to the fact that the Global Macro Controls have access to 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 Macro 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. This allows user to choose between the increased accuracy of the Global Macro Controls, the more semantically meaningful Time/Timbre Macro Controls, or

[2] 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's error. However, It is likely that this is due to the cyclic order causing the search algorithm to get stuck in a local optimum.

Fig. 6.3. Perfect/Imperfect user tests for Selection Interface



(a) Ordered/Random user tests for Selection Interface



(b) 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.

any combination thereof.

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 were identified, and used to initialise preset A, B and C in the blending algorithm (§5.3.1). *Patternsearch* was 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 Fig. 6.4. On average, there is a 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, and some in which the error decreases very quickly, however there is less variance than in the PCA interface. For the Perfect User, error is monotonically decreasing, but this is not always the case for the imperfect user, and for real users of the system. This reinforces the need for the Selection History Plot (§5.3.1), which allows users to select several of the local minima in the error plot to combine together.

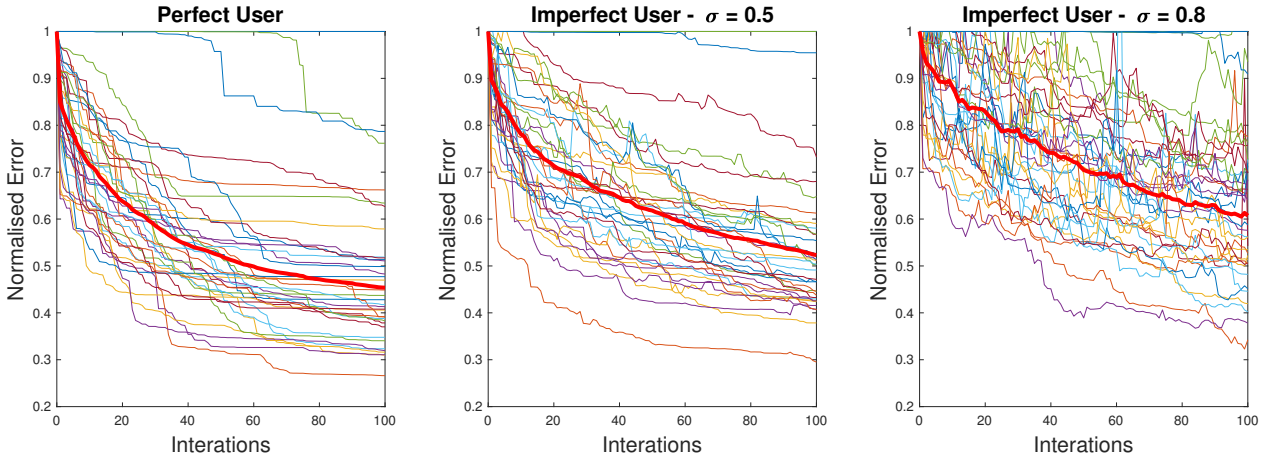


Fig. 6.4. Perfect/Imperfect User tests for Blending Interface

6.4 Comparison of Interfaces

A comparison of the test results from all three interfaces is shown in Figure 6.5.

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 error converge to zero. This result demonstrates that **the Traditional Interface can either 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, and so can extract some useful information from each.

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 of the speed of the Traditional Interface.

Over a larger number of iterations, Blending Interface is supassed by the Traditional 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 to switch to the other interfaces at any point,[3] this may not be an issue.

Before converging to their steady state value, the Macro Controls have a fast rate of error reduction - inbetween the Perfect User with Perfect or Random orders. Part of this speed is because the Macro Contols allow all of the synthesiser parameters to be varied at once. This suggests that PCA mapping of the preset dataset produces principal components that are well suited for use in Macro Controls.

These results suggest that a good workflow for editing synthesis parameters is as follows:

- Find closest three 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 switch 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 synth, an

[3] Giving the user the option of manual controls was shown to be advantageous in [4].

