CONTEMPORARY APPROACHES TO LIVE COMPUTER MUSIC: THE EVOLUTION OF THE PERFORMER COMPOSER

BY

OWEN SKIPPER VALLIS

A thesis
submitted to the Victoria University of
Wellington
in fulfillment of the requirements for the degree
of
Doctor of Philosophy

Victoria University of Wellington

2013

Supervisory Committee

Dr. Ajay Kapur (New Zealand School of Music)

Supervisor

Dr. Dugal McKinnon (New Zealand School of Music)

Co-Supervisor

NEW ZEALAND SCHOOL OF MUSIC

ABSTRACT

This thesis examines contemporary approaches to live computer music, and the impact they have on the evolution of the composer performer. How do online resources and communities impact the design and creation of new musical interfaces used for live computer music? Can we use machine learning to augment and extend the expressive potential of a single live musician? How can these tools be integrated into ensembles of computer musicians? Given these tools, can we understand the computer musician within the traditional context of acoustic instrumentalists, or do we require new concepts and taxonomies? Lastly, how do audiences perceive and understand these new technologies, and what does this mean for the connection between musician and audience?

The focus of the research presented in this dissertation examines the application of current computing technology towards furthering the field of live computer music. This field is diverse and rich, with individual live computer musicians developing custom instruments and unique modes of performance. This diversity leads to the development of new models of performance, and the evolution of established approaches to live instrumental music.

This research was conducted in several parts. The first section examines how online communities are iteratively developing interfaces for computer music. Several case studies are presented as examples of how online communities are helping to drive new developments in musical interface design.

This thesis also presents research into designing real-time interactive systems capable of creating a virtual model of an existing performer, that then allows the model's output to be contextualized by a second performer's live input. These systems allow for a solo live musician's single action to be multiplied into many different, but contextually dependent, actions.

Additionally, this thesis looks at contemporary approaches to local networked ensembles, the concept of shared social instruments, and the ways in which the previously described research can be used in these ensembles.

The primary contributions of these efforts include (1) the development of several new open-source interfaces for live computer music, and the examination of the effect that online communities have on the evolution of musical interfaces; (2) the development of a novel approach to search based interactive musical agents; (3) examining how networked music ensembles can provided new forms of shared social instruments.

ACKNOWLEDGEMENTS

The author wishes to express sincere appreciation to all those who have helped with the realization of this thesis. The work that follows is the culmination of the last ten years of my life, and was made possible by the many artists, colleagues, and friends who have inspired and helped me along the way.

Ajay Kapur, thank you first and foremost for your exceptional tutelage in not only my research, but also my life. I would not be here today if it was not for your insight, advice, mentorship, and motivation. You have the amazing gift of seeing the potential in those around you, and the genius to help us realize that potential.

Jordan Hochenbaum, thank you for all the projects, music, and art that we have worked on over the last few years, for all the incredible projects we will work on in the coming years, and most of all for being an incredible friend. Our work as Flipmu has been a source of many of the ideas in this thesis, and I look forward to all the coding, installation, music, and general madness that are to come. You are an incredible musician, amazing artist, and constant source of inspiration.

Many thanks to Dugal McKinnon, Nick Collins, Michael Norris, Martijn Zwartjes, Jim Murphy, Brad Hill, Tim Exley, Jason Edwards, Michael Darling, and Johnny McClymont for their help, ideas, and advice along the way. I would not have been able to complete this thesis without you sharing your expertise and thoughts.

Much of this thesis is the direct result of my experiences performing with other musicians in live computer music ensembles. Thank you to The KarmetiK Machine Orchestra; whom's broad musical vision allowed me to perform both with incredible musicians, and a shared robotic instrument. Thank you to Trimpin, Curtis Bahn, and Tomie Hahn for sharing your incredible work. Your

ideas have helped to inspire my own efforts. Thank you to Jeffery Lufkin for being a part of the genesis for many of the ideas found in this thesis.

Lastly, thank you to my family for their support throughout this journey. To my parents for their encouragement and support; to my incredible and loving wife Liv, for being my source of strength and my muse; and to Rory, for being an inspiration for my writing, and helping me through many challenging discussions, thoughts, and edits.

For any of my friends who I have most regrettably left out of these acknowledgments, please know that you have my deepest gratitude for your help.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1

1.1 INTERACTION CONTEXTS 2

1.2 PERSONAL MOTIVATIONS 5

1.3 THESIS OVERVIEW 7

CHAPTER 2 COMMUNITY BASED DESIGN: ITERATIVE MUSICAL INTERFACE DEVELOPMENT 11

2.1 GOALS AND MOTIVATION 12

2.2 INTERFACE DEVELOPMENT PRIOR TO ONLINE COMMUNITIES 13

2.3 COMMUNITY BASED DESIGN 15

2.4 ITERATIVE DEVELOPMENT 16

2.5 BACKGROUND CASE STUDIES ON THE MONOME 16

2.5.1 Monome 17

2.5.2 The computer musican as digital luthier 22

2.6 NEW WORK: ARDUINOME AND CHRONOME 22

2.7 MONOME, TENORI-ON COMPARISON 28

2.8 DISCUSSION 29

CHAPTER 3 ARMY OF ME: AUTONOMOUS AGENTS AND THE SOLO PERFORMER 33

3.1 GOALS AND MOTIVATION 34

3.2 BACKGROUND 35

3.2.1 Twentieth century composers 36

3.2.2 Computer aided algorithimic composition 38

3.2.3 Interactive musical agents 40

3.2.4 Contemporary systems 41

3.3 DEFINE THE CHALLENGE 42

3.4 IMPLEMENTATION: LIVE PERFORMANCE SYSTEMS 43

3.4.1 Search based systems 44

3.4.2 Considerations for use with improvisation 45

3.4.3 S2MP: a similarity matching algorithm 46

3.4.4 Training the system, and linking controller data 47

3.4.5 Implementation of the system for use in performance 49

3.4.6 Plugin design 53

3.4.7 Analysis 55
3.4.8 Challenges with using search based systems 64
3.5 DISCUSSION 65
3.5.1 Architecture of an interactive musical agent 65
3.5.2 The army of me 67

CHAPTER 4 THE ART OF COMMUNICATION: SHARED INSTRUMENTS AND NETWORKED MUSICAL ENSEMBLES 69
4.1 BACKGROUND 72
4.2 PHYSICALITY IN COMPUTER MUSIC PEFORMANCE, AND EXTENDING SHARED CONTROL TO MUSICAL ROBOTICS 74
4.3 MUSICAL ROBOTICS AND THE KARMETIK MACHINE ORCHESTRA 76
4.4 COMPOSITIONS AND PERFORMANCES 78
4.4.1 January 27, 2010 REDCAT - The Machine Orchestra 78
4.4.2 August 14, 2010 – Karmetik Collective 82
4.4.3 April 12, 2012 REDCAT – Samsara The Machine Orchestra 84
4.5 DISCUSSION 86

CHAPTER 5 CONCLUSION 89
5.1.2 Improvisation in live computer music 94
5.2 CONTRIBUTIONS 96
5.2.1 Online Community based iterative design and the Chronome 96
5.2.2 S2MP and an interactive system for continuous control 97
5.2.3 Shared social musical robotics 98
5.3 FUTURE WORK AND PHILOSOPHY 98
5.3.1 Bridging the gap between performer and audience 99
5.3.2 Final thoughts 102

APPENDIX A RELATED PUBLICATIONS 105

APPENDIX B CHRONOME TECHNICAL FILES 107

APPENDIX C COMPARATIVE SURVEY OF LOCAL NETWORK ENSEMBLES AND SOLO LIVE COMPUTER MUSIC 121

APPENDIX D PROBABILITIES AND MARKOV MODELS 131

APPENDIX E SEARCH-BASED ALGORITHMS 145

APPENDIX F REGRESSION SYSTEMS 151

APPENDIX G COMPARISONS AND REQUIREMENTS 157

BIBLIOGRAPHY 167

LIST OF FIGURES

Figure 1: Tools used in the evolving roles of the performer composer 7
Figure 2: Overview of the thesis layout 8
Figure 3: Iterations of the Monome 17
Figure 4: Commercially developed Native Instruments Maschine, Akai APC 40, Novation Launchpad, and Ableton PUSH 21
Figure 5: The first Arduinome. Built using Monome 40h buttons 24
Figure 6: Chronome prototype 26
Figure 7: BrickTable version I, II, and III 27
Figure 8: Monome left and Tenori-On right 28
Figure 9: Interactive musical agent for modeling continuous control data from musical interfaces 35
Figure 10: Controller data from both performers stored as 2-D and flattened 1-D collections 47
Figure 11 Search algorithm training algorithm. The two performers are sampled by the plugin, and performer B's current item set is linked with Performer A's previous item set. 48
Figure 12: Stored performances for S2MP algorithm. Sequential item sets for a performance are stored as $Seq_{2(n)}$. Each j^{th} item set represents both Performer A and B, and has a MIDI buffer associated with it. 49
Figure 13: Constraining the S2MP search to current position within a bar. This represents a good compromise between efficiency and search as it cuts the number of compares and is likely to return good matches. 50
Figure 14: Mapping score matrix. The intersection is taken between every i^{th} item set in Seq_1 and the j_{th} item set in the current sub-sequence from $Seq_{2(n)}$. 51
Figure 15: Performance diagram of search algorithm. Sampling Performer A along with the current agent output generates the input sequence. This is then compared against sub-sequences within the database, providing similarity scores. The MIDI buffer is taken from the sub-sequence with the highest similarity score, and then appended to the plugin's MIDI output. 53
Figure 16: Continuous Control S2MP plugin 54
Figure 17: Parsing MIDI buffers into sample windows 54

Figure 18: Routing setup for S2MP training in Ableton Live, and MIDI CC training data 56
Figure 19: Target CC sequence (top) vs. Output sequence (bottom). The output sequence didn't match the target sequence in the first bar, but otherwise was a perfect match. 56
Figure 20: Initial S2MP plugin test - The training sequence length was increased from one to four bars 57
Figure 21: Similarity percentage between output sequence and target sequence based on Mapping / Order weighting 59
Figure 22: Mapping / Order - Average distance of output sequence from the target sequence 60
Figure 23: Factor increase of discontinuities between output and target sequence 61
Figure 24: Similarity percentage between output sequence and target sequence based on number of item sets in the input sequence, and number of trained transitions 62
Figure 25: Increasing numbers of item sets - average distance of output sequence from the target sequence for input sequences 63
Figure 26: Factor increase of number of discontinuous matches by sequence size 64
Figure 27: Different design approaches for interactive musical agents. Clockwise from top: (A) All inputs affecting the model's output; (B) The model is only affected by itself, and live input is applied as a fitness function; (C) Inputs are split into simpler individual models, all acting independent of each other 66
Figure 28: Network topology of The Machine Orchestra ensemble 70
Figure 29: The League of Automatic Composers 1980 72
Figure 30: View of the marimbas from the musical robot Tammy 76
Figure 31: The Machine Orchestra at REDCAT 2010 79
Figure 32: The Machine Orchestra performing Samsara 2012 85
Figure 33: The Monome can be both highly programmable or immediately usable 90
Figure 34 Preference performing solo computer music vs. networked ensembles 122
Figure 35: A Markov Model representing the transition probabilities for the set of notes C through A. The notes on the left represent the source states, and the notes along the top are destination states. The values in the matrix are the transition probabilities for moving from a source state to a destination state. 133
Figure 36: Algorithm for determining the next destination state in a Markov Model given a row of transition probabilities 134

Figure 37: 2nd order Markov Model 135
Figure 38: 2nd order Markov Model shown as 1st order Markov Model 135
Figure 39: The description of a transition probability, containing the Source State, Destination State, and the number of times the transition has been observed during training 137
Figure 40: Markov source states stored as tree structure. This allows for searching variable length source state sequences. 137
Figure 41: Longest matching source state sequence for input sequence F G D C ... 138
Figure 42: Longest matching source state sequence for input sequence G A D C .. 138
Figure 43: Training diagram of Markov Model 141
Figure 44: Performance diagram of Markov Model 142
Figure 45: Input to Markov Model that does not match any previously seen source state 143
Figure 46: Matching sequences return the next stored state form the database..... 146
Figure 47: Matching sequences return values from a second linked sequence..... 147
Figure 48: Matrix with ideal rating scores comparing input sequence along the columns, and scored sequence down the rows 148
Figure 49: Insertion test: this tests for notes played by the performer that are not in the original score 149
Figure 50: Deletion test: this tests for notes from the score that are skipped over by the performer 149
Figure 51: Substitution test: this tests for notes substituted by the performer..... 149
Figure 52: Repetition test: this looks for notes from the score that are repeated by the performer 149
Figure 53: Constrained search to speed up sequence comparison 150
Figure 54: Basic linear regression 152
Figure 55 Similarity between linear regression and basic sequential ANN 152
Figure 56 Diagram of multi-variate linear regression model 154

LIST OF TABLES

Table 1 Reasons for solo or group performance preference 124
Table 2 Descriptions of each musician's roll within the ensemble 125
Table 3 Description of new performance modes afforded by computer ensembles 127
Table 4 Descriptions of the challenges of performing in computer ensembles 129
Table 5 Overview of algorithms for designing interactive musical agents 164

Chapter 1

Introduction

> "… although the actions of the traditional acoustic musician are familiar to an audience, the attribution of human agency to a computer may be more problematic. Schloss (2003) fears an excess of 'magic'; however, the enculturation of electronic music may automatically make new processes acceptable (Nick Collins 2003). Ultimately, the balance between innovation and tradition might be best served by a conception of 'the possibility of a music of technology with the clear imprint of the human will rather than the human presence' (Emmerson 2000)."
>
> — Nick Collins (2006)

Live music is a social art that allows humans to come together and share a collective experience. This experience is not only comprised of the sounds we hear at these events, but also elements as diverse as: a performer's virtuosity, and compositional skill; social interaction between the musicians performing, and between the musicians and the audience; the social identity that a specific event can imply about an individual; and the potential to experience something both exceptional, and ephemeral. The gestalt of this experience is a complex socio-musical interaction between musicians and the audience, made all the more complex by individual musical styles placing emphasis on different elements of these experiences.

While live computer music adheres to some of these existing expectations, it also provides the opportunity to create entirely new elements, and re-evaluate the

importance of others. For example, a computer's ability to automate and endlessly repeat a task creates an opportunity to expand the expressive potential of a single musician. The computer's ability to automate tasks enables a musician to simultaneously control more than one instrument; however, this "out-sourcing" of musical control may make the human agency within a performance difficult for an audience to discern.

So, what then is the role of human musicians in live computer music? Are we to be conductors, or informed musical selectors, who merely point our computers in a musical direction, leaving the details of the sound to the machines? Alternatively, are we required to adhere to existing expectations of live acoustic music, and only play those sounds that we can physically produce or actuate?

This thesis proposes that neither of these extremes are the answer. Instead, live computer music has revolutionized what it means to be a performer composer by providing a unique opportunity for performing musicians to simultaneously exist in many different roles. In order to realize live computer music in this way, current research is examining the augmentation of the computer as an instrument. How do we interface with our computers during performance? How do we musically leverage their computational power to expand what we can do as individual musicians? How do we perform with these systems as groups, or ensembles?

1.1 Interaction Contexts

The different roles that live computer music affords performers is key to understanding these questions, and to understanding why live computer music is an evolution of the performer composer. This section examines existing attempts to understand and describe the different roles that musicians play during a performance. The aim of this section is to derive a taxonomy that will describe these different performance roles, and allow for comparison and contrast between live acoustic music and live computer music.

Some of these questions have been addressed when the field of live computer music crosses paths with human computer interaction (HCI) research. Serji Joda makes the case that music performance's high bandwidth makes it a fertile ground for examining the way in which humans interface with computers (Jordà et al. 2007). He describes how music requires "a very precise temporal control over several multi-dimensional and continuous parameters, sometimes even over simultaneous parallel processes". He goes on to describe that while traditional instruments require the performer to physically control these many different parameters, digital instruments allow for the human to instead "direct and supervise the computer processes which control these details". A system like this allows a human musician to perform in several different contexts: playing low-level details like notes and timbre control, or higher-level control such as effects or score-level events.

Research has also looked at how musicians control these complex instruments within the context of *cybernetics*, i.e., the study of control and communication (Pressing 1990). A cybernetic view of traditional instruments would show the transfer of information between the human musician and the acoustic instrument as being dependent on the energy within a physical gesture. Pressing shows this dependency as a closed loop with the body actuating an instrument's interface, the instrument producing sound, and finally the ears feeding the sound back to the human performer. He describes this as a "one-to-one response between actions of the performer and the resulting sound", calling it a stimulus-response model. Pressing then describes how electronic instruments provide a different model that focus on the processing, shaping or effecting of either the sound or control source. This implies a distancing, or more diffuse mapping between a musician's physical actions and the sound being produced by the instrument. This model of instrument interaction relies on the idea of human musicians supervising or influencing a musical system.

Musical interfaces that afford these types of models are described as "composed instruments" (Schnell and Battier 2002). The composed instrument is defined by the decoupling of the "sound producing part, and the gestural performance

part". Schnell suggests that this creates a representational system, which links the human performer to a set of complex algorithms in the computer. He states, "composers use the representational nature of the system to define events, write scores and specify the computational and algorithmic layers while performers can apply gestural controls and adjust parameters". This thesis argues that the contemporary computer composer is also increasingly the performer. This would imply that contemporary computer musicians define a musical system, write a piece of music, and perform all aspects of that music both at the lower event level, as well as at higher representational levels.

These different levels of control can be seen as a taxonomy of interaction contexts, with modern computers affording musicians the ability to fluidly shift between them, creating a rich and expressive improvisation space. Wanderley and Orio describe seven of these interaction contexts, with the first three having relevance to the performance spaces discussed so far:

> "1. Note-level control, or musical instrument manipulation (performer-instrument interaction), i.e., the real-time gestural control of sound synthesis parameters, which may affect basic sound features as pitch, loudness and timbre.
>
> 2. Score-level control, for instance, a conductor's baton used to control features to be applied to a previously defined—possibly computer generated—sequence.
>
> 3. Sound processing control, or post-production activities, where digital audio effects or sound spatialization of a live performance are controlled in real time, typically with live-electronics (Wanderley and Orio 2002)."

These levels of control provide a coarse description of the different contexts in which a computer musician performs, with the potential for computer musicians to occupy all three of these interaction contexts simultaneously. Indeed, combinations of these states can be seen in other live computer music taxonomies (Croft 2007). A computer musician may be playing an instrument, note-for-note, while also allowing a program to generatively process input of the

sound from a microphone. Finally, through the push of a single button the computer musician can load a new representational system, and instantly change sounds, instruments, and generative processes. This would be similar to every member of an orchestra playing their instrument, while being able to conduct their own small section of the ensemble, and simultaneously effecting the sound reinforcement within the performance space.

1.2 PERSONAL MOTIVATIONS

The following section presents my motivations for undertaking this research, and provides some context for the work that follows. This thesis argues that while live computer music is related to existing forms of live acoustic music, its unique use of interaction contexts constitutes an evolution of the performer composer. This argument has emerged over the course of my own experiences in performing live computer music, as well through conducting the various research projects described in this thesis.

While my background in composing electronic music started around 1998, it wasn't until I attend CalArts in 2005 that my first attempts at live computer performance were made. The aim of these early attempts was to perform live computer music with as much detail and complexity as my fixed compositions or tape music pieces, while simultaneously allowing for the kinds of improvisation and musical dialogues that I experienced in acoustic instrumental performance. This goal has proven to be extremely challenging, and remains the focus of much of my work.

My initial attempts at performing improvisational live computer music strived to achieve the same level of complexity and density as my fixed media pieces. It soon became clear that it was unrealistic to improvise music using only the note-level context, and expect it to be as detailed as a fully realized, multi-part composition that took hours, days, months, or even years to compose. That level of complexity required many parts, and would necessitate an ensemble of musicians to perform using acoustic instruments. However, even an acoustic

ensemble would find it difficult to improvise multi-part music without knowing each of the other performers extremely well. This familiarity with each other as musicians and performers, equates to being aware of the basic ideas and musical styles that might be performed. In other words, successful ensemble improvisation in part relies on having a prior understanding of the potential musical space, and not on the immaculate conception of a fully formed piece of music.

With this in mind, I began to develop tools that used prerecorded material as the musical space, and enabled improvisation in not only the note-level context, but also the sound processing, and score-level contexts. This essentially allowed note level interactions on score level musical material, in the sense that the prerecorded material could be thought of as sections of a composition. This process feels very different from re-ordering a written score in that it alters both time and timbre, and is capable of creating wholly new musical ideas through the reuse of existing musical material. This ability to use existing music to create new music is similar to the DJ remix, except that the process is happening live and can therefore allow for improvisation. Performing in this way enabled me to explore musical ideas that had not been possible when I played acoustic instruments, and I began to see an evolution of the performer composer.