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

6.5 EARS Model Evaluation of Interfaces

The *EARS* model [34] describes four modes of creative cognition, *Exploratory, Reflective, Skilled* and *Algorithmic* (§2.4). The three interfaces are evaluated in terms of this model in Table 6.1. A conclusion to draw from this is that the interfaces effectively enable exploratory, algorithmic, and reflective processes, but are not suited to skilled interactions. To enable 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 [16, 34], and involve moving through a parameter space consisting solely of the useful sounds of the synth. An *interactive machine learning* system such as the *Wekinator* [13] would be well suited to this task, however more work needs to be done to integrate this approach with the other three 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 original 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 the three synthesiser interfaces

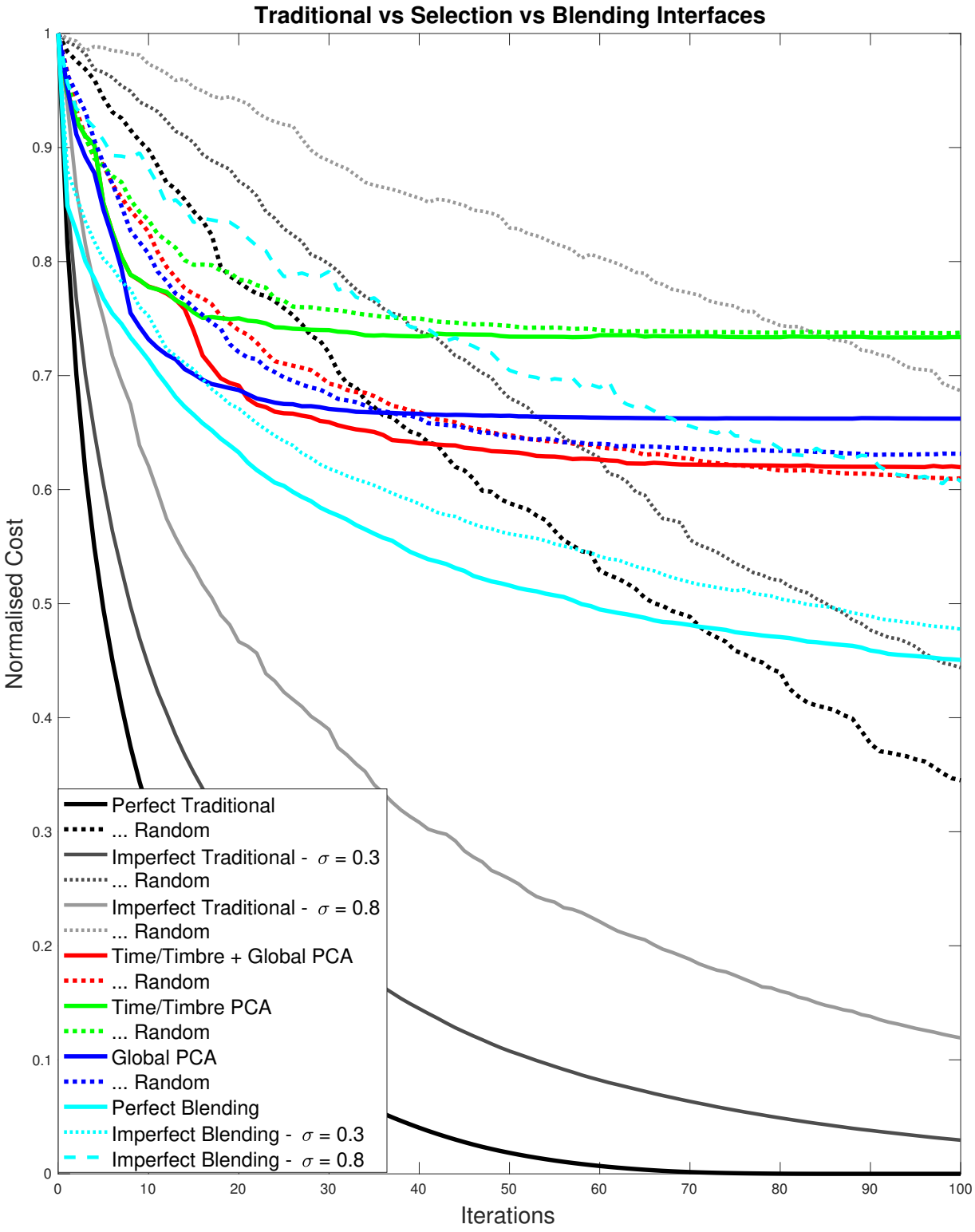


Fig. 6.5. Comparison of Interfaces - The fastest interface is the Traditional Inteface when controlled by a Perfect User, however it is unlikely that real users can ever attain such a fast rate due to the difficulty of selecing the perfect parameter order. The Blending Interface, and Selection Interface both have initial rates of error reduction faster than the Traditional Interface with random order, suggesting that they offer at least as good performance as the Traditional Interface for real users.