1.3 THESIS OVERVIEW

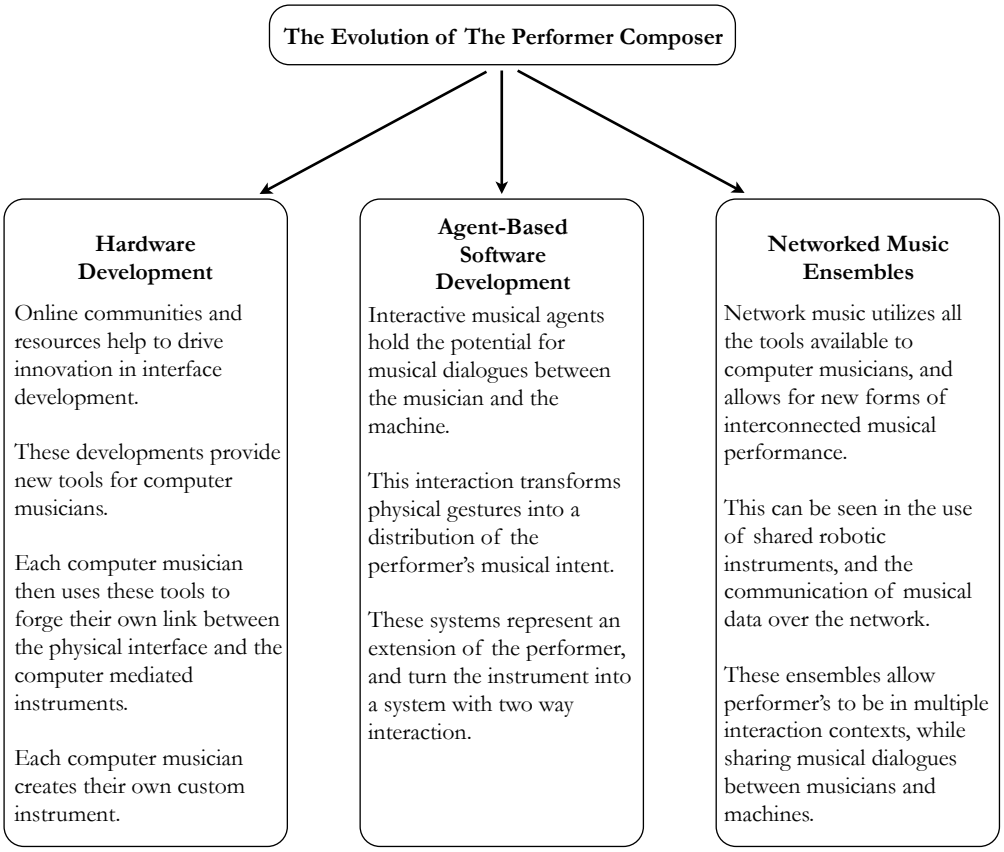


Figure 1: Tools used in the evolving roles of the performer composer

The following chapters discuss the development of tools for live computer music that are key components in the evolution of the performer composer, and discuss how these tools enable new ways of navigating the previously mentioned interaction contexts. These Chapters present the primary contributions of this thesis, and are presented as separate projects (see Figure 2). Although these projects are related, and together form the basis of the argument that live computer music has caused the performer composer to evolve into something new, they are separate enough to warrant being dealt with individually. With this in mind, the following chapters each contain a separate history section. As the topics covered by this thesis are broad, it is my hope that by organizing the thesis in this way, the information presented will be relevant to each section and increase the overall readability of the thesis.

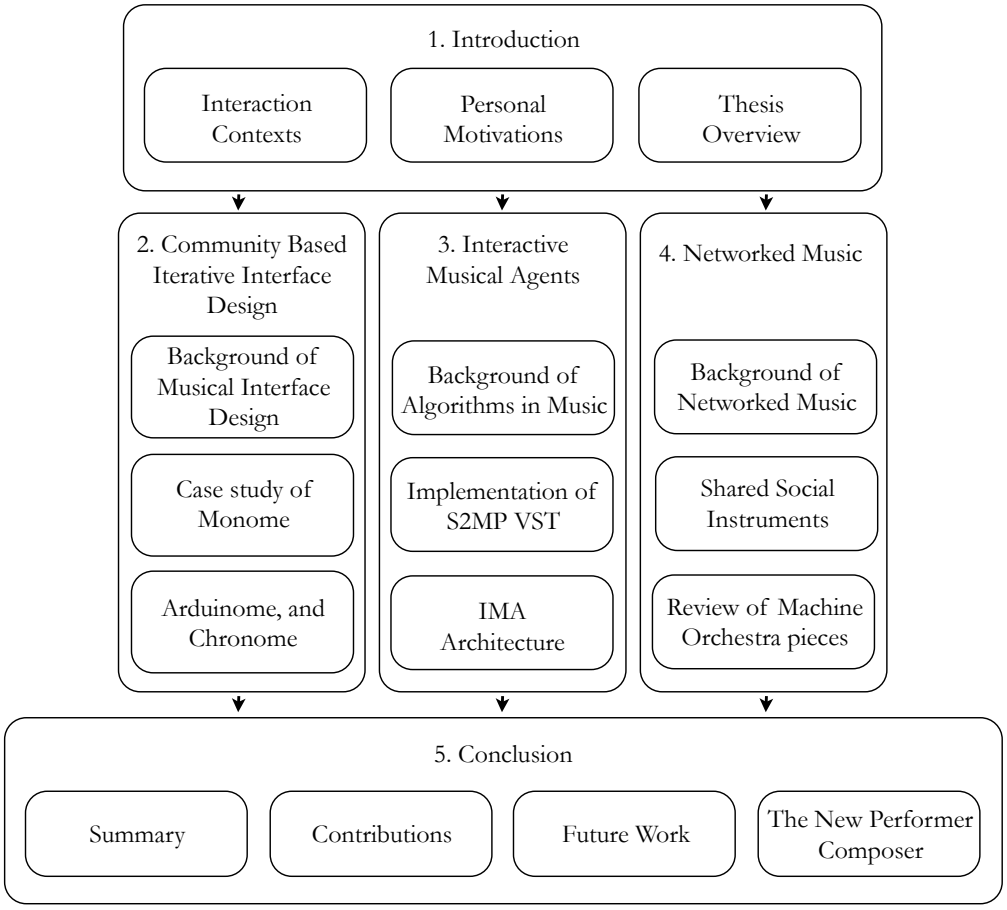


Figure 2: Overview of the thesis layout

Musical interfaces represent the bridge between a human musician and the virtual computer instrument. Each computer musicians is capable of creating a unique mapping for their interface that defines how they will use the different interaction contexts. Chapter 2 examines the iterative development of new interfaces for live computer music, and the impact that online communities who share information about these interfaces has on innovation and the spread new ideas. The Monome is presented as an example of this process.

Chapter 3 discusses the development of systems that can help extend the influence of single actions from a computer musician. These systems can help to distribute the performer's musical intent into the computer along multiple paths, creating new forms of improvisation, and furthering the evolution of the performer composer. These types of systems are described as interactive musical agents, and hold the potential to act as extensions of computer musicians, or

virtual representations of their personas. This chapter also presents a novel approach to search based interactive musical agents by extending existing work with similarity-matching algorithms.

Chapter 4 examines performing as a local networked music ensemble, and the use of shared social robotic instruments. Several pieces by The Machine Orchestra (A. Kapur et al. 2011) are described, and illustrate how these ensembles afford both the more traditional socio-musical interaction found in acoustic ensembles, and the use of interaction contexts presented earlier.

This is followed by a concluding chapter that presents a summery of this thesis' main contributions, examines the new performer composer and what it means to improvise as a live computer musician, and discusses the relationship between the performer and the audience.

Chapter 2

COMMUNITY BASED DESIGN: ITERATIVE MUSICAL INTERFACE DEVELOPMENT

> "The controller is the first component of the digital instrument chain. Controllers constitute the interface between the performer and the music system inside the computer, and they do so by sensing and converting continuous and discrete analog control signals coming from the exterior into digital messages or data understandable by the digital system."
>
> —Serji Jorda (2005)

Like their acoustic instrument counterparts, interfaces are the physical component of the live computer music instrument; however, interfaces are also fundamentally different from acoustic instruments in that they are not the sound-producing agent themselves, but rather the translator between the physical action of the musician and the sounds generated by the computer. This decoupled relationship between sound actuator, and sound generator, allows musicians to map a physical gesture to any sound they wish to control. These custom mappings are an essential component of live computer music, enabling the performer to have control over a multitude of virtual instruments, and play an integral role in the evolution of the performer composer. In order to further this exploration of custom mappings between interface and computer, it is important that computer musicians have resources available to them for developing and customizing their interfaces; the advent of online communities

such as Arduino (Banzi 2008) and Monome[1] has now made such resources available.

This chapter discusses how online communities have changed the way in which interfaces for live computer music are designed and developed, and how this change has led to an iterative development process that adds new functionality to existing interfaces. The chapter begins by describing the evolution of hardware interface design for computer music, and how the emergence of online communities has altered the development cycle of these types of interfaces. The idea that public access to information at these websites has lead to a community driven iterative approach to interface design is then presented in a case study of community-based design which examines the Monome interface and derivative interfaces, its design cycles, and the different roles that people take on within the online Monome community. Following these examples will be the presentation of two new interfaces: the Arduinome, created by the author in collaboration with Jordan Hochenbaum, Brad Hill, and Ben Southall; and the Chronome, created by the author. Both of these interfaces are themselves derivatives of the Monome, and are examples of iterative interface development stemming from online communities. Finally, a comparison between the Monome and the Tenori-On will be presented to explore the differences between interfaces tied to online communities, and interfaces developed by commercial vendors.

2.1 GOALS AND MOTIVATION

In my own practice of live computer music, it has been necessary to develop custom iterations of existing interface controllers in order to achieve specific musical interactions. Basing the design of these interfaces on existing devices expedites the implementation of the technology, in turn allowing for a greater focus on musical performance practice. Additionally, making the modifications available online allows other individuals to further modify or re-contextualize these instruments. This process of modifying an interface, and then providing

[1] Monome - http://monome.org/

online technical information about the changes, has greatly impacted my own implementation of musical interfaces for live computer music, often presenting surprising new modes of use.

The aim of the following research is to examine online communities, and the impact these have on the development of new interfaces for live computer music.

2.2 INTERFACE DEVELOPMENT PRIOR TO ONLINE COMMUNITIES

Performing live computer music requires an interface between the human musician and the computer that is creating the sound. Even the act of playing back an audio file requires the use of a number of physical and virtual interfaces. These interfaces represent an opportunity to explore new mappings between physical actuators and sound engines. These mappings can be realized in many different ways, including live-coding (N. Collins et al. 2003; Wang and Cook 2004), extended laptop instruments such as Hans Koch's piece *bandoneonbook*[2] and the framework Small Musically Expressive Laptop Toolkit (SMELT) (Fiebrink et al. 2007), performing with an external interface (Cook 1992; Mathews and Schloss 1989), or using the computer solely as a sound generating device, or data router to external musical robotics (Kapur 2008). Although the computer itself provides an existing interface in the form of a screen, a keyboard, and a mouse, it certainly does not represent the ideal tool for leveraging the human body in live computer music performance. New media artist Golan Levin even went so far as to say that, "the mouse is an extremely narrow straw through which to suck all of expressive human movement"(Levin 1999). Computer musicians such as Max Matthews, with the Radio Baton (Mathews and Schloss 1989), Nicolas Collins in his work with the Trombone Controller (Nicolas Collins 1991), Michel Waisvisz with the Hands (Krefeld and Waisvisz 1990), Dan Trueman with the BoSSA (Trueman and Cook 2000), Perry Cook with the

[2] Hans Koch - http://hans-w-koch.net/performances/bandoneonbook.html

SqueezeVox (Cook and Leider 2000), Curtis Bahn with the sBass (Bahn and Trueman 2001), Joe Paradiso with gestural sensors (J. Paradiso 2004; J. A. Paradiso 1999), and Sergi Jorda with the Reactable (Jorda et al. 2005), all have created musical instruments that explore the different ways in which a physical interface can be map human actions to computers. Through these mappings interfaces may extend the creative potential of existing instruments, and/or provide entirely new forms of physical interaction with sound.

Traditionally, these experiments in interface design and parameter mapping were shared at institutions such as MIT's Responsive Environments Group, [3] Amsterdam's Studio for Electro Instrumental Music (STEIM), [4] Stanford's Center for Computer Research in Music and Acoustics (CCRMA),[5] UC Berkley's The Center for New Music and Audio Technologies (CNMAT),[6] Princeton's Soundlab[7] and France's IRCAM[8]. In an effort to provide a common space for these separate research institutions to come together, The International Conference on New Interfaces for Musical Expression (NIME)[9] was founded on 1 April 2001. With the establishment of NIME, research into new musical interfaces coalesced into a global community focused not only on building new interfaces but also on examining how to make better ones (Cook 2001; Cook 2009; Arfib, Couturier, and Kessous 2005; Van Nort 2009), as well as how to evaluate their effectiveness and potential (Kiefer, Collins, and Fitzpatrick 2008). The NIME community built off well-established methodologies developed in design fields such as human-computer interaction (HCI) (Drummond 2009; Fiebrink et al. 2010), design theory (Birnbaum et al. 2005; Malloch et al. 2006) and tactile feedback for performers also known as haptics (Berdahl, Steiner, and Oldham 2008). While the research conducted at these institutions and

[3] MIT - http://media.mit.edu/resenv/

[4] STEIM - http://steim.org/steim/

[5] CCRMA - https://ccrma.stanford.edu/

[6] CNMAT - http://cnmat.berkeley.edu/

[7] SOUNDLAB - http://soundlab.cs.princeton.edu/

[8] IRCAM - http://ircam.fr/

[9] NIME - http://nime.org/

conferences of interface design is extensive, prior to the emergence of online communities, publicly available information was primarily static, and artists outside of these academic circles had no global space in which to easily interact and discuss these ideas. However, all of this has changed with the advent of online communities. Both the ways in which musical interfaces are developed, and the ways in which individuals participate in the development process, are now intertwined with online access to information.

2.3 COMMUNITY BASED DESIGN

Online communities accelerate the development cycle of new musical interfaces, allowing an interface to rapidly evolve over a number of iterations. These iterative development cycles are made possible through public forums, and access to information.

Community forums provide a space for artists to share design ideas, and to discuss the different ways in which they use the interfaces. This public interaction provides important feedback to developers, and allows for the way in which the interface is being used during performances to influence the development of future iterations. Forums also provide a space for novices and experts to come together. This allows novices to learn from the accumulated wealth of knowledge provided by community experts, acting as a public educational resource.

Communities also provide educational resources through public access to technical information. Websites provide a centralized repository of information in the form of source code, wikis, and through searching archived forum threads. All of these resources provide access to current technical information, as well as an archived history of the development of the interfaces. New developers can use this information to create and modify existing interfaces, thereby introducing new ideas and functionality back into the community.

2.4 ITERATIVE DEVELOPMENT

Computer science ideas such as open-source development and version-control systems provide public access to a code repository. These repositories allow individuals to learn from the information, or change the information to alter the functionality of the software. With the maturity of microcontroller platforms such as Arduino, analogous ideas within hardware development have become a reality and can now be applied to the development of new musical interfaces.

These ideas have allowed for the iterative design of musical interfaces to take place. Online access to schematics, firmware and software provides the information necessary for a group of individuals to augment a device. These new devices can be shared with the online community, and other community developers can in turn use the altered interface as the basis for further augmentations. This iterative process may fork into separate and unique development streams as new functionalities are explored. These divergent iterations may also converge later, combining functionality into new devices that represent a hybrid of components from previous generations.

This development is driven by a small number of "seed" artists who contribute alternate versions of an interface through iteratively modifying the functionality; at the same time, a larger group of artists access information about these interfaces in order to build, customize and implement existing versions in their own projects.[10]

2.5 BACKGROUND CASE STUDIES ON THE MONOME

This section examines the development of the Monome, and Monome derivative interfaces. Each of the interfaces discussed in this section have benefitted from

[10] For more technically detailed information regarding the iterations of the Monome see <http://flipmu.com>; Owen Vallis, Jordan Hochenbaum and Ajay Kapur, "A Shift Towards Iterative and Open-Source Design for Musical Interfaces," In Proceedings of *NIME* (Sydney, Australia: 2010).

an online community providing information regarding design, development, construction, and use. The results of this have been not only new interfaces with additional functionality, but also new mappings between the performer's physical actions and the computer mediated instruments. These changes afford live computer musicians with greater control, enable new methods of simultaneously performing in multiple interaction contexts, and provide tools that further evolve the role of the performer composer.

2.5.1 MONOME

The original Monome serves as a great example of online community-based iterative design. Members of the online Monome community have led the development of numerous clones and derivative devices such as the Arduinome, the Lumi, the Octinct and the Chronome (See Figure 3). These derivative interfaces have added new functionalities to the original Monome interface, and are shared with the online community to provide the basis for future modifications. The following section briefly describes the development history of these iterations.

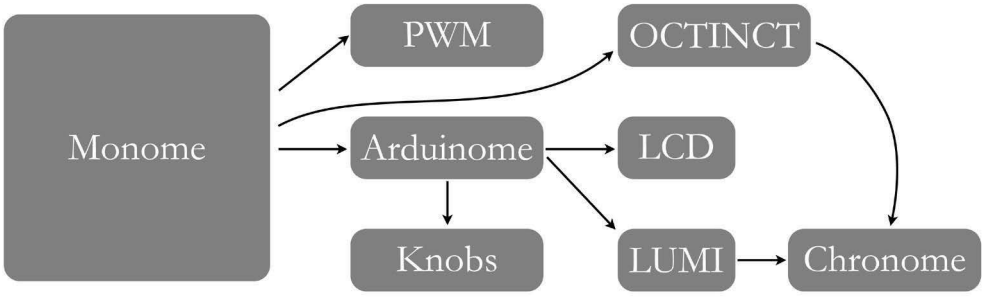


Figure 3: Iterations of the Monome

2.5.1.1 MONOME 40H

Created in 2005 by Brian Crabtree, the original Monome is a two-layer NxN device consisting of a matrix of silicon buttons situated over a matrix of Light Emitting Diodes (LEDs). The Monome's minimal interface allows a user to quickly gain an understanding of how it works. This immediate understanding leads to greater exploration as users begin to augment the Monome's

functionality and thereby increasingly customize their connection---through the interface---to various software instruments.

Upon releasing the original interface, Monome created an online community providing users with a place to discuss and share their custom software programs, and provided open-source access to technical documentation, firmware, and schematics. Monome's decision to provide public access to the original firmware source code, allowed an early user of the Monome 40h to add support for LED brightness control.[11] This change in the firmware represents one of the earliest community modifications to the original Monome, with this feature being officially added to the interface in later versions. Soon after the original Monome was released, the author and collaborators ported the firmware to the open source hardware platform Arduino, providing a new platform on which to modify and hack the interface. This iteration of the Monome was called the Arduinome, and will be discussed in detail in section 2.6.1.1. Other Monome derivative devices such as the LUMI have used the Arduinome as the bases for their own development.

2.5.1.2 LUMI

The LUMI (Gao and Hanson 2009) is an interface consisting of 32 pressure buttons combined with custom software and a touch screen. With the added functionality of pressure data from the 32 buttons, the LUMI constitutes a major change to the original Monome interface. Created at Stanford in 2009, the pressure sensitivity was added by modifying the ArduinomeSerial to OSC convertor, and by implementing a variable pressure sensor using conductive fabric (Freed 2008). In addition, several continuous input devices were added, such as potentiometers, infrared (IR) sensors, and a touch screen. Although this work represents a substantial extension of the Monome's functionality, the project is not fully integrated into the larger Monome community. This could be due to several factors, including custom firmware, custom serial protocols,

[11] "Monome - Per Led Intensity, Video" - http://post.monome.org/comments.php?DiscussionID=913.

unreleased build information or the larger Monome community's unfamiliarity with the work. It is possible that because of these reasons the LUMI's significant modifications have not yet had as broad an impact on the iterative design process as they potentially could. One of the aims of the inclusion of these ideas into the Author's Chronome iteration is to make these modifications more readily available to the larger Monome community.

The LUMI exemplifies the idea of iterative online community based interface design. The developers modified the Arduinome Serial-to-OSC application, which in turn was a modification of the original Monome software. Furthermore, the LUMI developers describe how access to information on both the Arduinome and Adrian Freed's work made, "…it possible to rapidly prototype the interface in the short span of a month" (Gao and Hanson 2009).

2.5.1.3 OCTINCT

Almost as soon as the original Monome 40h interface was released, members of the Monome community began to contemplate the possibility of adding RGB (multi-color) LEDs to the device. The addition of color mapping to individual buttons would create an additional dimension of visual information, allowing performers to map a richer cognitive connection to the controller. One of the first successful iterations to include this was the Octinct, developed by Brad Hill, Jonathan Guberman and Devon Jones. The Octinct information was not initially shared with the Monome community. This stalled the progress of the project, as community developers did not have access to the information needed to build their own Octinct interfaces. In 2008 Jonathan Guberman, who developed the original Octinct firmware, gave Brad Hill permission to make all the code publicly available. Brad Hill has since posted the technical information on his own blog and announced its availability to the Monome community, including making several updates to the firmware and hardware. Most recently, a group of

artists from the Monome community[12] have been collating all related Octinct information and begun to further refine the original design.

Again, the development of the Octinct shows the effect of online communities on interface design. The original Octinct was developed by a small group of talented artists but had no way of reaching a larger user base. Once information about the interface was shared, interested members of the Monome community used it as the basis for further interface development, in turn sharing their own modifications back with the community. Both the original Octinct, and these later modifications, were used as resources by the author during the development of the Chronome (see section 2.6.1.2).

2.5.1.4 COMMERCIAL ITERATIONS

While the creation of the previously described Monome, and Monome derivative interfaces have all been related to the online community, there has also been development of button matrix interfaces from commercial manufacturers. A famous example is the Tenori-on developed in 2005 by Yamaha and artist Toshio Iwai (Nishibori and Iwai 2006). This device is discussed in greater detail in section 2.7. In 2009, Novation released the Launchpad[13] interface, and Akai released the APC 40,[14] both of which featured Monome style grids of push buttons, and have since seen wide spread support from commercial music software such as Ableton. More recently, Native Instruments has added RGB color support to its Maschine interface,[15] and Ableton has released an RGB button matrix interface called the PUSH[16] (see Figure 4).

---

[12] Start:octint [lab] - http://hangar.org/wikis/lab/doku.php?id=start:octint.

[13] Novation Launchpad - http://novationmusic.com/products/midi_controllers/launchpad

[14] Akai APC40 - http://akaipro.com/apc40

[15] NI Maschine - http://native-instruments.com/en/products/maschine/production-systems/maschine

[16] Ableton PUSH - http://ableton.com/en/push

Figure 4: Commercially developed Native Instruments Maschine, Akai APC 40, Novation Launchpad, and Ableton PUSH

While it is difficult to confirm that the devices developed by the Monome community have directly inspired these commercial interfaces, it is hard to believe that it is merely coincidence that several companies would spontaneously begin to make these Monome style grid controllers. This is not to sound negative about the commercial availability of these devices, in fact, quite the opposite. The development of these devices has not only made these interfaces more readily available to musicians, but has also added new functionality such as dual color LEDs from the Launchpad and the APC40, RGB LEDs from the Maschine and PUSH, and continuous control sources from almost every interface. Additionally, members of the Monome community have made translator software, enabling these devices to be used with software developed for the original Monome. This process allows musicians who use these

commercial interfaces to participate in the Monome community, sharing their ideas, opinions, and experiences with others.

2.5.2 THE COMPUTER MUSICAN AS DIGITAL LUTHIER

This section has presented the Monome grid style interface, and the Monome derivative interfaces developed by members of the online community. Each new iteration serves to extend the functionality afforded by existing versions of the interface, thereby expanding what musicians can do with these instruments. Members of the Monome community drive this development by participating in a cycle of musical performance, group discussion, development of ideas, and creation of new instruments. This cycle allows community members to develop new iterations of the Monome built off the shared information found online.

The instruments developed by these online communities represent physical connections to virtual instruments inside of computers. It is up to each individual musician to create their own unique mapping between the physical interaction with their interface, and the way in which that action becomes a distribution of musical intention inside their computer. Through the use of physical interfaces and computer software, this process of customizing the computer instrument furthers the evolution of the performer composer by making every computer musician a digital luthier.

2.6 NEW WORK: ARDUINOME AND CHRONOME

This section presents two new interfaces developed either solely by the author, or by the author in collaboration with Jordan Hochenbaum, Brad Hill, and Ben Southall. The first interface is the Arduinome, a port of the Monome to the open-source hardware platform Arduino. The Arduinome has had a major impact on the Monome community, providing increased access to the interface, a platform on which to further modify and augment the original Monome concept, and has become a pedagogical resource for those interested in interface design. The second interface presented is the Chronome, an RGB and pressure sensitive Arduino based Monome clone. This interface was an attempt to add

completely new functionality to the original Monome concept by adding pressure sensitive buttons, and multi-color LEDs. These new features open up the potential for new mappings between the continuous data supplied by the Chronome and Monome compatible music software. Lastly, this section will present the author's work with Bricktable multi-touch interface as an additional example of the impact that online communities have on interfaces for musical expression.

2.6.1.1 ARDUINOME

Monome is a small boutique company that builds limited quantities of their interfaces. Each interface is hand made at Monome, and all parts are locally sourced.[17] This can make it challenging and expensive to purchase an interface when compared to the pricing and availability of devices made by larger companies. One solution to this challenge of obtaining a Monome is to build a clone of the interface using the online technical documents shared at their website; however, taking the provided files from information to an actual physical interface requires knowledge and skills not generally associated with musicians. So is it possible to make a clone of the Monome that is both available to everyone, and could be assembled with little to no technical knowledge, thereby expanding access to the Monome interface?

[17] About Monome - http://monome.org/

Figure 5: The first Arduinome. Built using Monome 40h buttons

This question was a central motivation for a community-based project Jordan Hochenbaum and I started with the help of Monome/Arduino community members Brad Hill and Ben Southall in the summer of 2008. This project, now the Arduinome (see Figure 5), was an effort to port the original Monome firmware from a custom circuit to the readily available and affordable Arduino microcontroller platform. In addition to being an affordable and easy to find microcontroller, the Arduino's large online community, extensive documentation, and additional I/O ports provided new potential for expansion and exploration of the Monome as an interface. This potential has resulted in members of the Monome community modifying the Arduinome with components as complex as fully featured LCD displays and multiplexed rows of continuous controllers. Monome has fully embraced this modification and exploration by including the Arduinome on its website wiki.[18] The support Monome has shown for this Arduino based clone has yielded many benefits for

[18] Monome::Arduinome - http://monome.org/docs/tech:ports:arduino

Monome, including individuals creating new hardware modifications to the original interface concept, and creating new Monome-compatible software applications. These new hardware ideas and musical applications developed by Arduinome users further extends both the Arduinome's and the Monome's functionality, and represents an example of community driven interface design.

Since the project's initial release to the Monome community, significant Arduinome activity around the clone has warranted a separate and dedicated Arduinome category in the Monome user forums. This has provided a space for Arduinome users to share their ideas with other Arduinome users, as well as the larger Monome community. This sharing of information has provided a learning resource for people interested in working with the Monome firmware, and has seen the development of a plethora of new firmware modifications and Monome derivative projects.

One remaining challenge of the project is the extensibility in the original Monome 40h serial protocol. The serial protocol describes the transmission of interface data to and from the computer. The original Monome 40h protocol supported on/off states for buttons, on/off states for the LEDs, and transmitting a handful of continuous values. This protocol made it difficult to add completely new or novel functionality such as continuous pressure from all of the 64 buttons, variable LED intensity, or LED color support. A community project called serialOSC has been developed that can potentially address these challenges by providing a prototype description of a generic Monome style interface. By creating a prototype description, an interface can transmit additional custom data messages while still being compatible with existing Monome applications. This feature was used in the development of the Chronome interface.

2.6.1.2 CHRONOME

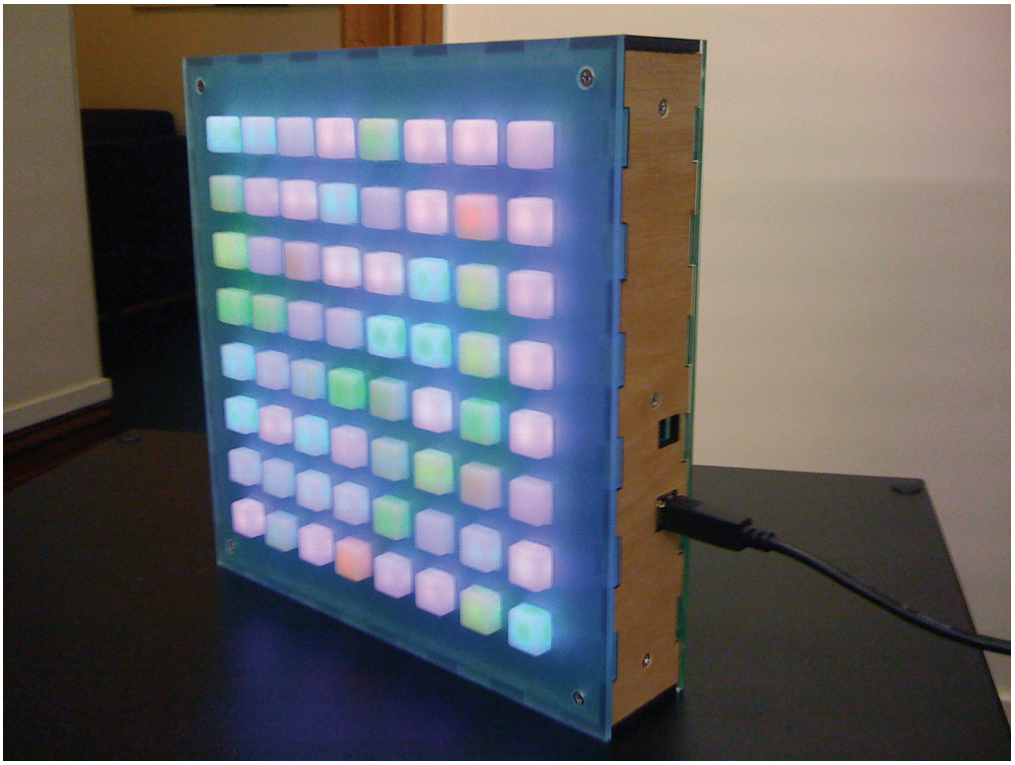


Figure 6: Chronome prototype

Designed by the author, the Chronome represents a new iteration of the Arduinome that takes inspiration from both the RGB LED support of the Octinct and the pressure sensitivity of the LUMI. A key goal of the new device was to bring both the RGB LEDs and button pressure functionality into the existing Monome application framework while at the same time continuing to use the Arduino platform as the microcontroller. The additional functionality the Chronome provides allows for a continuous spectrum of data and opens up new expressive ground for musical performance with a Monome style interface. The original Monome design was a discrete-event controller, and lent itself well to both percussive material and triggering time-sensitive events; however, the Chronome's focus on continuous data now allows for musical control to vary in gradations as opposed to the binary interactions of the original Monome. The aim of this project is similar to the Arduinome in that all information pertaining to building the interface is provided to the Monome user community in hope

that it will spur a growth in applications that take advantage of this new functionality.

The Chronome is an example of community-based design as it is inspired by two Monome derivative interfaces, the LUMI and the Octinct. Additionally, the new serialOSC software being developed by members of the Monome community provides a way to extend and modify the types of data sent to Monome compatible applications. This extension allows the Chronome project to bring color support to the existing serial protocol of the Monome. With multiple Chronomes already being built around the world, it will be exciting to see what new ideas and software will be given back to the Monome community.

2.6.1.3 THE BRICKTABLE

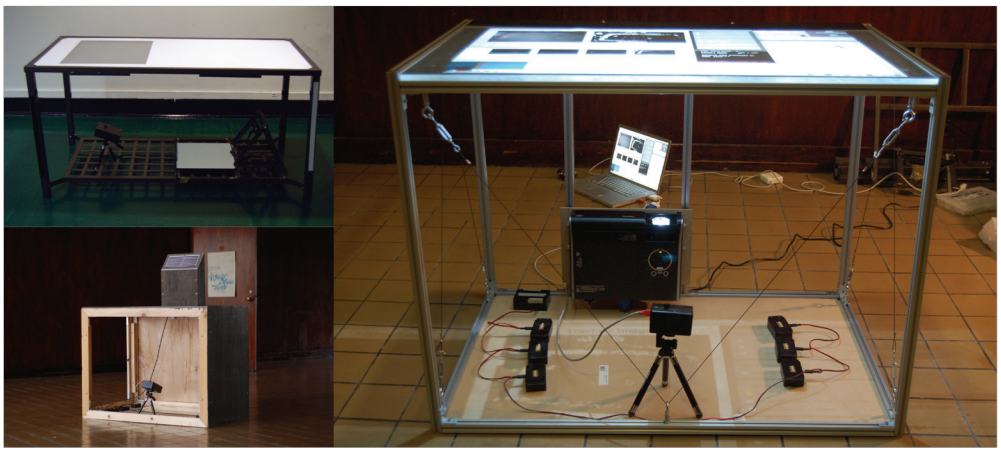


Figure 7: BrickTable version I, II, and III

The BrickTable (Hochenbaum and Vallis 2009) is a large multi-touch interface built by the author in collaboration with Jordan Hochenbaum. The interface was inspired by the Reactable (Jorda et al. 2005; Jordà et al. 2007) and originally used the open-source software ReacTIVision (Kaltenbrunner and Bencina 2007) for prototyping. Three versions of the BrickTable were built between 2008 and 2009 (see Figure 7), with all three of these versions being based off of resources and software found at the online Natural User Interface Group (NUI Group) community. Like Monome, NUI Group represents an online community of interface users and developers, and the resources provided by the community

have similarly facilitated the development and modification of an interface. The NUI Group provided technical information on building the physical interface, community developed software for vision tracking, and answers to development questions in the online forum. The BrickTable and its several iterations were made possible by using the resources provided by NUI group, and benefited greatly from iterative community based design.

2.7 MONOME, TENORI-ON COMPARISON

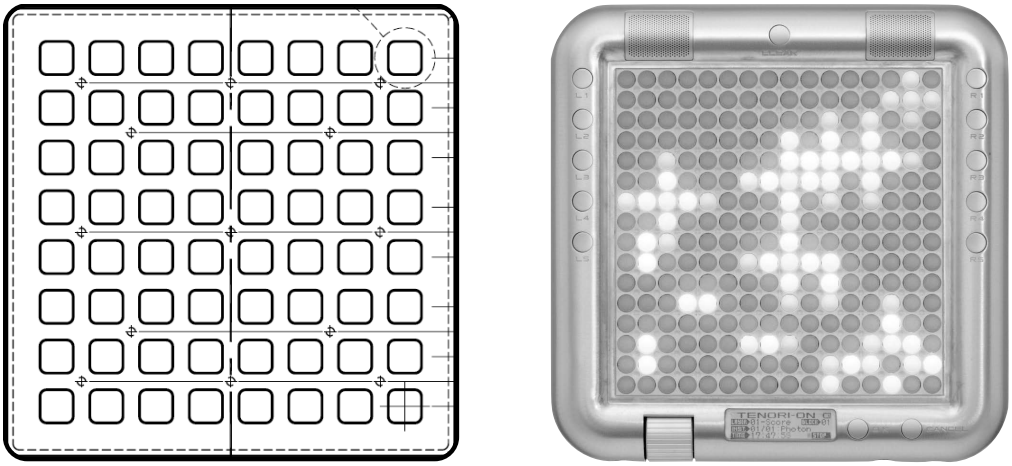


Figure 8: Monome left and Tenori-On right

Both the Monome and multi-touch interfaces represent an interesting, subtle and significant shift in how a community of users can approach interface design. The previous sections have shown how online repositories of information have enabled users with access to the Internet to learn, build, and augment musical interfaces. Contrasting the Monome with the Yamaha Tenori-On shows how an online community-driven iterative design approach, compared to a closed-box design approach, can lead to greater versatility in use.

The Tenori-On was introduced by Yamaha in 2008 and, like the Monome, contains a two-layer, NxN device consisting of a matrix of buttons situated over a matrix of LEDs. Unlike the Monome however, the Tenori-On's firmware is locked, its design specs are not made public and the device does not easily support hardware modifications. When compared with the Monome, the Tenori-On has not seen the same community of users, library of applications or variety

of uses develop. Even though these two devices share a very similar form, the history and function of the two interfaces are very divergent. The Monome has spawned a wealth of custom applications, a thriving user community and several community developed derivative interfaces, while the Tenori-On, in spite of being an interesting and exceptionally well-conceived instrument, has remained unchanged in its design and fixed in its functions.

2.8 DISCUSSION

This chapter has presented an online community-based iterative model of interface design in which expert users, making up a small percentage of the community, develop new and innovative functionalities. These extended functionalities are then made available to the larger user community without requiring the community to learn the technical details of the interface. This process allows computer musicians to perform using custom interaction between their physical interfaces and the sounds produced by the computer. While an acoustic instrument may be played in different ways, it will still be constrained to the physical interactions and the sounds resulting from those actions. In contrast, through the use of customizations and software, an interface and computer can become entirely different instruments performed in entirely different ways. The majority of users do not create these new functionalities, but instead use these developments and in return share their experiences with the rest of the community, contributing novel application and modification ideas. By allowing for an online community to develop, modify and re-envision an interface through an iterative process, a new model for interface design has been created; a model that encompasses both basic users and advanced developers alike.

Online communities have not only democratized the hardware development of musical interfaces; they have also similarly democratized the process of software development for musical interfaces. Community software developers actively listen to requests from users and regularly implement these ideas in new applications for the interfaces. This process creates a feedback loop inside the community forums; real-world use of the interfaces informs the development of

software. Conversely, software design requirements can drive the development of new functionality for an interface. In this way, software informs the design of new hardware, and hardware informs the design of new software.

An example of this is the Monome community's large repository of free and open-source custom software. Specifically, in an iterative process similar to the hardware development, an application known as MLR,[19] a program that allows for the chopping of buffered loops of audio using the Monome, has seen users take an existing open-source application and create custom versions that are then shared back with the larger community. This process helps drive new hardware ideas, including the desire to display visual information from the application on the Monome's LEDs using multiple colors. Implementing this multi-color support was a central motivating factor in developing the Chronome.

While open-source software is not a new idea, coupling it with open-source hardware creates a powerful combination that allows users to explore new ideas and helps drive development. Software such as reacTIVision and the NUI Group's Community Core Vision [20] (CCV) finger-tracking program were developed as open-source projects that required most users to build custom hardware devices in order to use them. Without access to the online community resources regarding hardware designs, the software programs would not have had the physical interfaces needed for people to implement their ideas. As an example of projects that benefit from this "completed loop" of hardware and software, the Argos project (Diakopoulos and Kapur 2010) built off the resources found at the openFrameworks community, implementing an application that simplified the designing of GUIs and extended the usability of CCV and the multi-touch hardware interfaces.

The sharing of technical resources for live computer music can even be seen in the development and evolution of the laptop orchestra. Early versions of the server that connected PLork and SLork were built using ChucK, and made

[19] App:mlr [monome] - http://docs.monome.org/doku.php?id=app:mlr.

[20] CCV - http://ccv.nuigroup.com/

available to other laptop orchestras following in their footsteps. This original ChucK server has since lead The Machine Orchestra to develop several new iterations for use in our own concerts, and has even led to derivative applications being developed in other languages. This sharing of ideas and resources is a positive influence on the development and refinement of not only the interfaces used by computer musicians, but on tools in general for live computer music.

Access to these tools allows a computer musician, using a single physical interface, to become an entire ensemble of unique instruments in a way that has not been possible using acoustic instruments. Each computer musician takes the functionality afforded by the interface, and then creates their own custom mapping to the computer instruments. This turns the computer musician into the creator of his or her own custom instruments, and represents an evolution of the performer-composer.

Chapter 3

ARMY OF ME: AUTONOMOUS AGENTS AND THE SOLO PERFORMER

> "Eventually it should be possible to develop a synthetic performer that would not require priming with a written score of what initially to listen for. Chamber music players typically perform from single part-books, building their sense of the full score strictly from the experience of rehearsal. In that this appears to be a prime route by which those players inform their overall performance, we would eventually like to understand a little of how that works."
>
> —Barry Vercoe & Miller Puckette (1985)

Computers enable the development of autonomous agents for solo live performance. Through these systems, new modes of improvisation may be explored where musicians have musical dialogues with virtual versions of themselves. These avatars are not so much separate identities as they are extensions of the performer's musical will. By listening to the performance of the human, the computer agents are able to output new musical ideas that in turn influence the musician's actions, creating a feedback system of computer-mediated improvisation. These systems create the opportunity for the performer composer to express one musical idea while having a computer generate other contextually related music. This in effect makes the computer musician simultaneously conductor and performer, and represents an evolution of the performer composer.