6.6 Test of the Interface for Image Filtering

To test the interface in a different context the Blending Interface was adapted to control a simple image filtering algorithm. The results of this process are shown in Fig. 6.6, and the image filtering algorithm is described below.

Two copies of an input image are seperately processed then combined together. The *background layer* is formed by a linear combination of the input image and a greyscale version of the input image. The *foreground layer* first uses MATLAB's *imadjust* functon to adjust the *intensity range* and *gamma* of the input image's RGB channels, then applies a Gaussian shaped *alpha map*. The algorithm has 18 parameters, all in the range $[0, 1]$. A small collection of presets was generated for this algorithm, which the user can select between.

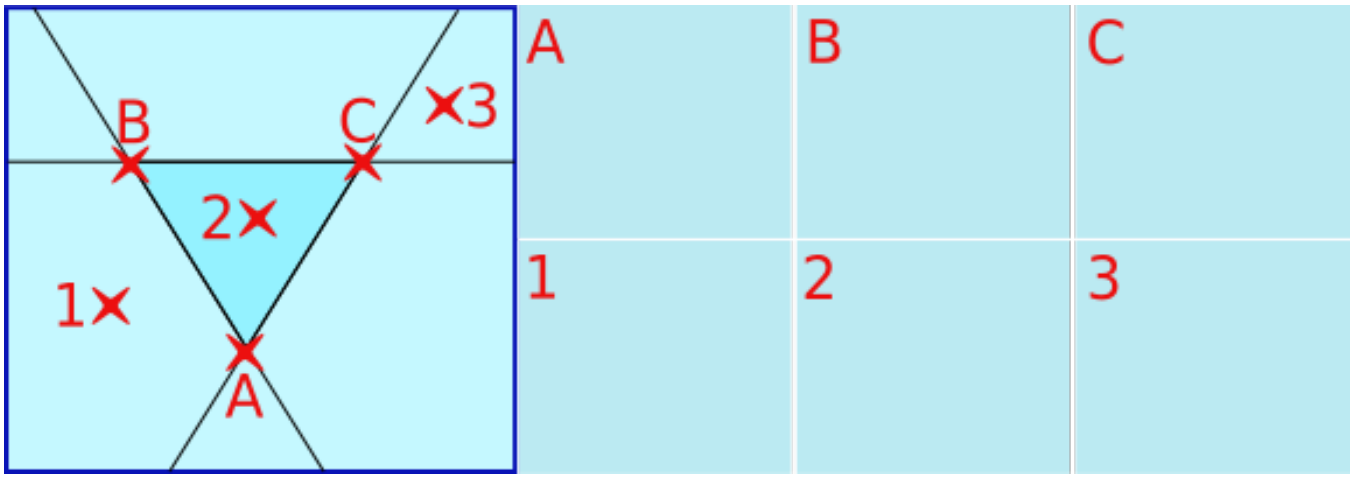


Fig. 6.6. The Blending Interface applied to Image Filtering

As the algorithm does not spatially transform the image in any way, a simple metric (the L_2 norm of the difference between the images[4]) can be used to compare the outputs. This allows a Perfect User test (§6.1) to be carried out on the Blending Interface, however instead of using a parameter based error metric (§6.2) the image based error metric can be used. This simulates a real user in a more realistic manner as the comparison is in the perceptual space rather than the parameter space. Once again, MATLAB's *Patternsearch* algorithm was used to optimise the blend between presets A, B and C. These tests were carried out using a variety of test images, start presets and end presets. This allowed the rates of convergence to be compared when developing the preset generation algorithm (§5.3.2), which was used to set its hyperparameters.