The following chapter begins by providing a historical overview of algorithmic composition, how these ideas are applied to automated musical systems, and finally an examination of current attempts to develop autonomous agent systems for live computer music performance. A novel search based plugin is presented for modeling two performers that are using continuous control data. Lastly, the chapter presents several different architectures for autonomous agents, and discusses the idea that agent systems can allow a solo performer to create concurrent virtual personas, and use these virtual musicians to turn single actions into multiple events.

3.1 GOALS AND MOTIVATION

The aim of this research is to examine the potential of interactive musical agents to expand the musical control of a single musician. These interactive systems listen to incoming musical data in order to generate models of the performers. These models then generate new material in the style of one performer given a context provided by another performer and the output of the interactive musical agent (see Figure 9). Arnie Eigenfeldt states that "Such performance systems can be considered as complex instruments, in that multiple gestures are generated that proceed and interact in complex ways, yet under the direction of a single performer/operator" (Eigenfeldt 2006). In a similar manner to how piano pedals expanded expressive potential by allowing pianists to sustain notes, the development of interactive musical agents expands the level of control in live computer music by increasing the number of simultaneous actions a performer can control or influence.

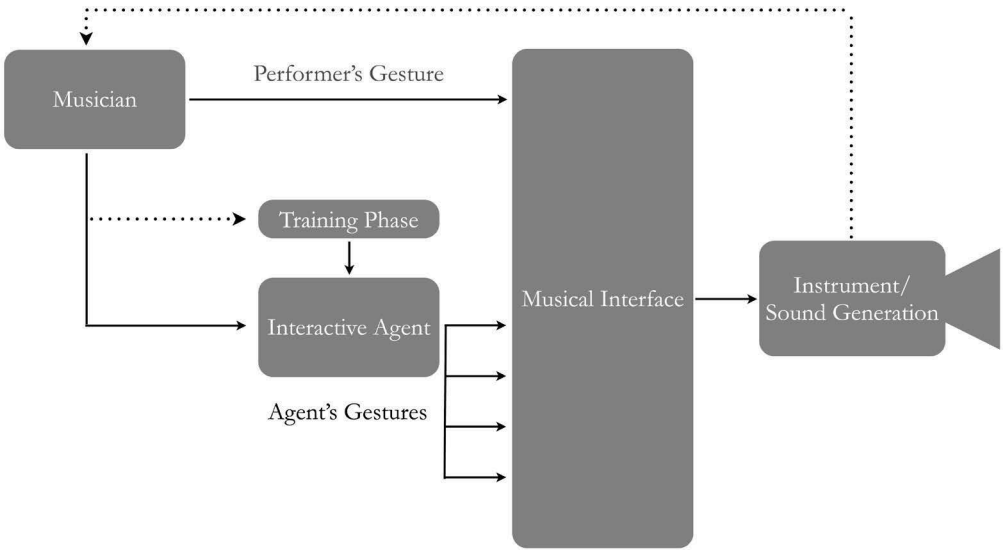


Figure 9: Interactive musical agent for modeling continuous control data from musical interfaces

The design of these systems draw heavily from the fields of statistics, AI, and Machine Learning, in which models are built to describe complex mappings between inputs and outputs. These systems are built using a variety of approaches, including statistical modeling, sub-symbolic networks, genetic algorithms, or search based systems. Through the application of these various approaches, models can be learned by presenting the computer with examples of input and output pairs, and in this way build a system that addresses Vercoe and Puckette's desire for an interactive instrument that would learn from rehearsals. The development of interactive agents represents an evolution of the performer composer as they afford musicians the ability to perform and improve with multiple versions of themselves. This creates the opportunity for the instrument to become a new musical tool in which the ideas of the performer composer are used as contextual input to generate parallel streams of musical material from a single idea, and in the process inform the musician's future decisions.

3.2 BACKGROUND

The interactive musical systems discussed in this chapter can be seen as evolving out of a long history of algorithmic composition. Gerhard Nierhaus defines algorithmic composition as "a formalizable and abstracting procedure which – applied to the generation of musical structure – determines the field of

application of algorithmic composition" (Nierhaus 2009, pg. 11). Three composers are frequently presented as early examples of algorithmic composition. Guido of Arezzo's treatise "micrologus" was written around AD 1026 (Nierhaus 2009, pg. 30), and lays out a set of rules for the writing of early polyphonic music called organum. Johannes Ockeghem's 15th century work *Missa Prolationum* consists of prolation canons in which two contrapuntal melodies are split between four voices each at a different speed, with the interval of imitation becoming larger with each piece in the work (Groot 1997). And finally, Mozart's *Musikalisches Würfelspiel* is a famous example of the use of dice to choose from a predetermined set of compositional ideas (Nierhaus 2009). Mozart's use of dice can be seen as a precursor to aleatoric compositional approaches applied in the 20th century. The attempt to formalize an approach to composition has carried on to the present, with composers exploring increasingly complex algorithms. With the advent of the computer age a new world of possibilities has now opened up, enabling composers to explore algorithms that would have previously been prohibitively complex.

3.2.1 TWENTIETH CENTURY COMPOSERS

The twentieth century was witness to an expanded exploration of algorithmic compositions by composers such as Arnold Schoenberg, John Cage, Steve Reich, Iannis Xenakis, and many others. These composers experimented with formalizations outside of tonal music, the application of chance procedures to composition and of procedures in general, as well as the potential application of advanced mathematics to music. The work of these composers represents a transition period from hand written to computer-mediated algorithmic composition, ultimately leading to the current development of agent-based systems.

The early part of the twentieth century was witness to the rise of atonal music by such composers as Alban Berg and Anton Webern. This music aimed to avoid a single tonal center in a piece of music. As early as 1921, Arnold Schoenberg began exploring a compositional system for formalizing the avoidance of a tonal

center by evenly distributing all twelve notes of the chromatic scale within a composition. The rules governing this even distribution of tones represent a formalized approach to composition. The twelve-tone system evolved into the broader genre of Serialism, which included the development of derivative systems such as integral serialism, total serialism, general serialism, and multiple serialism.

In his 1971 book *Formalized Music* (Xenakis 1971), Iannis Xenakis contested that the complexity inherent in serialist compositions is not discernable by the listener, and instead is perceived as a random collection of notes. Xenakis aimed to remedy this with his "stochastic music", which attempted to create systems that would avoid a tonal center while still providing a cohesive form ascertainable by the listener. These systems were derived from various fields of mathematics such as probability theory, Markovian processes, set theory, and others. Xenakis' stochastic music can be seen as an early attempt to model a system with a finite domain and a deterministic nature, preserving larger scale structures while probabilistically deriving the details.

Xenakis' criticism of indeterminacy was not limited to serialism, and extended to the work of John Cage, with Xenakis saying that "complete freedom, as is the case of Cage, says in effect 'do what you like, at any moment, no matter how'" (Bois 1967). While John Cage's use of the I Ching to compose his music through chance procedures does represent a more random approach to algorithmic composition, it still requires that certain basic assumptions—or boot strapping—be made about the musical domain prior to composing. Decisions such as what musical elements the I Ching will decide, and how these elements interact with each other, place a set of constraints on the musical domain that inevitably end up defining the final composition.

Xenakis may be right in questioning if the listener would be able to discern the complex relationships within either Serialism or chance based music, but unforeseen events within the compositional complexity may give rise to an emergent music. This idea of an emergent music became very important to

network computer music ensembles of the 1980's (see Chapter 4), and provides a fertile ground for the exploration of algorithmic composition as it accounts for the idea of unplanned structure emerging from a chaotic system.

Another solution to Xenakis' criticism of indeterminacy is seen in the process music of composers such as Steve Reich and Terry Riley. Steve Reich's tape phasing and Terry Riley's *In C* represent the algorithm as a procedural approach to composition; procedural in the sense that the music takes the compositional algorithm and makes it the central aesthetic element during both the creation and the performance of the pieces. As an example, Reich's phase pieces allow for the music to emerge through the process of phasing two identical melodies, allowing for a multitude of different musical ideas to come out of a single melody. A more contemporary example of form emerging from process would be William Basinski's Disintegration Loops. In these pieces a loop of music is played back on old decaying tape. Each time the loop plays, a little more of the tape recording disintegrates, creating a clear aural link between the process and the music. These procedural approaches to composition constitute algorithmic composition in that they are formalized, and are unique in that the composition and performance are linked together through the procedure.

3.2.2 COMPUTER AIDED ALGORITHIMIC COMPOSITION

As computers continued to increase in processing power, composers began to explore algorithmic approaches to composition that would have previously been too laborious and time consuming to undertake by hand. As an example, the computer enabled Xenakis to create programs that would automate the calculation of probabilities, allowing him to continue exploring ever more complex musical ideas.

In Charles Ames' review of automated composition (1987), he describes the 1956 efforts of Klien and Bolitho's *Push Button Bertha* as one of the first computer assisted automated compositions. He goes on to discuss other seminal pieces such as Hiller and Issacson's 1959 *Illiac Suite* in which Markovian processes were used to compose the music, James Tenney's 1963 work *Stochastic*

String Quartet which used Max Matthew's musical programming language MUSIC 4, as well as Xenakis' use of computers for his 1962 work *Morsima-Amorsima*. All of these pieces leveraged the ability of a computer to perfectly execute instructions, opening a new world for algorithmic composers.

As computer automated composition developed, computing languages dedicated to algorithmic composition began to develop. Paul Lansky created the MIX language, and Larry Polansky, Phil Burk, and David Rosenboom developed the language Hierarchical Music Specification Language (HMSL) (Polansky 1994). These languages, and many others like them were designed to aid in algorithmically composing music using computers, and created building blocks from which ever more complex algorithmic compositional systems could be built.

While the frameworks above allowed for the creation of complex algorithmic compositions, they were not built to "learn" or imitate a style or genre of music. David Cope created a system called Experiments in Musical Intelligence (EMI) that could be trained with a corpus of example music, and then generate new compositions in that musical style or genre (Cope 2005). Cope proposes that a musician's style is related to the re-occurrence of themes and ideas, and that probabilistic modeling of a musician's corpus of work will reveal these themes. EMI is important to the development of an interactive agent system as it shows that a musician's style or musical personality can be modeled by examining their performances.

This research has also found its way into commercial products such as PG Music's Band-in-a Box,[21] and Steinberg's Groove Agent.[22] These advances can be seen as an evolution of composition from hand-written ideas to computer-mediated algorithms; whoever, it was not until computers became fast enough to support interactive systems that an evolution of the performer composer began.

[21] PG Music - http://pgmusic.com

[22] Groove Agent - http://steinberg.net/en/products/vst/groove_agent/groove_agent.html

3.2.3 INTERACTIVE MUSICAL AGENTS

The 1970's saw the beginnings of efforts to use computers to not only automate composition, but also to interact with, and influence algorithms in real time. Systems such as GROOVE enabled composer Emmanuel Ghent to write his 1974 piece *Lustrum* by influencing a probabilistic compositional program while it was running (Ames 1987).

As computers became faster, and real time interaction becoming a reality, composers began to look beyond simply automating algorithmic composition. Research looked into more complex methods of interaction between human performers and computers. Programs were developed that allowed for automated accompaniment systems capable of listening to human performers, and determining their position within a musical score (Dannenberg 1984), as well as systems complex enough to model, and act as a synthetic performer within an ensemble (Vercoe 1984). These systems represent a major step forward in the evolution of the performer composer as they allowed for the performance of the human to contextualize the musical output of the computer. Although these systems were able to process live input, they weren't capable of increasing the quality of their performance through rehearsals. With this in mind, researchers began to examine ways in which systems could learn, and then later reference this past performance data (Vercoe and Puckette 1985).

These early systems used optimized search algorithms to compared performer's inputs against a fixed score that provided a context for the automatic accompaniment. While this was an effective approach, it does not work for more improvisatory situations. Improvisation focused performance has a less defined form, making it challenging to provide a context from which the computer can generate music. Research into utilizing machine learning techniques allowed systems to no longer simply accompany a score, but instead be able to "learn" and recognize different styles of music (Dannenberg, Thom, and Watson 1997). In order to recognize the more diffuse connections defining musical styles,

systems increasingly began to use sub-symbolic systems that learned from inference, rather then symbolic systems that learned from a set of rules.

Armed with this new ability to learn and emulate performance styles, "intelligent" auto-accompaniment systems were developed that allowed for improvisational interaction. Examples of these systems are George Lewis' Voyager, a nonhierarchical improvisation instrument that listens to performance and improvise with the musician (Lewis 2000); Francois Pachet's use of Markov Models to design The Continuator, an interactive musical system that generates music in the style of the performer using call and response (Pachet 2002); and an interactive improvisation system for generating rhythms in the style of North Indian Tal (Wright and Wessel 1998). These interactive systems no longer needed to follow a fixed score, and the ideas of Interactive Music Systems, Agents, or Machine Musicianship was defined (Rowe 2001).

3.2.4 CONTEMPORARY SYSTEMS

There has been considerable growth in the development of interactive musical agents since the publication of Rowe's Machine Musicianship. Major conferences such as the International Symposium on Music Information Retrieval (ISMIR) (Downie, Byrd, and Crawford 2009), and the international conference on New Interfaces for Musical Expression (NIME) (Cook 2001), have provided a forum for the development and application of interactive musical agents. New interactive systems have been developed which listen to input from musicians, learn from the performance, and then generate improvised, contextually relevant material in response to the incoming data. Some examples of these systems include: A jam session system capable of simultaneously listening two three guitarists and generating a "personality model" of each player (Hamanaka et al. 2003), with these models then being used to substitute the musicians with virtual performers; Haile, an anthropomorphic drum robot that is able to listen to a musician and synchronize its drumming with the performance of the human (Weinberg, Driscoll, and Parry 2005); Kinetik Engine, a drumming ensemble system consisting of four agents that generate rhythms by listening to each other

and the input of a human "conductor" (Eigenfeldt 2006); and the work of Nick Collins towards the creation of autonomous agents for live computer music (Nick Collins 2006), and his development of an improvisational system that provides contrary musical improvisation in opposition to a musician's performance (Nick Collins 2010).

3.3 DEFINE THE CHALLENGE

Existing research into creating interactive musical agents has shown promise in developing systems that can learn from a musician, and then interact with them in real-time during a performance. However, it would be difficult to imagine a "one-size-fits-all" solution to the creation of these interactive agents. The reason for this is linked to the fact that musical styles are defined using a wide variety of parameters, e.g., pitch, rhythm, timbre, tempo, duration, polyphony, harmony, dynamics, cultural influences, etc. Additionally, the parameters that define one genre may be meaningless in another.

Even exceptional human musicians are unlikely to be virtuosic in every musical style that exists. The subtleties of each type of musical style are not necessarily linked to each other in any meaningful way, and are often tied to cultural influences. One could imagine that someone's mastery of baroque figured bass would not necessarily make them a virtuosic classical north Indian musician. Similarly, it seems unrealistic to create an interactive musical agent that is capable of accompanying in all styles. Nick Collins explains how the rules described in the Generative Tonal Theory of Music (GTTM) (Lerdahl and Jackendoff 1996) do not apply well to the music of Bolivian campesinos (Stobart and Cross 2000). He goes on to say that, "the great variety of metrical structures in the world's music (Temperley 2004; Clayton 2001; College 2004), the lack of any 'universal' musician conversant with all musics, let alone the notorious difficulties in ascribing musical meaning, all suggest that cultural factors are essential in obtaining musical competency" (Nick Collins 2006). This would imply that not only is it unlikely that an interactive musical agent would be effective at all

musical styles, but that such a homogenous approach to modeling music is not desirable.

The research presented in this chapter is interested in developing a custom interactive system that will be focused on the live performance of beat-based electronic music such as techno or electronica. This interactive system will leverage specific features found in these genres, such as traditionally adhering to a single tempo, and frequent use of a 4/4 beat structure. Additionally, the control parameters currently used by the author are based around custom mappings between the Chronome described in Chapter 3, and Ableton Live.[23] These mappings tend not to control note or pitch based instruments, but instead focus on controlling timbre using effects and re-sequencing sampled material to create new arrangements. This makes leveraging rules regarding tonality difficult to use. For example, data reduction techniques such as reducing pitch to interval relationships do not apply to a knob that controls the cutoff of a filter, or a button that bypasses an effect.

3.4 Implementation: Live Performance Systems

During the course of this research several approaches to designing interactive musical agents were explored in an effort to gain a better understanding of the possible solutions, and challenges (see Appendix D, Appendix E, Appendix F and Appendix G). This research led to several requirements for the interactive system:

1. The system is a plugin, allowing for integration with the Chronome/Ableton performance system.

2. The system is able to continue to perform with or without input from the musician.

[23] Ableton - http://ableton.com

3. The system is able to learn from rehearsal instead of being presented with a score.

4. The system is based off of a concatantive synthesis approach, where incoming performance data is used to search a database for an appropriate response.

Taking these requirements into account, the following system is built using a search approach to interactive musical systems, based off a similarity algorithm called S2MP (Martin et al. 2011).

3.4.1 SEARCH BASED SYSTEMS

Search based systems are algorithms that take an input as a key, and find a matching value in a stored database of examples. These examples can be single states such as notes, or longer sequences of states that each store many different pieces of information. An example of this type of sequence might be a melody line, where each state in the sequence stores the pitch, velocity, and duration of the note.

Search algorithms have been used in the development of auto accompaniment systems (Dannenberg 1984), improvisation systems (Rowe 2001), and more recent interactive musical agents that attempt to model the relationship between two performers (Martin et al. 2011). While the basic concept behind search based systems is a simple key-value database, musical implementations of the approach require special considerations. The system must decide how to handle matching the real-time input to the database states, what state information is to be returned once a match is found, and how to do all this fast enough for live musical performance.

This section will (1) discuss the considerations when adapting the auto accompaniment systems for improvisational interactive agents; (2) present developments and modifications of the S2MP algorithm originally described in

(Martin et al. 2011); (3) and discuss the challenges with designing interactive musical agents using search-based algorithms.

3.4.2 CONSIDERATIONS FOR USE WITH IMPROVISATION

While the approach described in Appendix E allows for a faster search by only comparing neighboring sequence events, improvisational systems do not closely follow a pre-determined score, and therefore require additional constraints (Rowe 2001). However, while the progression of sequences may not be tied to a score, they may represent phrasing, and allow for segmentation to provide a similar form of constraints. Certain styles of beat based live computer music such as techno are closely associated with grid based rhythmic structures such as strong beats and bars, and it is possible to segment or group sequences along these rhythmic divisions.

With this in mind three assumptions can be made. First it can be assumed that the neighbor of the previous best matching sequence within a database is likely to provide a high scoring match, although not necessarily the highest score. Secondly, it is also reasonable to assume that a high matching sequence is more likely to be found at similar beat locations within in a bar versus other positions within the bar. Finally, if the database in use is a collection of sequences that represent performances (see Figure 12), then it can be assumed that related improvisational ideas will be explored closer to each other rather then randomly spread out. While these assumptions cannot be guaranteed, they will allow for constraints to be applied to the search in a similar manner to Dannenberg's system (Dannenberg 1984).

Additionally, searching algorithms require training a database of good examples to find and match during performance. Describing the examples as "good" is important because a search algorithm looks for the closest matching input, and will return a poorly played matching performance just as easily as a well played matching performance. While lots of training may help to mitigate this issue of "bad example" data for probability or regression based systems, search based systems have no statistical inference for preferred input states versus aberrant

state examples. One solution to these challenges could be a pruning or approval of the data before submitting it to the search database.

3.4.3 S2MP: A SIMILARITY MATCHING ALGORITHM

To explore the potential of search-based approaches for improvisational systems, the following research implemented a similarity-matching algorithm called S2MP (Martin et al. 2011; Saneifar et al. 2008). Martin's research shows the potential of this algorithm in developing interactive musical agents that perform alongside a human musician in the style of another performer. His original system is based around binary state buttons that switch on or off ten algorithmic musical instruments. The system described here aims to expand this algorithm for use with continuous control data from musical interfaces, and to implement it as a plugin for integration in modern Digital Audio Workstations (DAW) such as Ableton Live.

A brief description of the algorithm follows, but the reader should refer to (Saneifar et al. 2008) for a more details. The original algorithm uses an input sequence as a key to search for a match within a much larger stored sequence. This is achieved by comparing the input sequence against all sub-sequences of the same size from a database of past performances. The algorithm considers a sequence to be made up of a collection of item sets. Item sets can be thought of as a chord, while a sequence would be the order of chords as they are played in a performance. This algorithm is well suited to finding generalized sequence matches by calculating both a *mapping score* for the union of the items within two sequences, and an *order score* for how well the matching item sets maintain a similar order. These two scores are then combined to provide a general similarity score for the two sequences. For the system described here, the item sets are samples of continuous controls used by two computer musicians, stored as a single sorted set, while the sequences would be these samples stored to disk as a performance.

3.4.4 TRAINING THE SYSTEM, AND LINKING CONTROLLER DATA

Creating these item sets requires converting the controller ID and value for the two performers into a single 1-dimensional array, and is achieved for the first performer by multiplying the controller ID by 128 and adding the product to the controller value. The second performer is concatenated to first performer by adding 128 to the controller ID, then multiplying by 128, and finally adding the controller value (see Figure 10 and Figure 12). This essentially flattens the matrix of controller IDs and controller values, and converts it to a 32,896-position array, allowing for synchronous samples of the state of both performers' controllers. However, the S2MP algorithm allows for some optimizations in storing the array. The order of events within an item set does not matter; therefore there is no need to store duplicate events. Additionally, the interactive agent should only update a controller when the value has changed; therefore the item sets only need to store events that have changed since the last item set was created. Lastly, the mapping score can be efficiently calculated using the intersection of two sets. For these reasons, the flattened array is best stored as a sorted set.

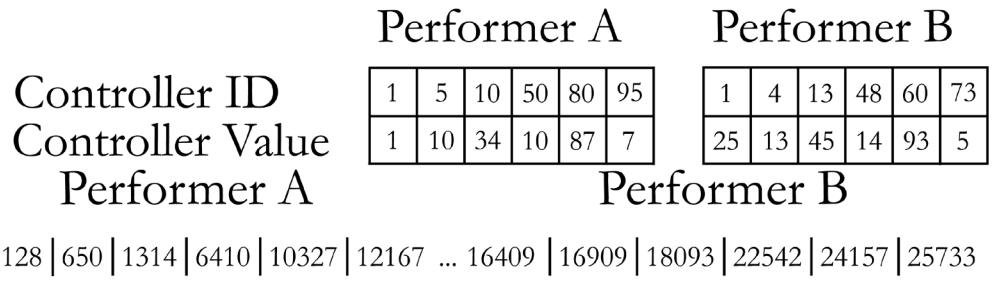


Figure 10: Controller data from both performers stored as 2-D and flattened 1-D collections

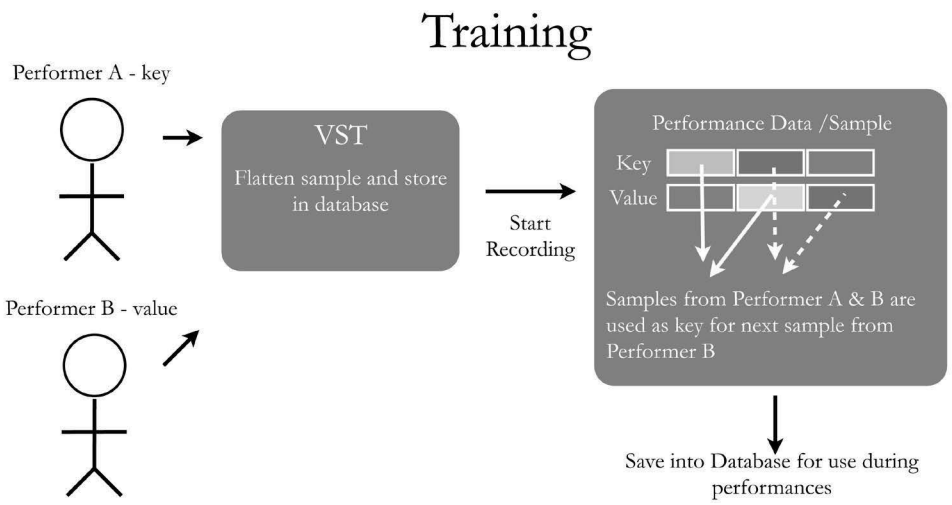


Figure 11 Search algorithm training algorithm. The two performers are sampled by the plugin, and performer B's current item set is linked with Performer A's previous item set.

During training, the incoming data from both performers A and B are sampled at a constant rate. Each sample is used to create a sorted set that contains the control data from both performers, and a MIDI buffer containing the time stamped control data from performer B. The MIDI buffer is then paired with the previous sorted set in a key/value struct, and this struct is then appended to a database, extending the recorded performance. This process creates a long sequence of item sets and linked MIDI buffers, representing synchronous samples of both performers' controllers for a particular performance. Additional rehearsals or performances can be stored as separate sequences, allowing for different "takes" on the piece (see Figure 12).

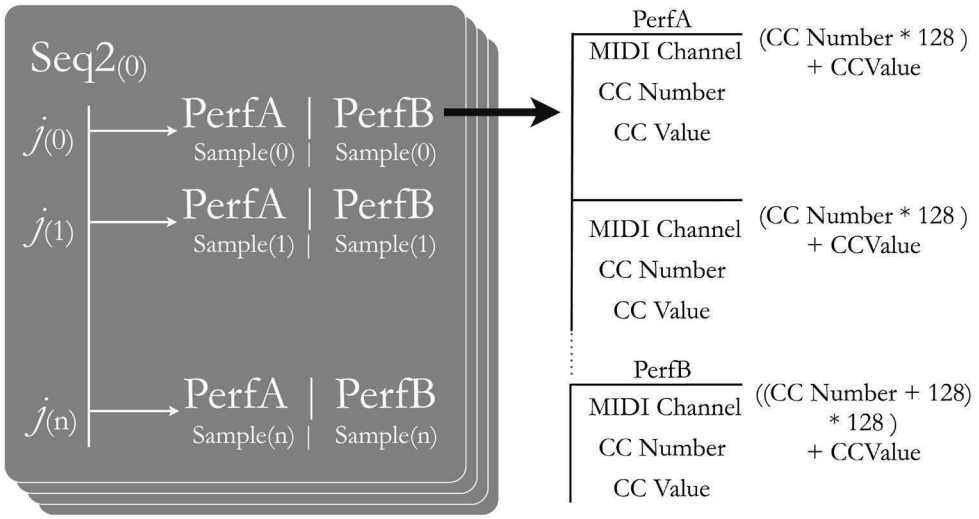


Figure 12: Stored performances for S2MP algorithm. Sequential item sets for a performance are stored as $Seq_{2(n)}$. Each j^{th} item set represents both Performer A and B, and has a MIDI buffer associated with it.

3.4.5 IMPLEMENTATION OF THE SYSTEM FOR USE IN PERFORMANCE

During performance, a sequence is built using both Performer A's current input and the virtual Performer B's current output from the system. The input sequence, which will henceforth be referred to as Seq_1, is a fixed size buffer. Once Seq_1 is initially filled with item sets, it is then compared against the sequences from the database, which will henceforth be referred to as $Seq_{2(n)}$ where *n* refers to a specific stored performance. As described in section 3.4.2, an exhaustive search of the database is not only slow, but also not necessary if the comparison between Seq_1 and the database leverages the current position within the bar. Seq_1 can be compared against all sub-sequences in $Seq_{2(n)}$ that end at the current position since the start of the last bar (see Figure 13). If the samples are taken at 96^{th} notes, then an exhaustive search of a bar will require a total of 384 comparisons, while this location-based approach will only require one. Additionally, this can be taken further if it is assumed that the last matching location in the database will likely provide a good match for the next search. The current location can be thought of as highly likely to yield similar musical ideas,

with good matches dropping off as the search distance increases. This can be used to increase the distance between searches, i.e., the search would look at the neighboring bars, then two bars form that, then four again, and so forth. While this distribution of musical matches is not strictly true of all music,[24] it represents a good compromise between speed and search.

Finally, Martin's paper discusses the tendency of the search results to jump around the database, and consequently create undesirable discontinuities in the output. Testing of the algorithm confirms this, and my proposed solution is a bias towards the compares closer to the previous compare location using weights on the similarity scores. This should help mitigate the issue of jumping around without adversely affecting performance, as it's likely that a gestural phrase or segment will be continuous and not jump between different bars. These weights re-enforce the idea of imposing a Gaussian distribution to searching the recorded sequences.

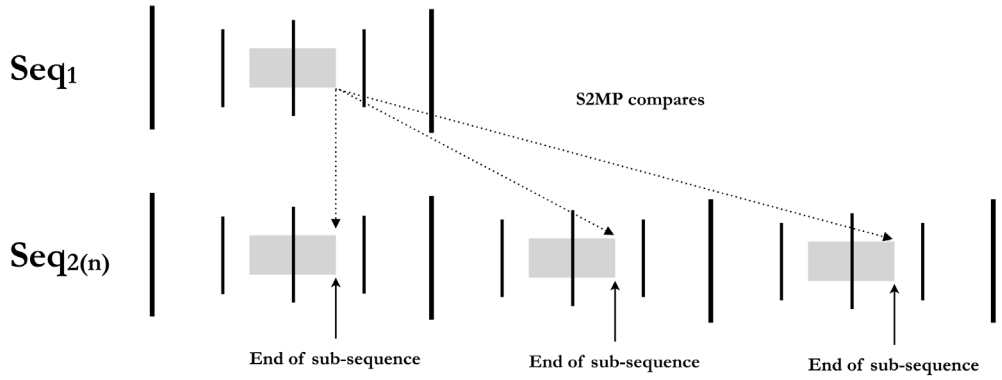


Figure 13: Constraining the S2MP search to current position within a bar. This represents a good compromise between efficiency and search as it cuts the number of compares and is likely to return good matches.

The actual comparison between Seq_1 and sub-sequences of $Seq_{2(n)}$ first requires taking the intersection of the item sets between the two sequences. This creates a matrix of similarity mappings of items between item sets by assigning a score for the i^{th} item set in Seq_1 for every j^{th} item set in $Seq_{2(n)}$ (see Figure 14). Once the

[24] One could imagine a pop song that has several choruses with similar musical material for sub-sequence matches, but are separated into chunks within the recorded sequence.

matrix is complete, the highest mapping score for each i^{th} item set in Seq_1 is stored, and any conflicts where two item sets from Seq_1 are mapped to the same j^{th} item set in $Seq_{2(n)}$ are resolved to unique mappings if possible. This process places a preference for mappings that maintain the temporal order of the stored item sets within Seq_1. For example, in Figure 14 item sets 1 and 2 of Seq_1 would both be mapped to item set 1 of $Seq_{2(n)}$. In this case it would be preferable to remap item set 2 of Seq_1 to item set 2 of $Seq_{2(n)}$. Once all mapping conflicts have been resolved, a final single mapping score is calculated by taking the average of the individual mapping scores for all item sets in Seq_1.

Seq_1	Sub-sequence from $Seq_{2(n)}$ 1	2	3	4	5	6
1	.9	.3	.01	.4	.2	.8
2	.9	.8	.01	.01	.05	.4
3	.5	.01	.3	.5	.9	.2
4	.01	.2	.5	.9	.7	.2
5	.01	.01	.01	.3	.2	.4
6	.2	.2	.5	.3	.01	.2

Figure 14: Mapping score matrix. The intersection is taken between every i^{th} item set in Seq_1 and the j_{th} item set in the current sub-sequence from $Seq_{2(n)}$.

The algorithm then compares how well this mapping maintains a similar temporal order between the input sequence and the sub-sequence, similarly assigning an order score. This is achieved by comparing the order of the mapped item sets from $Seq_{2(n)}$. A perfect temporal match would see the highest mapping scores create a diagonal from the top left, down through the mapping matrix. Conversely, any mapping where the position of the j^{th} item set from $Seq_{2(n)}$ is less then the previously mapped item set from $Seq_{2(n)}$ equals a break in the order of the sequence. The most extreme example of this breaking of the temporal order

would see the highest mapping scores create the reverse diagonal through the matrix starting at the top right and moving to the bottom left. This is similar to Rowe's explanation of sequence order tests described in Appendix E and Figure 48.

Lastly, a weighted sum of the mapping score and the order score is then returned. The weight allows for the algorithm to bias the amount that the matching score, or the ordering score contributes to the final similarity score. This process is repeated for each sub-sequence returned by the database. The MIDI buffer from the sub-sequence within $Seq_{2(n)}$ with the highest similarity score is then used as the output for the virtual performer B. In effect, this system takes a concatantive synthesis approach to creating the output of Performer B.

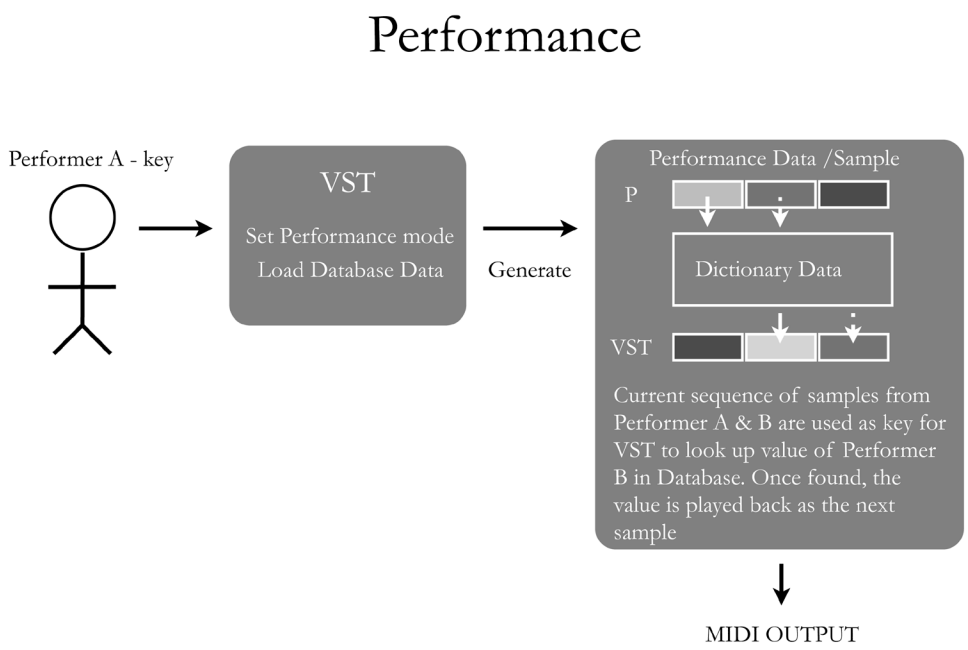


Figure 15: Performance diagram of search algorithm. Sampling Performer A along with the current agent output generates the input sequence. This is then compared against sub-sequences within the database, providing similarity scores. The MIDI buffer is taken from the sub-sequence with the highest similarity score, and then appended to the plugin's MIDI output.

3.4.6 PLUGIN DESIGN

The system is implemented as a plugin using the JUCE C++ audio library,[25] and is run inside Ableton Live. Developing a plugin allows the system to synchronize the sampling of MIDI data with Ableton's global sample clock, and allows for integration with a popular live computer music platform.

During training, the plugin samples the MIDI data of two musicians with Performer A being sent on MIDI Channel 1, and Performer B being sent on MIDI Channel 2 (see Figure 16). This allows the plugin to simultaneously sample both performers, for a total of 256 different control sources.

[25] JUCE - http://rawmaterialsoftware.com/juce.php

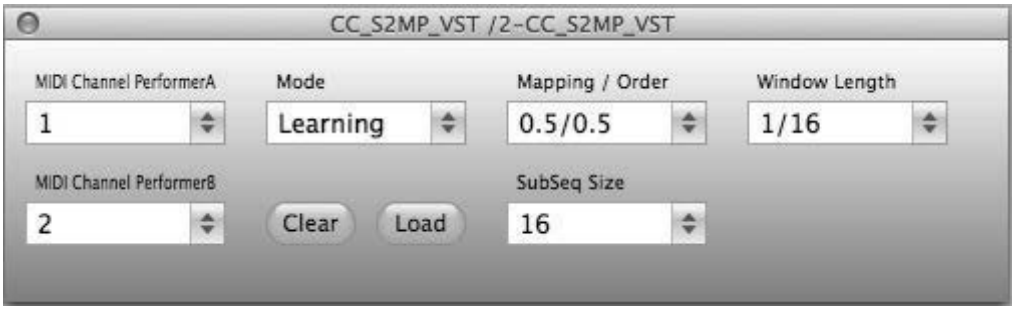


Figure 16: Continuous Control S2MP plugin

The plugin is passed a MIDI buffer for each process block, and parses the buffer into before or after the start of the next sample window (see Figure 17). If the MIDI buffer is within the current sample window then the MIDI event is added to a sorted set based on the associated MIDI Channel (see 3.4.4), and the MIDI from performer B is stored in a separate sample window length MIDI buffer. If a MIDI buffer straddles a sample window, then the MIDI events that occur after the sample window are parsed into an overflow sorted set which will be used as the first events of the next sample window. Sample windows can be set at a $1/384^{th}$ note or higher using the *Window Length* parameter, with a window of a $1/16^{th}$ note performing well in tests (see the following section). At the end of a Sample window, the sorted set and the MIDI buffer are added to a database representing key/value pairs (see section 3.4.5).

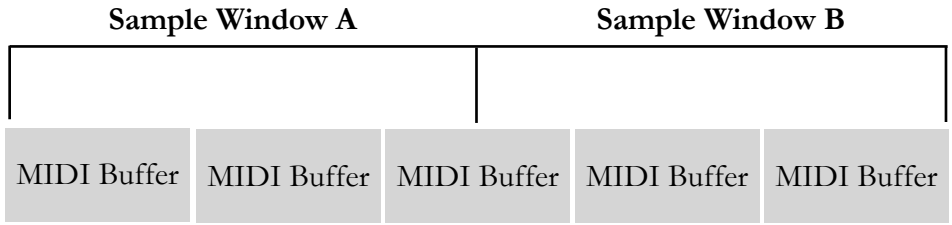


Figure 17: Parsing MIDI buffers into sample windows

Once the plugin has recorded the MIDI data from the two musicians, the system can then be set into *Performance* mode. This mode samples incoming data from performer A in the same way as before, but also adds the current output of the system to the sorted set as well. This set is then concatenated to a sequence of sorted sets, and used as the input sequence to the S2MP algorithm. The length of

this input sequence is determined by the S2MP algorithm's *sub-sequence length* parameter, which is set via the plugin. Lastly, the *Mapping / Order* parameter controls the weighting of the similarity score for a given sequence, which the user can also set from the plugin interface. Weighting more strongly towards "Mapping" places a greater emphasis on the similarity of events within the item sets of two sequences regardless of their order, while weighting more strongly towards "Order" places greater emphasis on the similarity of order between the mappings of the two sequences (see section 3.4.5). The settings in Figure 16 would create even weightings of 0.5 and 0.5 for the Mapping / Order parameter, with a new item set being created every $1/16^{th}$ note, ultimately creating a sequence of sixteen sorted sets that represent one bar of performance data.

3.4.7 ANALYSIS

The S2MP algorithm was tested using Ableton Live, and eight bars of CC data representing two performers (Performer A - Human, and Performer B – Agent) setup in pairs. Each bar of MIDI data contained two different CC curves, with the Human channel using CC numbers 14 and 15, and the Agent channel using CC numbers 16 and 17 (see Figure 18). Each bar of CC data was created to be unique and "simple" in order to easily differentiate bars of MIDI in the output sequences. The training consisted of routing this CC data into two different MIDI channels, with the Human channel being sent to S2MP channel 1 and the Agent channel being sent to S2MP channel 2. The initial training provided a single transition between each bar of MIDI, i.e., A leads to B, which leads to C, which leads to D, which finally leads back to A. Later tests were trained using the sequence ABCDACBDBADCA that provided three transitions for each bar of MIDI.

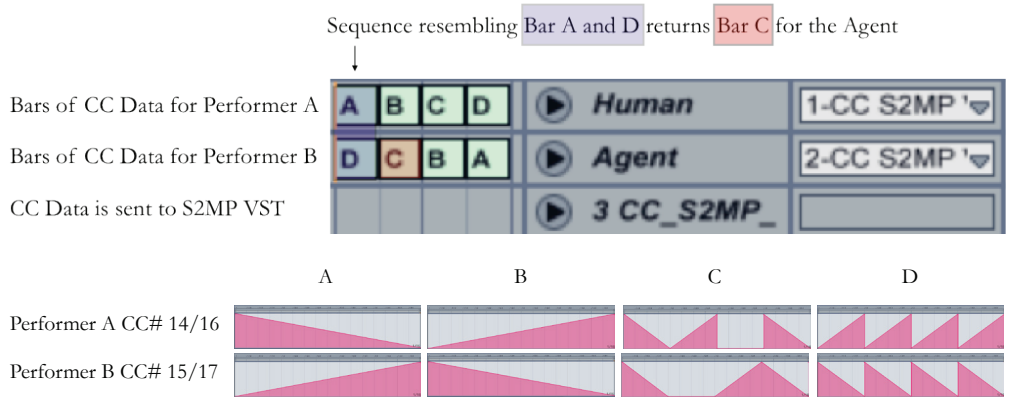


Figure 18: Routing setup for S2MP training in Ableton Live, and MIDI CC training data

The following tests consist of training the plugin using musical sequences of increasing length, and then measuring the ability of the S2MP algorithm to match a target sequence. The *Target* sequence is what the algorithm believed Performer B would play in response to an input sequence from Performer A, based on the initial training data. This is then compared against the actual *Ouput* sequence generated but the plugin (see Figure 19).