[4] The image can be downsampled first to reduce the computation time.

Chapter 7

Conclusion

This project investigated using *machine learning* to make synth programming more efficient and intuitive. Using *dimensionality reduction* to extract information from a set of synth presets, an interface has been designed which allows users with no knowledge of synth programming to find new sounds by varying and combining presets. The two novel sections of this interface are the Selection Interface (an interactive low dimensional mapping of a synth preset dataset), and the Blending Interface (an iterative approach to combining presets in order to explore the parameter space).

The interface created in this project has many benefits over a traditional synth interface. Based on simulated user studies, the interface performs as good as (and in some cases much better than) a traditional interface when carrying out search based tasks. As it was designed using heuristics from the fields of *Human-Computer Interaction* [16, 38, 15], and *Creative Cognition* [34], and statistical techniques from the field of *Machine Learning* [31, 4], it is more flexible, expressive and powerful than a traditional interface. Feedback about the interface from a small number of users has been positive, however more rigourous user-testing is required.

Several insights have been gained about synth parameter spaces:
The key limitation of traditional synthesiser interfaces is the order of parameter selection (§6.3.1). An iterative preset blending algorithm using a stochastic preset generation algorithm has been shown to be able to effectively navigate synth parameter spaces (§5.3.1). Synth preset datasets are not random, containing strong correlations and internal structure (§5.2.5). *PCA* combined with *histogram equalisation* has been shown to be capable of exploiting this internal structure to perform dimensionality reduction, and to create effective Macro Controls (§6.3.3).

The interface has also been validated in a different context, image filtering (§6.6), with promising results. Popular photo-sharing applications, such as *Instagram* and *Snapchat*, use

preset-based creative image filtering as a key feature. Incorporating the Blending Interface into such applications has the ability to increase the creative control of users, whilst keeping the application simple and intuitive to use. 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 using protocols such as *OSC*.

7.1 Further Work

The evaluation of this interface was only carried out on a single synth, due to the project's time constraints. An immediate goal is to validate the conclusions from this work on several other synths, especially those from other categories, such as *subtractive*, *additive* and *physical modelling* synths. It is non-trivial to extend this work to modular synths (§3.2). To determine the effectiveness of the parameter-based evaluation carried out in this work, and evaluate other aspects of the interface such as creativity, a user study should be carried out.

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 customisable *OSC* controllers *Lemur* or *Mira* could be used. The interface could be further extended by taking into consideration the *multi-touch* capabilities of modern touchscreens.

In the Blending Interface, users carry out a large number of interactions, all of which show preference over hyperplanes of parameter settings. If recorded, it may be possible to use these interactions to form a training set for more sophisticated preset generation algorithms (§5.3.1). As each interaction finds the optimum in a hyperplane, the interactions have the potential to be more information-dense than binary choices. This may allow the use of techniques such as bayesian optimisation, which require large amounts of training data [4].

In typical synths, 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 synths store their presets in the *Javascript Object Notation (JSON)* 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 those 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.

More work needs to be done to create a truly general purpose soft-synth controller. The interface has been designed to be as general purpose as possible, allowing it to control arbitrary synths over the *OSC* protocol, however it can currently only control one custom designed synth. The key difficulty preventing this was a lack of standardisation between soft-synths[1]. 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.

Machine learning is a rapidly developing field with groundbreaking results in a variety of applications [28]. There is a large amount of media attention about the distopian futures which may arise if machine learning algorithms gain a human-like level of intelligence [3]. However, machine learning also has the potenital to augment the human experience, enhancing productivity and creativity of users by acting as a skilled assitant [8], or an *oracle*. This project is an attempt to create such an assistant, and develops the framework for a creative interface to control synths. The interface currently uses simple machine learning algorithms and a small amount of data to acheive this. However, if adapted to control more synths and deployed to a large amount of users, the interaction data collected could allow more sophistochated algoirthms to be used. As the field of machine learning continues to advance, the potential power of such an interface will only increase.

[1] Different soft-synths use a variety of communication protocols and preset storage methods, however efforts are being made to standardise these, such as the *Native Kontrol Standard*.

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] Bostrom, N. (2016). Superintelligence. Oxford University Press.