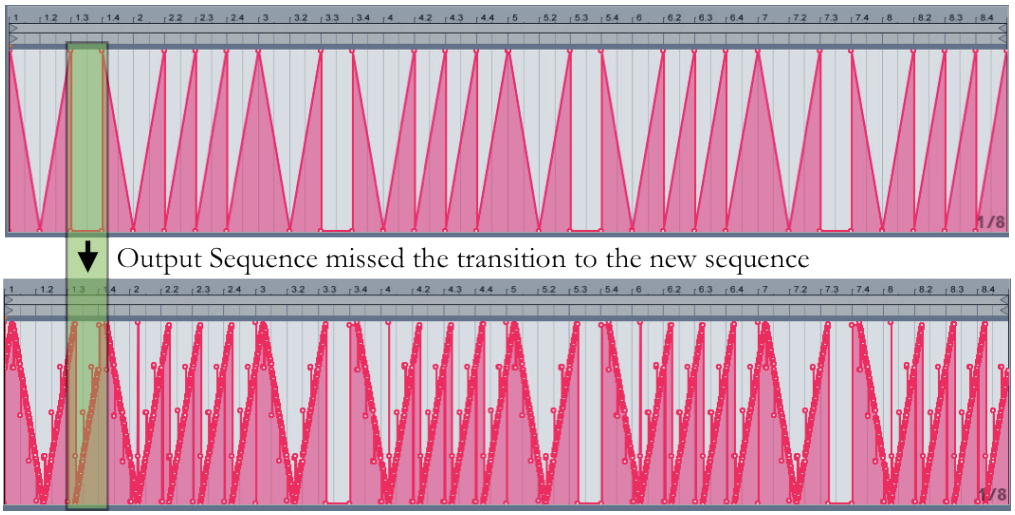


Figure 19: Target CC sequence (top) vs. Output sequence (bottom). The output sequence didn't match the target sequence in the first bar, but otherwise was a perfect match.

3.4.7.1 INITIAL TESTS

The first four tests show whether or not the S2MP plugin is capable of finding "correct" matches within the training database. Each test appends one additional bar of CC data to the training sequence, with the first test using sequence AA

and the last test using sequence ABCDA. These training sequences provide the plugin with one transition for each bar of MIDI. A transition in this case represents the end of a bar of MIDI, and may require the algorithm to jump to another section of the database to continue creating an output sequence.

Once training is complete, the S2MP algorithm generates an output sequence of eight bars for Performer B, using Performer B's previously trained data. These test then compare the output sequence against a target sequence, e.g., the first target sequence consists of eight bars of A, the second target is then alternating bars of AB, the third consists of ABC, and finally the fourth is eight bars of ABCD. For the tests, equal weighting is given to the Mapping / Order parameter, the window length is set to one item set every 1/16th note, and the input sequence length is set to sixteen item sets. These settings mean that the plugin creates an item set every 1/16th using the performance data from the last window. These stored item sets are then used during performance by the search algorithm in blocks of sixteen, creating one bar sequences for the search.

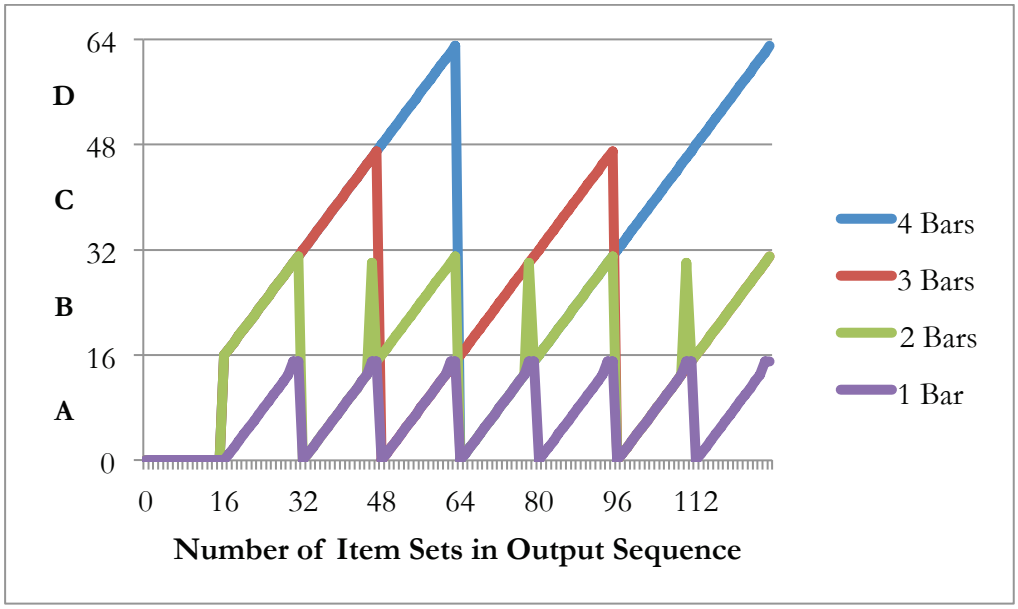


Figure 20: Initial S2MP plugin test - The training sequence length was increased from one to four bars.

The x-axis in Figure 20 represents the number of item sets in the output sequence created by the plugin during performance. The y-axis represents the item set positions within the training database, e.g., bar A of MIDI CC training

data is stored in item sets 0-15, while bar B is stored in item sets 16-31. In this case, if the algorithm was in performance mode and presented with an input sequence of AAAAAAAA, the target output sequence would cycle through item sets 0-15 in the training database, and would create a repeating ramp in the graph.

The results in Figure 20 show exactly this, and illustrate that the S2MP plugin was able to reproduce the target sequences for all four tests. As the length of the training sequence increased from one unique bar of MIDI to four, the number of item sets in the output sequence also increased. This is not surprising as the test input sequences were in the same order as the training sequences, thus making the S2MP search a matter of stepping through the stored examples, and creating an output of sequential item sets form the training database; however, it does show that the algorithm is satisfactorily outputting the proper sequences for Performer B given an input sequence from Performer A. Additionally, the graph also shows an initial empty bar in the output sequence from item sets 0-16. This is where the algorithm uses the input sequence to create the initial sequence of item sets used during the search. After this first bar has been created, the search sequence is complete and the algorithm will begin to return matches from the database.

3.4.7.2 MAPPING VS ORDER

The second set of tests explores the impact of the Mapping / Order weighting parameter. The parameter has eleven settings, biasing the algorithm to either rely more on the intersection of Item Sets or the order of the sequence. The plugin is trained on the same four bars of CC data used in the previous set of tests, and is shown only the single transition sequence of ABCDA. Once trained, the four bars of CC data are arranged into a new target sequence of BACADABA. This target sequence includes transitions not seen in the training database, and requires the S2MP algorithm to find generalized matches. Each test incremented the Mapping / Order weighting parameter, starting at 100% Mapping, and

ending at 100% Order. The window length is kept at one item set every 1/16th note, and the input sequence length is kept to sixteen item sets.

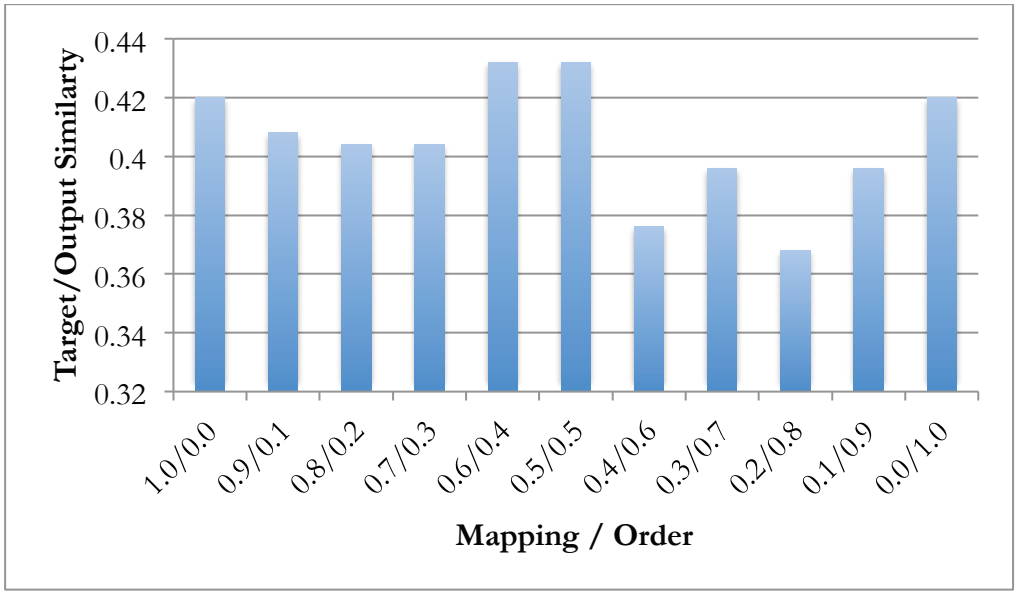


Figure 21: Similarity percentage between output sequence and target sequence based on Mapping / Order weighting

The results of the test revealed that 0.6/0.4 and 0.5/0.5 Mapping / Order settings returned the highest percentage target matching with 43.2%, while 0.2/0.8 scored the lowest target sequence matching percentage with 36.8% (see Figure 21). In general, the tests show stronger performance for matching target sequences using mapping of item sets compared with sequence order. This may be due to the introduction of untrained transitions in the target sequence. These untrained transitions force the algorithm to find a best match from new and unseen sequences in the target. While a setting that is biased towards the order of item sets will not handle these unseen sequences well, a setting biased towards mapping will fair better as it will be able to recognize similar groupings of CC data regardless of the order. In a musical scenario this ability to find matches that are similar in content but possibly different in order is important, as real world performance data will rarely look exactly like the training data.

As described in section 3.4.5, the search looks at the current beat within a bar, relative to the DAW timeline, and searches only that beat within every bar stored

in the database. These tests were trained on four bars of performance data; meaning that there are four possible values for every search, with a search being performed every 1/16th note. This gives a 1 in 4 chance of getting a target match, meaning that the 0.6/0.4 and 0.5/0.5 Mapping / Order settings performed 21.2% better then chance alone.

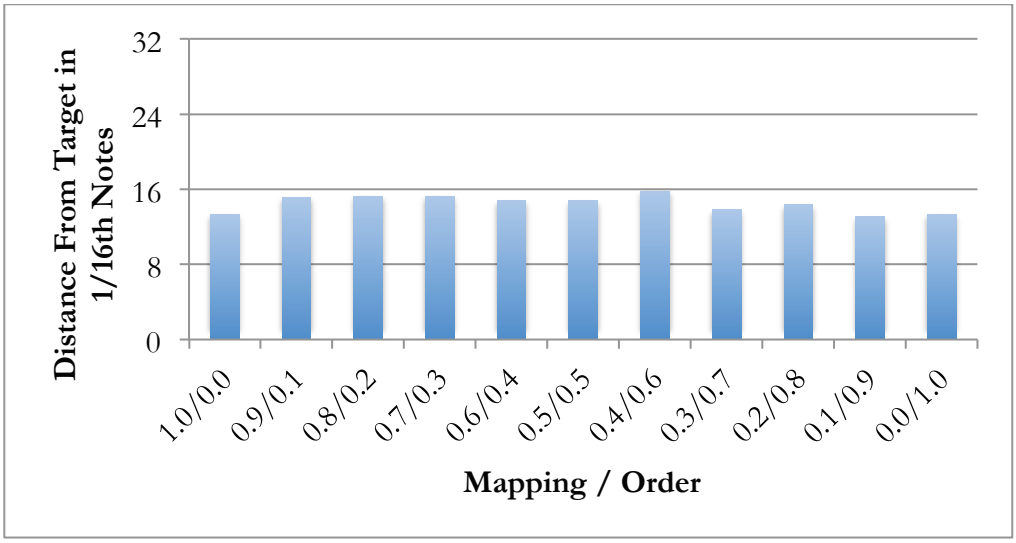


Figure 22: Mapping / Order - Average distance of output sequence from the target sequence

While the previous figure shows the overall similarity between the output and the target, Figure 22 shows the average distance between the two sequences. The average distance shows how close the output sequence was to the target sequence on average. This is an important addition to the similarity percentage between the two sequences as it gives an impression of how close the output sequence actually was to the target. An output sequence could have a 30% match with the target sequence, but on average be returning item sets from the training database that are 2-3 bars away from the target sequence. This would become a bigger issue as the size of the training database became larger. However, all settings of the Mapping / Order showed that the output sequence was on average one bar away from the target sequence, with a small bias towards the target. This is significant as it shows that when the algorithm did not accurately match the target, on average resulted in a sequence that was close to the target bar.

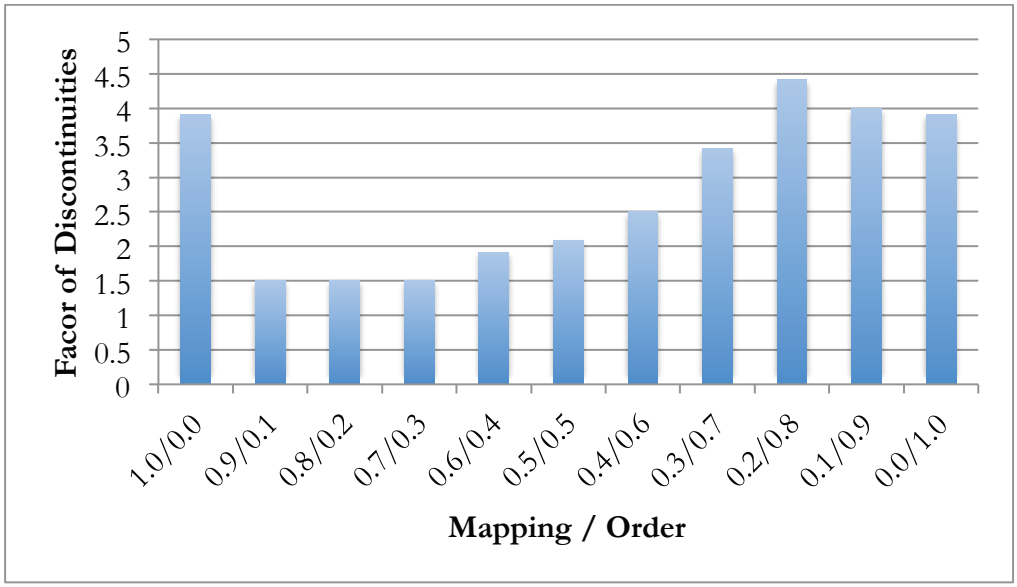


Figure 23: Factor increase of discontinuities between output and target sequence

Lastly, the output sequence was evaluated for the total number of discontinuities relative to the target sequence. A discontinuity is a break in the sequence of item sets produced by the algorithm. These breaks are usually found at the end of a bar when the target sequence jumps using a transition not present in the training sequences, e.g., the target sequence jumps from MIDI bar B to MIDI bar D. The target sequence used in the test contained a total of 12 discontinuities, thus a perfect matching output sequence having the same number. Up until this point, both the 0.6/0.4 and 0.5/0.5 Mapping / Order settings have shown the same performance; however, Figure 23 shows that a setting of 0.6/0.4 has slightly fewer discontinuities then 0.5/0.5. In practice the reduced discontinuities in the sequences amount to fewer jumps, and longer musical phrases from the plugin. Additionally, a factor slightly greater then one is to be expected as the algorithm cannot foresee the bar transition ahead of time, and may quickly attempt to modify an output sequence with a better matching sequence shortly into the new bar. This would lead to two discontinuities for every transition, with more than two implying that the algorithm is having difficulty creating longer phrases.

3.4.7.3 Number of Item Sets in Sequence

With the previous results in mind, a final set of tests evaluated the effect of changing the input sequence length, and the number of trained transitions

between MIDI bars. The tests were in two parts, first training using only one transition between MIDI bars and then training using three possible transitions. For each of these parts the number of item sets in the input sequence is set at 4, 8, 16, 24, 32, 40, and finally 48. The algorithm was trained on the same four bars of CC data used in the previous two tests. However, the algorithm was trained using only the single transition sequence of ABCDA for the first part of the test, and then later trained using the sequence ABCDACBDADCA for the second part. Once trained, the four bars of CC data were arranged in a similar manner as before, this time creating an even more complex target sequence of BACADABACDBACDBA. For the first part of the tests this target sequence included transitions not seen in the training database, and required the S2MP algorithm to find generalized matches, while for the second part of the tests the transitions were from the training database and should provide improved performance. The window length was kept at one item set every 1/16th note, and a Mapping / Order value of 0.6/0.4 was chosen based off the results of the previous tests.

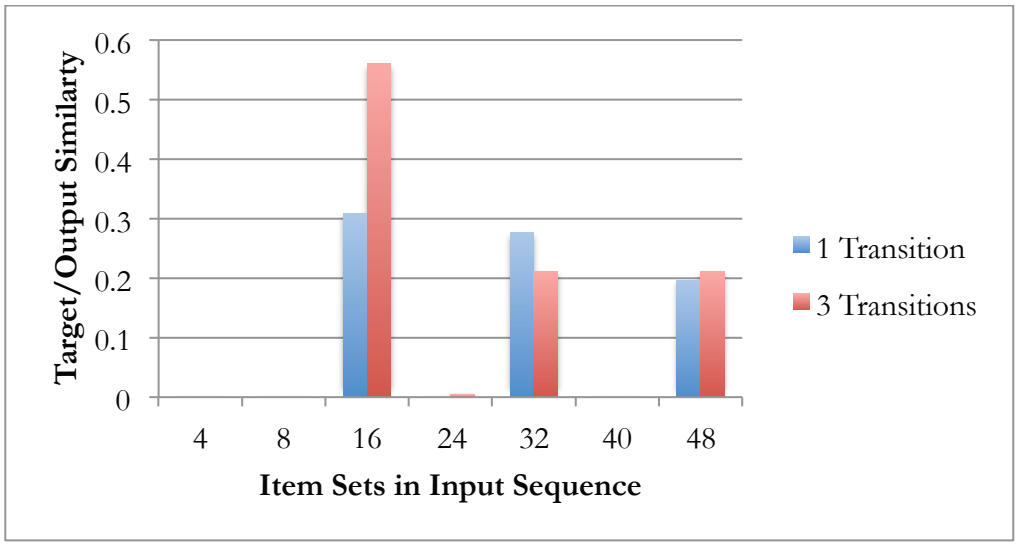


Figure 24: Similarity percentage between output sequence and target sequence based on number of item sets in the input sequence, and number of trained transitions

The results of the tests revealed that an input sequence of 16 item sets created an output sequence that best matched the target sequence. In general, the higher the bar along the y-axis, the better the output sequence actually matched the target

sequence. There was a 30.8% match for the single transition training set, and a 56% match for the training set with three possible transitions. These tests had the same 1 in 4 chance of matching the target sequence as the previous tests, this means that the three transition, 16 item set input sequence performed 31% better then chance.

Additionally, Figure 24 reveals that item set sequences that are not multiples of one bar returned near zero matches with the target sequence. This is due in part to the quantizing of the search to a distance of one bar. The effect of this causes item sequences of size 4, 8, 24, and 40 to have jumps that include fractional bar amounts, causing the sequence to flip flop between phrases. Lastly, input sequence lengths greater then the phrase length of one bar used during training seemed to perform poorly. This may be another side effect of quantizing the search to one bar jumps, or it may imply that S2MP is sensitive to the phrasing length during training.