[4] Brochu, E. et al. (2010). *A Bayesian interactive optimization approach to procedural animation design*. Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '10). Eurographics Association, Goslar, Germany, 103-112.

[5] Clifton, Paul G. et al. (2016) *Design of Embodied Interfaces for Engaging Spatial Cognition*. Cognitive Research 1.1: 24. PMC. Web. [Acccessed 30 Apr. 2018].

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

[7] Cycling 74. (2017). *Max 7* [computer software]. San Francisco.

[8] De Man, B. (2017). *Audio effects 2.0: rethinking the music production workflow*[keynote]. Audio Developer Conference (ADC'17). https://www.youtube.com/watch?v=yJcqNR7by9o [Accessed 1 May. 2018]

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

[10] Dzulkifli, M. A. et al. (2013). The Influence of Colour on Memory Performance: A Review. The Malaysian Journal of Medical Sciences: MJMS, 20(2), 39.

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

[12] 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].

[13] 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).

[14] 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.

[15] 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

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

[17] 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

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

[19] Linn, R. (2016). *Keynote*. Audio Developer Conference (ADC'16). https://www.youtube.com/watch?v=3bHGeSv37rU [at 19:00] [Accessed 20 Apr. 2018]

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

[21] 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

[22] Moogmusic.com. (2015). *System 35 — Moog Music Inc*. [online] Available at: https://www.moogmusic.com/products/modulars/system-35 [Accessed 30 Apr. 2018].

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

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

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

[26] 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].

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

[28] Rotman, D. (2017). *The Relentless Pace of Automation*. MIT Technology Review, March/April 2017.

[29] Roy P.C. Kessels et al. (2001). *Varieties of human spatial memory: a meta-analysis on the effects of hippocampal lesions*. Brain Research Reviews, Volume 35, Issue 3.

[30] Settles, Burr. (2010). *Active Learning Literature Survey*. University of Wisconsin, Madison.

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

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

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

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

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

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

[37] Wang, Z. et al. (2016). *Bayesian Optimization in a Billion Dimensions via Random Embeddings*. arXiv:1301.1942 [stat.ML]

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

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

[40] Engel, J. et al. (2017). *Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders*. arXiv:1704.01279

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

[42] Besbes, O. et al. (2014). *Optimal Exploration-Exploitation in a Multi-Armed-Bandit Problem with Non-stationary Rewards*. arXiv:1405.3316

Department of Engineering Science

UNIVERSITY OF OXFORD

Student declaration and Academic Approval

Student Declaration:

I have completed the DSE Workstation Checklist and the Supplementary Questions for my computer-related risk assessment for 4YP Project Number indicated below:

4YP Project Number: 11394

4YP Student's Name *(please print)* Thomas Simon O'Connor

4YP Student's Signature: 

Academic Approval

I confirm my approval of this 4YP DSE Risk Assessment.

Academic Supervisor's Name: *(please print)* Stephen Roberts

Academic Supervisor's Signature 

Department of Engineering Science

Supplementary Questions for 4th Year Project Students

Risk Factor	Answer	Things to Consider	Record details here
Has the checklist covered all the problems that may arise from working with the VDU?	★ Yes ☐ No		
Are you free from experiencing any fatigue, stress, discomfort or other symptoms which you attribute to working with the VDU or work environment?	★ Yes ☐ No	Any aches, pains or sensory loss (tingling or pins and needles) in your neck, back shoulders or upper limbs. Do you experience restricted joint movement, impaired finger movements, grip or other disability, temporary or permanently	
Do you take adequate breaks when working at the VDU?	★ Yes ☐ No	Periods of two minutes looking away from the screen taken every 20 minutes and longer periods every 2 hours Natural breaks for taking a drink and moving around the office answering the phone etc.	
How many hours per day do you spend working with this computer?	☐ 1-2 ★ 3-4 ☐ 5-7 ☐ 8 or more		
How many days per week do you spend working with this computer?	☐ 1-2 ★ 3-5 ☐ 6-7		
Please describe your computer usage pattern	I work with proper typing technique on properly adjusted monitor and chair. I take breaks every hour to stretch and walk around outside.		