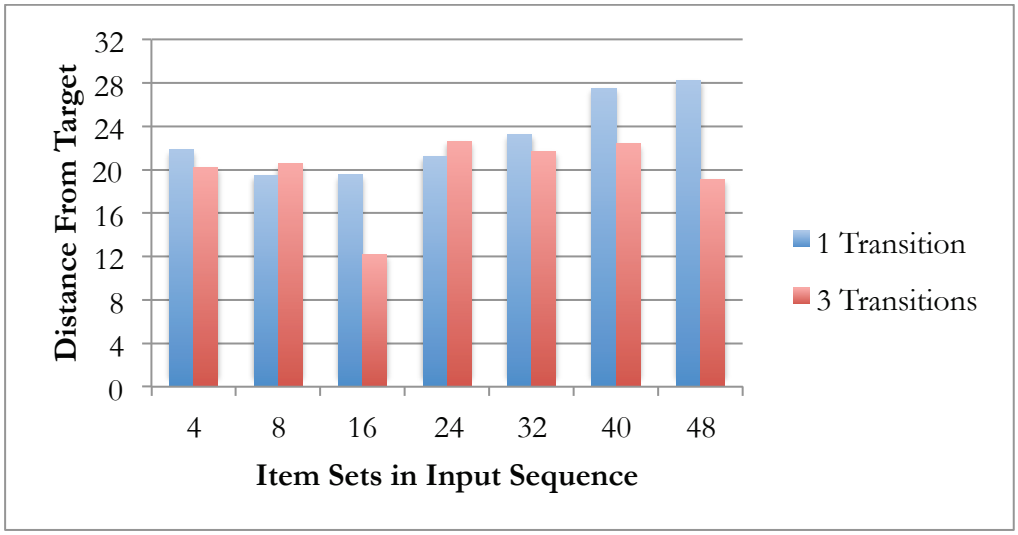


Figure 25: Increasing numbers of item sets - average distance of output sequence from the target sequence for input sequences

The tests also revealed a large drop in the average distance from the target for the three transition, 16 item set input sequence (see Figure 25). An Item set of 16 and 3 transitions resulted in a distance of twelve $1/16^{th}$ notes away from the target on average, and showed that the output sequence was less then one bar

away on average, meaning that not only did target matching increase but the output sequence was closer overall.

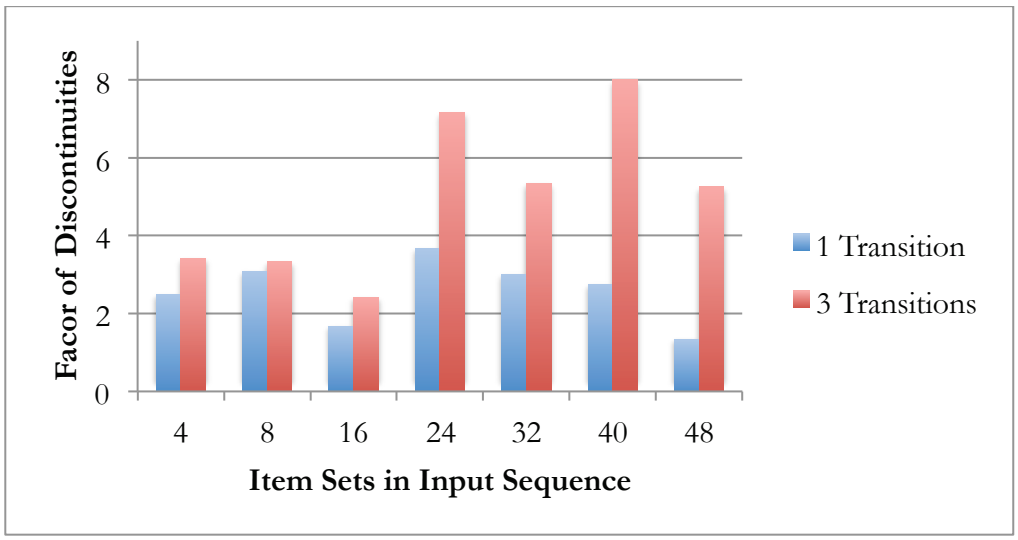


Figure 26: Factor increase of number of discontinuous matches by sequence size

Lastly, while the increase in transitions present during training improved the target matching for the 16 item set input sequence, it also increased the number of discontinuities. This can be seen as an increase in noise, and can be seen as an indication of the length of phrases created by the plugin. A factor of 1x could mean that there is a discontinuity (or a transitional jump) at the end of every bar, while a factor of 8x for a 32 item set input sequence (seen in Figure 26) would represent a discontinuity every 1/8th note within the bar. This is not to say that there would be an output sequence of only 1/8th notes, as some of the discontinuities may cluster together.

3.4.8 CHALLENGES WITH USING SEARCH BASED SYSTEMS

Search based approaches to designing interactive musical agents will always face a paradoxical issue. The issue being that as the amount of training data collected increases the model becomes more accurate; however, inversely as the number of searches increases the search performance becomes slower. As mentioned in Appendix E, systems that attempt to perform automatic accompaniment handle this challenge by restricting the search area to just before or after the assumed position with the fixed score. While this chapter has presented a similar solution

for improvisational systems that have access to a transport or tempo clock (see section 3.4.5), this solution requires assumptions that amount to compromises within the search. Essentially, in order to ensure the system is fast enough for use in real-time constraints are placed on the search, meaning that the algorithm may not return the overall "best" match from the database. This may be an acceptable compromise however, as a match that is "good enough" may provide a solid base for interaction between the human and the interactive musical agent.

3.5 DISCUSSION

The search based interactive musical system presented above represents only a single approach to designing interactive musical agents, with many other approaches to configuring the system existing. The following sections discuss several different methods for linking the input values to provide context for the system; the requirements for building an interactive musical agent within the context of each of the three approaches; and how these systems help to further the evolution of the performer composer.

3.5.1 ARCHITECTURE OF AN INTERACTIVE MUSICAL AGENT

In section 3.1, designing an interactive musical agent was framed as an attempt to use control change data from two performers, and then generate new material in the style of performer B given a context provided by both performer A and the output of the interactive musical agent (see Figure 9). Over the course of researching the design of interactive musical agents, there emerged several approaches to linking the inputs of the system in order to provide this contextualization of the model, each with advantages and challenges (see Figure 27). Three of these design architectures are presented.

The first, and most complex relationship is one in which the state of every control from both performer A and the virtual model of performer B influence the future state of every control in the model. A second approach allows the virtual model to be independent of performer A's state, and instead use performer A's input to apply a fitness function to the model's output. Lastly,

each control can be modeled independently, allowing for greatly simplified models; however, while statistically over time these separate models may individually show behavior indicative of Performer B, there is no guarantee that this behavior will emerge for these models as a group.

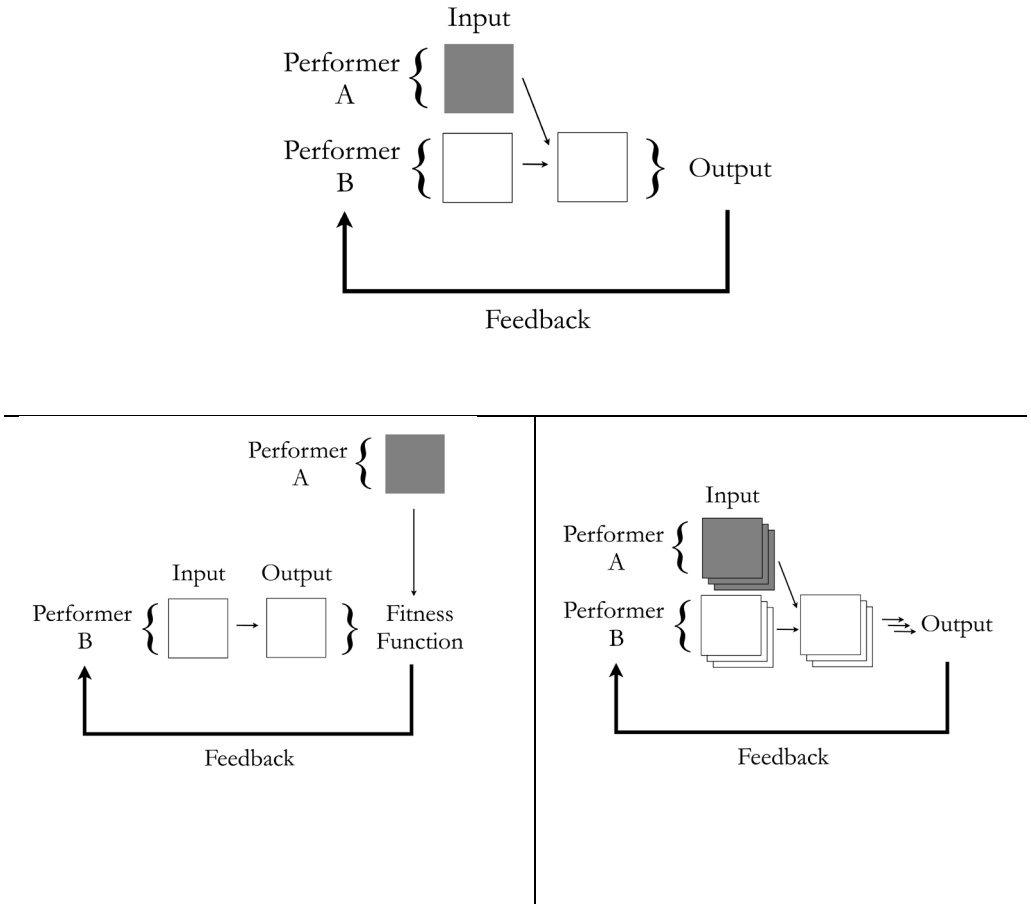


Figure 27: Different design approaches for interactive musical agents. Clockwise from top: (A) All inputs affecting the model's output; (B) The model is only affected by itself, and live input is applied as a fitness function; (C) Inputs are split into simpler individual models, all acting independent of each other

The first approach combines input from performer A with the previous output of the model (see Figure 27 A). This approach is inspired by the way human musicians listen to each other during improvisation. In order to decide what to play next during improvisation, a musician must listen to both what they have previously played, and what other musicians are currently playing. Implementing

an interactive musical agent that models this kind of listening can become challenging with large numbers of controllers being modeled, and increasing numbers of musicians within the ensemble. These situations can have large numbers of input parameters, and lead to models that require prohibitively large amounts of data during training.

The second approach to contextualizing the model is to only feed the model's output back into itself, and then apply a fitness function to the output using data from Performer A. This effectively eliminates the input parameters provided by performer A, shrinking the state space and simplifying the model (see Figure 27 B). While this approach simplifies the model, there are also complications. By applying a fitness function to the output of a probabilistic model, the likelihood of a getting a given sequence becomes altered, effectively changing the model itself (Pachet and Roy 2011; Pachet, Roy, and Barbieri 2011). Similarly, the fitness function may not accurately represent the complex interaction between two human performers; however, an accurate model may not ultimately be the most important factor, but rather the quality of interaction, or new modes of performance afforded by the model may be most desirable. If this is the case, then experimentation with parameters for a fitness function may be an effective solution.

Lastly, a third approach allows for the most simplified models, as the domain only needs to describe the relationship of a single control to its prior state (see Figure 27 C). This model is not capable of capturing the inter-dependencies of controller states on the state of other controllers, e.g., the value of a filter's resonance may be dependent on the value of the filter cutoff. In the end, it may be that the models will statistically perform in the style of performer B, and the complex inter-dependencies between different controllers will simply emerge.

3.5.2 THE ARMY OF ME

This chapter has presented a new similarity search algorithm for continuous control, based off of work done by Martin (Martin et al. 2011), shown this algorithm in use as a plugin within Ableton Live, and presented architectures for

defining relationships between inputs and the model. Through the development and use of these kinds of systems, computer musicians are able to extend their expressive potential using virtual personas. This extension represents the evolution of the composer performer as it creates an instrument that performs with the musician, creating an improvisational feedback loop between the human and the system, and allowing a single physical action to become multiple, distributed musical events.

While these interactive agents afford the performer composer the ability to further distribute their musical intent into the machine, they also have the potential to create an increasingly disembodied performance. Audiences may not currently be comfortable with this disconnect between the musician's physical actions and the interaction with the musical agents, and a more embodied approach may help to bridge this gap. The following chapter examines networked music ensembles, and how they provide computer musicians the opportunity to create this sense of embodied performance through interaction with other human musicians and the use of shared social robotic instruments.

Chapter 4

THE ART OF COMMUNICATION: SHARED INSTRUMENTS AND NETWORKED MUSICAL ENSEMBLES

> "The concept of a musical instrument designed to be played simultaneously by more than one person is not new, but there are very few examples in the history of western music, other than the piano. With local high-speed computer and sensor technology, a new universe of possibilities has been unveiled…"
>
> —Alvaro Barbosa (2003)

Computer networks facilitate the exchange of information between computer musicians, creating new forms of musical communication. This information can be in the form of sensor readings, algorithms, text, or whole programs. Exchanging data allows computer musicians to share control of their instruments with every other member of the ensemble, essentially making the musicians a part of the larger network instrument. The data passed over the network can be manipulated by other musicians, or used as input for creating an emergent music from network algorithms. This model of networking musical information represents an evolution of the performer composer, and creates entirely new worlds of musical performance.

The aim of this research is to explore this potential for networked musical ensembles to become shared social instruments in the hope of developing new modes of interaction for live computer music. Specifically, this chapter will look at how The KarmetiK Machine Orchestra has created a shared social instrument using networked musical robotics. These robotic instruments are accessible to every member of the ensemble, and provide a way to directly embody the actions of the computer musicians through the physical movements of the robotic actuators (A. Kapur et al. 2011). Analogous to several performers playing on a single piano, the shared instrument allows multiple performers to express themselves independently within a social context (Barbosa 2003). Through the use of a central server, the musicians are able to control the shared robotic instruments both at the note-level using hardware interfaces for musical expression, and simultaneously at the score-level through software sequencers communicating over the network (see Figure 28).

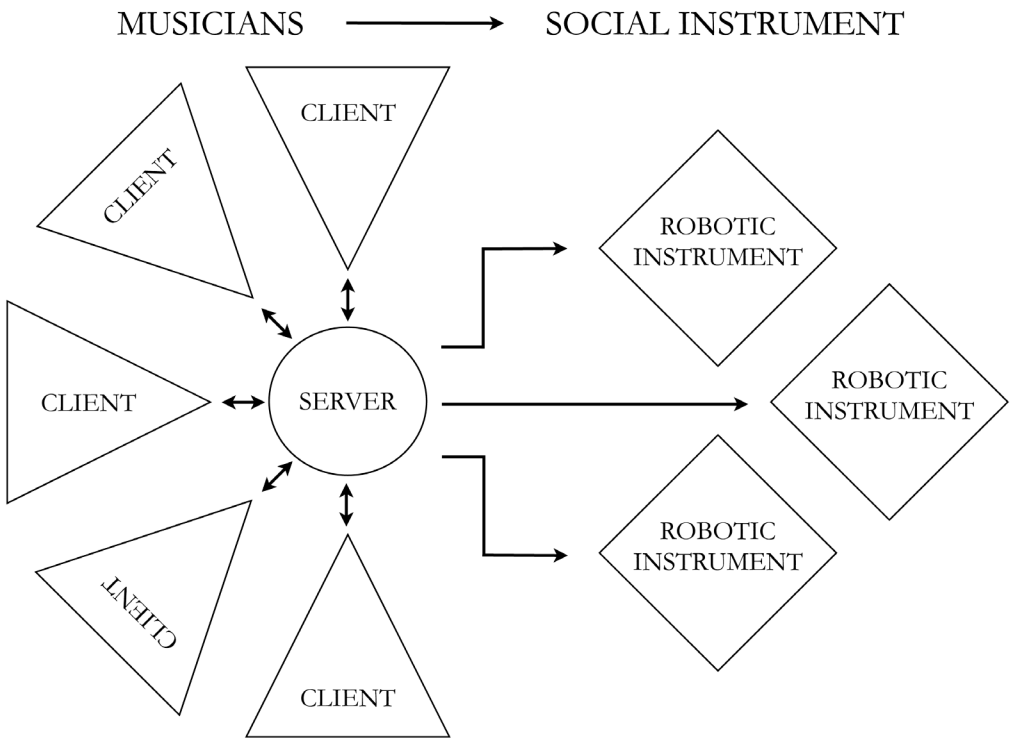


Figure 28: Network topology of The Machine Orchestra ensemble

The previous chapter explored the challenge of simultaneously controlling many musical parts by developing autonomous accompaniment systems that react to a human musician; this chapter approaches the challenge through networked music ensembles and shared social robotic instruments. Networked musical ensembles provide a solution to controlling many parts by dividing the performance among several different musicians. This reduces the scope of musical control that a single performer is responsible for, while also helping the audience to connect with the performance through the embodied interactions between musicians. These interactions between musicians amount to externalizing parts of the performance, thereby helping the musicians to more effectively communicate with the audience. Additionally, the use of shared robotic instruments provides a physical point for this musical interaction between performers. Physicality in performance has historically been a challenge for electronic music (Bahn, Hahn, and Trueman 2001), but combining network ensembles with musical robotics creates a new social instrument with which to address these issues.

This chapter examines these issues through the work of The Machine Orchestra, and begins by presenting a historical review of networked computer music performance. This review examines the precedence of networked musical ensembles as social instruments, and serves to illustrate how The Machine Orchestra stands on the shoulders of historic network ensembles. The chapter then examines The Machine Orchestra's extension of the shared social instrument to include musical robotics, and the use of this shared instrument within the context of several The Machine Orchestra compositions.

4.1 BACKGROUND

Figure 29: The League of Automatic Composers 1980

Over the past 35 years, network music ensembles have evolved into a rich and diverse field of research and exploration for computer musicians. The idea of a musical network has grown to encompass a wide range of topologies and configurations, from small local networks of performers sharing data between each other (Bischoff, Gold, and Horton 1978; Gresham-Lancaster 1998; Smallwood et al. 2008), to large ensembles of computer musicians half a world apart (Cáceres et al. 2008). During this same time, research has also explored systems designed to overcome the challenges inherent in high latency, low-bandwidth communication (Lazzaro and Wawrzynek 2001; Chafe and Gurevich 2004; Barbosa, Cardoso, and Geiger 2005; Cáceres and Chafe 2010; Driessen, Darcie, and Pillay 2011).

Founding network computer musicians were interested in the potential of networks to connect and share data. Early experiments by The League of Automatic Composers (Bischoff, Gold, and Horton 1978) involved three networked microcomputers (KIM-1), each with its own custom software instrument, all sharing control data. This process of allowing the performers to control each other's instruments created an ensemble that had never before been

possible, and led to the creation of a shared and social instrument, diffusing the absolute control a musician traditionally had over his or her own instrument. The members of The League named this new style of music "Network Computer Music", and continued to explore the possibilities of this new style until 1986.

The Hub formed out of the pioneering efforts of The League, and expanded on existing research using the newly developed MIDI protocol. In his 1998 article on the aesthetics and history of The Hub, Scot Gresham-Lancaster explains how "the advent of both the microprocessor and the affordable, multi-parameter, controllable MIDI synthesizer made possible a new type of network-based performance" (Gresham-Lancaster 1998). He goes on to suggest a link between this new type of network-based performance and the process music of composers such as John Cage, David Tudor, and Pauline Oliveros. The processes that Lancaster alludes to are a product of the rules governing the ways in which performers share data through various network topologies and algorithms.

This potential for performers to share musical data through networks has become one of the central focuses of networked computer music, with Brian Kane going so far as to say "Any aesthetics of Net music would, correspondingly, imply a set of musical practices that exploit these (and other) specific affordances of networks" (Kane 2007). Additionally, while developing a classification framework for describing the multitude of possible network ensemble interconnections, Gil Weinberg states that he attempted "… to map the field based on what [he sees] as the central innovative concept of the medium: the level of interconnectivity among players and the role of the computer in enhancing the interdependent social relations" (Weinberg 2005). In developing his framework, Weinberg renamed Network Computer Music to Interconnected Music Networks, reflecting this focus on interconnectivity.

This focus on interconnectivity can be found in many different networked music compositions. The Hub's 1991 piece *waxlips* (Tim Perkis,) is based around a rule regarding the way a client requested notes from the master machine. Lancaster

describes *waxlips* as "an attempt to find the simplest Hub piece possible, to minimize the amount of musical structure planned in advance, in order to allow any emergent structure out of the group interaction to be revealed clearly" (Gresham-Lancaster 1998). This "emergent" behavior results from sharing of musical data between performers, ultimately providing each individual musician's contribution as musical source material for the entire group.

More recent pieces by Princeton's Laptop Orchestra PLOrk (Smallwood et al. 2008) examine the way in which data can be passed around a network using a wireless router and topologies as complex as peer-to-peer interconnectivity. Ge Wang' composition *Clix* explores the use of the network to quantize all musical output to a common pulse rate, thereby tightly synchronizing the musical output of a large ensemble. Dan Trueman's *The PLOrk Tree* explores using a tree structure to propagate musical ideas throughout an ensemble. Much like a game of telephone, the musical information received by performers at the edge of the tree is a modified version of the original idea performed by the conducting computer at the root of the tree. The concept of sharing musical data with other members of a network ensemble, and allowing them to alter or modify it, is central to many network compositions as far back as the League of Automatic Composers. These pieces share a common algorithmic approach to composition, where rules govern the way in which an interconnected ensemble of musicians share and manipulate performance data. These rules for sharing data turn the network itself into a shared social instrument.

4.2 PHYSICALITY IN COMPUTER MUSIC PEFORMANCE, AND EXTENDING SHARED CONTROL TO MUSICAL ROBOTICS

As described in the previous section, existing network ensembles afford computer musicians unique ways to share control over each other's instruments. This democratic approach to performance creates opportunities for new modes of interaction, such as social games and algorithms requiring input from the

entire group. These approaches can lead to emergent behavior from the ensemble, and create music that is not a direct result of any one performers actions, but truly dependent on the sum of all the actions of the musicians.

As the ensemble's musical control becomes more diffuse, the connection between the music and the listeners—both ensemble participants and audience members—potentially becomes unfocused and difficult to ascertain (Gresham-Lancaster 1998). Weinberg describes this as one of, "the field's main drawbacks, in [his] opinion, stem[ming] from the focus that was put on complex interdependent connections which forced participants and audiences to concentrate on low-level analytical elements in order to follow the interaction" (Weinberg 2002). He goes on to say in a later article, "these networks posed high entrance barriers for players by requiring specialized musical skills and theoretical knowledge in order to take part in and follow the interaction in a meaningful manner" (Weinberg 2005). One of the original members of The Hub also expresses a similar idea stating, "the audience was often mystified by what they heard in relation to what they saw the performers doing" (Gresham-Lancaster 1998).

Weinberg suggests that, "the design of expressive gesture-based interconnected instruments… [would provide] participants with an expressive as well as coherent access to complex interdependent network topologies, which will allow them to focus on the artistic aspects of the experiences" (Weinberg 2002). The instruments proposed by Weinberg would provide a strongly embodied link between the performers' actions and the music created. The Machine Orchestra's implementation of this idea is a shared social instrument comprising an array of custom built electro-mechanical instruments.

4.3 Musical Robotics and The Karmetik Machine Orchestra

Since 2009, The Machine Orchestra has been performing as a music ensemble using a local network, exploring the concept of shared musical robotic instruments. The creation of The Machine Orchestra came out of the thesis work of Ajay Kapur (A. Kapur 2007) which in part explored the mechanization of classical North Indian instruments. This led to a class at CalArts in 2008 where students designed and built a set of robotic instruments based off the technology used in Kapur's original work (A. Kapur et al. 2011). During this class, the author built a musical robot named Tammy, in collaboration with Jordan Hochenbaum, Carl Burgin, Steve Rusch, and Jeff Lufkin (see Figure 30). Tammy consisted of hand carved marimba bars, metal bells, and a metal string resonator (A. Kapur et al. 2011).

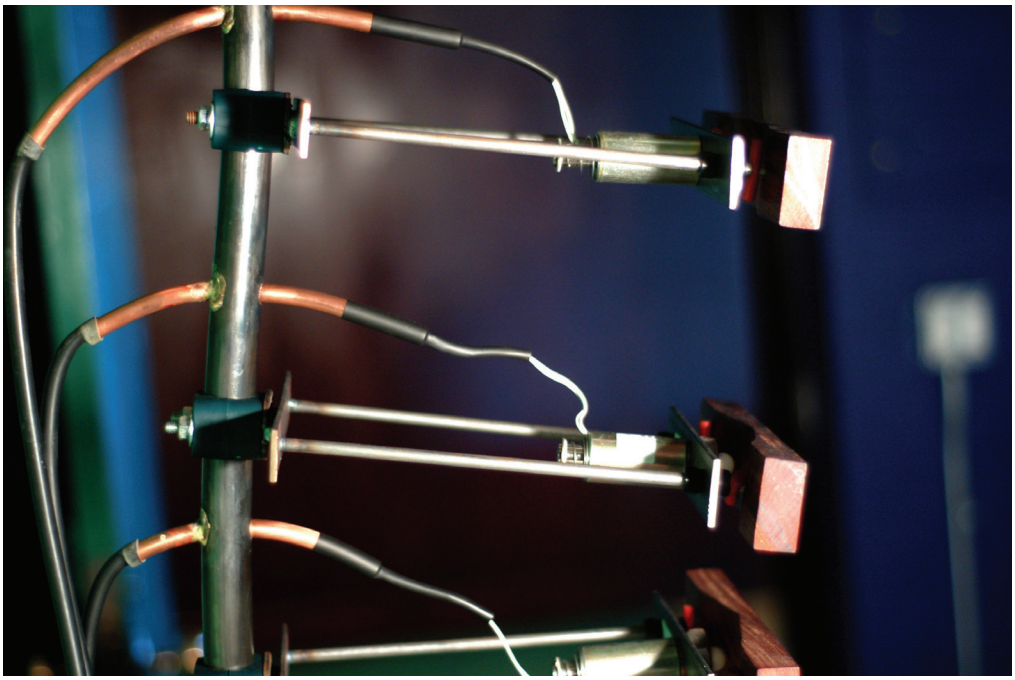


Figure 30: View of the marimbas from the musical robot Tammy

Shortly after this class, the original Machine Orchestra developed as a combination of laptop orchestra and musical robotics (Aj. Kapur, Darling, and Kapur 2012). Each of the musicians in the ensemble connects through a central

server, to all the other musicians and to the shared robotic instruments (see Figure 28). This configuration allows for the traditional interconnected network topologies, as well as configurations where the musicians are unlinked and only share musical control over the robotic instruments. Initially, the actuators of the drums were divided up during composing, with performers often sharing control over the same mechanized instrument. This division of control was utilized in a composition called Mechanique, where members of the ensemble wove a polyrhythmic texture using the robotic drums. As many as three or four players could be sharing the same robotic instrument during certain section of the piece.

As The Machine Orchestra evolved over the years, several new modes of control were explored. One notable approach involved using research from the previous chapter to write an algorithmic drum sequence for the robots. At the appropriate section of the song, the robots took a "solo" which increased in in intensity until suddenly changing back to shared control at the following section. The solo was not pre-programmed, and instead was an algorithm that allowed the robots to increase their playing intensity with the human performers during the specific section. This system represents an effort to integrate the ideas from Chapter 3 into networked music ensembles.

Lastly, the use of shared robotic instruments in The Machine Orchestra has provided a means to realize embodied performance. The physical actuators of the musical robotics act as a bridge between action and sound. Although the mappings may still remain complex and diffuse, the audience has responded positively to the physical movements of the robots when compared to a performance using computers only. This may be attributed to enculturation of the audience, with the physical robots providing a link between the physicality of an acoustic ensemble and the distributed mappings of a computer only ensemble. Additionally, from a performance perspective, the spatialization of the robotic instruments, and shared control over them leads to an ensemble experience that is uniquely different from performing with computers alone.

4.4 COMPOSITIONS AND PERFORMANCES

The following section presents several concerts that the author performed in over the course of this thesis, and illustrates how members of The Machine Orchestra:

- Compose and perform using the musical robotics as a shared social instrument.

- Embody and communicate musical interactions to the audience via the use of custom musical controllers and the musical robotics.

- Perform in multiple interaction contexts.

These concerts were performed with several different configurations of The Machine Orchestra, and as such provided an opportunity to experience networked ensembles of various sizes. Additionally, the musicians of The Machine Orchestra come from diverse musical backgrounds. These factors created the opportunity to learn, compare, and contrast performing both solo computer music, and within networked ensembles. A survey of the musicians involved in the following concerts is discussed in Appendix C, and looks at some of their thoughts and ideas regarding solo live computer music versus networked computer music ensembles.

4.4.1 JANUARY 27, 2010 REDCAT - THE MACHINE ORCHESTRA

The 2010 REDCAT show represented the premier of the full KarmetiK Machine Orchestra ensemble, comprising of Ajay Kapur, Perry Cook, Curtis Bahn, Jordan Hochenbaum, Jim Murphy, Carl Burgin, Maeson Wiley, Dimitri Diakopolous, and the author. This production represents a fusion of networked computer music, musical robotics, and world music; allowing for solo and group improvisation, as well as performance using a shared robotic instrument. This concert also presented the opportunity to test network MIDI clock sync for

more than two performers, providing an opportunity to compose rhythmically based music with the parts distributed among the ensemble (see Figure 28).

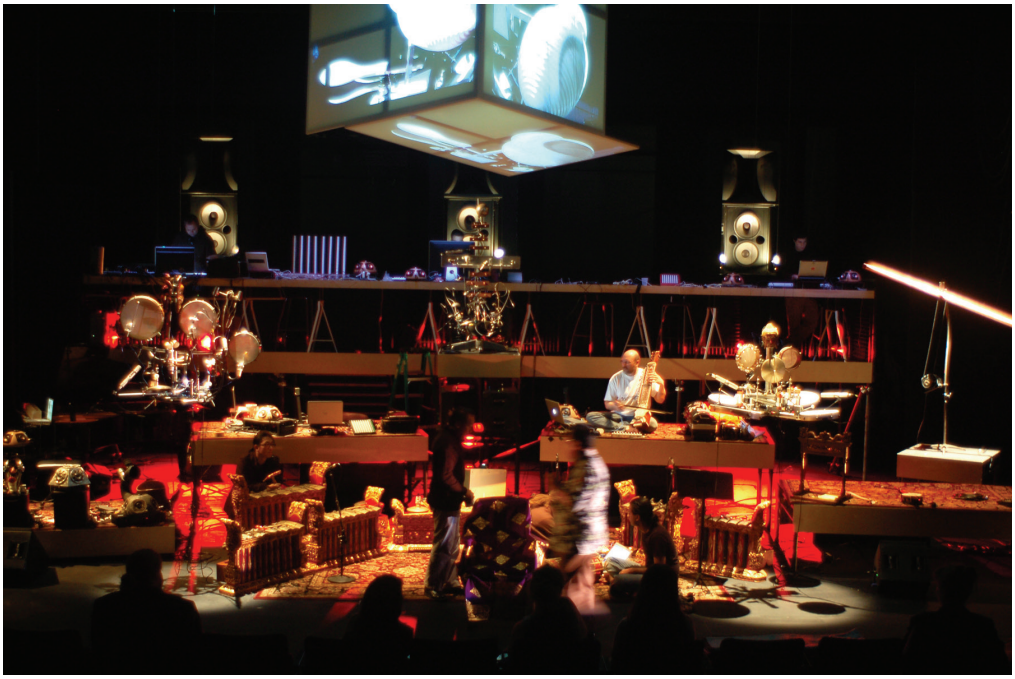


Figure 31: The Machine Orchestra at REDCAT 2010

4.4.1.1 COMPOSITIONS

The concert consisted of a number of pieces, and was made of three main sections. The first part of the concert consisted of the networked computer ensemble and the musical robotics; this was followed by a performance with the Machine Orchestra and the renowned classical north Indian musician Aashish Kahn; lastly, The Machine Orchestra ensemble left the stage and a piece by a Gamelan ensemble with a robotic Reyong was performed.

The first piece to have the entire ensemble use the musical robotics was called Mechanique (briefly described in section 4.3). Mechanique explored the idea of the shared social instrument by creating a many-to-many relationship between the performers and the robotic instruments. Musicians were not assigned to a single robotic instrument, but rather each musician was assigned several actuators across multiple robots; as a result, the network afforded the musicians the ability to perform simultaneously on all of the physically separated robotic instruments. This is a unique feature of working with the shared social

instrument that would be difficult to achieve for traditional acoustic ensembles. The ability to divide a single musical instrument across multiple performers enables complex musical interactions, e.g., each performer plays a simple rhythm on their controller, which are then summed together at the robot to create complex interlocking beats. This idea is similar to styles of Indonesian music, where instruments in the gamelan such as the Reyong are situated amongst 2-4 players in order to achieve extremely fast polyrhythm.

Although Mechanique is an improvisatory piece that encourages performer spontaneity (A. Kapur et al. 2011), there is a higher-level structure dictating when musicians enter the piece, as well as the dynamics. Mechanique begins by sparsely introducing the various robotic instruments, and then growing denser as the ensemble begins to play more of the actuators. The piece gradually crescendos until the performers all simultaneously play a final abrupt note. Additionally, the physically separated, percussive robotic instruments are given a sense of spatial coherency using instrumental drones and synthesized textures provided by several of the performers. These ideas are central in providing the piece with a simple core against which the complex robotic improvisations can contrast.

A key goal of Mechanique was to bring in the robotics slowly, thereby introducing the audience to the robotic instruments and the performer's various custom interfaces and controllers. This process allowed the audience to identify individual performer's actions with the physical sound producing actuators of the robots, and in doing so, reinforced the potential for embodiment represented by the robotic instruments. Additionally, the piece was highly rhythmic and improvisatory in nature, requiring accurate timing and sync between the musicians and robotic instruments. This timing allowed for both pre-composed rhythms, while also providing the opportunity for more improvisational explorations. Syncing computers to a master MIDI clock allowed for triggered MIDI sequences to be in rhythmic time with the rest of the ensemble. These sequences were controlled through various interfaces such as the Arduinome (Vallis and Kapur 2011), the Helio (Murphy, Kapur, and Burgin 2010), and the

Multi-Laser Gestural Interface (MLGI) (Wiley and Kapur 2009) which provide visual feedback to the audience through physical interaction and LEDs. At the same time that these sequences were being triggered, it was also possible to manually play individual actuators on the robots. This was achieved by mapping the controllers to both sequences, and individual actuators.

The second section of the concert consisted of the pieces Sitka, Moksha, and Twilight. Although all performers were locked to a synchronized clock, in these three pieces they had the ability to move freely through the arrangement by launching different groupings of loops. This is similar to Terry Riley's *In C* (1964) in that the loops can be thought of as cells that represent different sections of the song. In a similar manner to Barbosa's description of non-improvisational music (Barbosa 2003), the pre-composed pieces dictated what grouping of loops all performers should be playing at a given point in the piece, but there remained the flexibility to vary or manipulate the arrangement by moving within the small group of currently available loops.

The main ensemble was split into three distinct groups: group A performed harmonic and melodic material; group B added more timbre and gesture based sound material to the pieces; Group C was responsible for all the drum and percussion parts within the pieces, providing these elements while performing in several different interaction contexts. These contexts consisted of a combination of audio loops sent to the speakers, MIDI loops sent to the robots, manipulation of the audio material through custom Reaktor software, and note for note performance of samples and/or the robotic instruments. Much of the percussion material composed for the piece was polyrhythmic, with the perceived groove of these parts being highly dependent on the synchronous timing between two of the performers. This proved challenging as the network seemed to incur clock jitter, smearing the timing relationship between the rhythmic parts.

Lastly, control over the robots was split between several musicians. These musicians were able to send pre-recorded sequences to the robots, while also having note-level control over individual actuators. This would be similar to a

player piano having two or three people playing along with the automated paper score. Additionally, the individual note-level control over the robots allowed for a single robot to be used for call and response sections between musicians in the ensemble. This interaction and performance used the same musical interfaces as Mechanique, but utilized completely different mappings. This flexibility in interface mapping is one of the main evolutions of the performer composer, allowing a physical interface to become virtually any instrument.

4.4.1.2 OUTCOMES

This concert was the debut of The Machine Orchestra, and represented the first public opportunity to perform in a networked ensemble using interfaces such as the Arduinome, the Helio, the Multi-Laser Gestural Interface (MLGI), and musical robotics. The concert resulted in a unique opportunity to improvise as a solo live computer musician, using all of the interaction contexts previously described in this thesis, while also engaging in musical dialogues between computer musicians. Interestingly, when these musical dialogues were between the author and another performer, the improvisation felt like two distinct voices; however, when more then three members of the ensemble simultaneously performed on the shared robotic instruments, there was a very different connection to the musical dialogues. The improvisational ideas became less driven by call and response interaction, and increasingly driven by a desire to become a part of a texture or system. This appeared to be a product of the network having the potential to merge multiple musicians' musical intent into a single instrument, relocating the actuations of the performers to a single physical location.

4.4.2 AUGUST 14, 2010 – KARMETIK COLLECTIVE

A Machine Orchestra ensemble, consisting of Ajay Kapur, Curtis Bahn, Jordan Hochenbaum, Jim Murphy, and the author, performed in Auckland New Zealand on the 14th of August 2010. The piece Tarana was composed for the Auckland performance, and is notable for its use of the algorithmic improvisation section described in section 4.3. Composed in a similar manner to

Sitka Chant, Moksha, and Twilight, the piece was a combination of pre-composed loops and improvised material.

4.4.2.1 COMPOSITION

The performance opened with an alap, a traditional North Indian classical improvisation that introduces the melodic ideas used in the composition. As this introduction came to an end, all the performers and robots simultaneously entered the piece. This entrance marked the start of the common clock shared between the robots and the human performers, and set up the piece to move towards an improvised middle section. Once this middle section was reached, the robots were signaled to begin a generative improvisation program, with the human musicians responding to the robot's performance. The generative drum section was incorporated directly into a ChucK (Wang and Cook 2003) based client server application. At the appropriate moment in the score, a message from an Arduinome interface was sent to the server and activated the generative process. The section began with the robots playing sparsely, and then slowly crescendoed to a flurry of rhythmic activity on the drums. Finally at the end of the improvised section, all performers and robots simultaneously move back to the main theme introduced in the alap, effectively ending the robots' autonomy.

Tarana is unique among the Machine Orchestra compositions in that it contains a role reversal between the human musicians and the robotic instruments. For one small section of the music, the robots are leading the performance, with the humans responding to the musical ideas being generated. This creates an interesting example of computer musicians performing in multiple interaction contexts, where they are not only playing individual notes, and effecting the sound from their computers, but also simultaneously responding to semi-autonomous musical robotics.

Additionally, several of the pieces from the original 2010 REDCAT production were performed in this concert, but required new arrangements in order to work with the smaller ensemble. This process of creating new arrangements began with a discussion regarding the new roles each of the musicians would play. This

discussion resulted in an increase in parts for each musician, completely re-imagining the original compositions for performance by the smaller ensemble. This re-imagining of the works is interesting in that it illustrated the versatility and flexibility of the computer as an instrument.

4.4.2.2 OUTCOMES

This concert explored the use of generative approaches for controlling robotic percussion. The system provided the opportunity for control of the robots on a score-level, while simultaneously allowing them to generate new musical material on top of which the ensemble could improvise. Tarana represents a clear example of the unique interactions that live computer music affords, and the way in which this leads to an evolution of the performer composer. This interaction can be seen as a simpler version of the systems described in Chapter 3. While it was interesting to respond to the generative material played by the robots, it is my hope that future development of interactive musical agents will lead to a full two way musical dialogue between human and machine.

4.4.3 APRIL 12, 2012 REDCAT – SAMSARA THE MACHINE ORCHESTRA

The Samsara concert is the biggest Machine Orchestra production to date. Bringing together music, technology, animation, production, and dance. The development and production was the work of Ajay Kapur, Michael Darling, and Raakhi Kapur (Aj. Kapur, Darling, and Kapur 2012). The bulk of the concert consisted of several networked ensemble pieces that heavily utilized the shared social robotic instruments, and aimed to build off of ideas from previous Machine Orchestra perfromances. In addition to these pieces were works by visiting artists Trimpin, Curtis Bahn, Tomie Hahn, and Jeff Aaron Bryant.

Figure 32: The Machine Orchestra performing Samsara 2012

4.4.3.1 COMPOSITION

Seminya was the opening piece, and was a reconceptualization of contemporary Bhangra music and Bollywood themes for the Machine Orchestra. This piece was composed in a similar style to earlier machine orchestra pieces. The pre-composed loops were used to move through a loosely defined arrangement, while the musical robotics and computer generated sounds were improvised over top. For the closing of the concert, the pre-composed material for Seminya was used as source material for a new completely improvised piece. This resulted in each musician selecting several loops from the original composition and then creating an entirely new piece of music by modifying and manipulating the source material in real time. Similar to re-mixing and sampling techniques, the manipulation of the original musical material became the instrument the ensemble played. In this way, prerecorded material can be thought of as the written score, and manipulating that material can be thought of as the improvisational space, becoming an instrument in its own right. The Seminya reprise is an important piece for The Machine Orchestra as it illustrates how a networked ensemble can have musical dialogues as acoustic ensembles do, but can also leverage different interaction contexts to create entirely new interpretations of the musical material.

4.4.3.2 OUTCOMES

This concert represents a fusion of the ideas presented in this thesis. While many of the pieces used all three interaction contexts–with concepts and ideas learned from previous performances of The Machine Orchestra–the final piece illustrates these in a special way. The aim of this final piece was to take the material used in the opening number of Samsara, and then improvise an entirely new piece of music. This could be described as a reprise as it brought back the musical ideas at the opening of the concert, but it also represents a uniquely computer music approach to this process. Several rehearsals were spent without any fixed compositional requirements or ideas, and instead focused on improvising and reworking the material to find new sounds and ideas. While many forms of live music reinterpret songs in this way, The Machine Orchestra's approach actually built the new interpretation using the audio from the original as the instrument. The result of this is that the reprise is actually built from the opening piece; not just the musical ideas, but also the musical material itself.

The reprise stands out as a moment where all three interaction contexts were in use by the members of the Machine Orchestra. Each musician took the original material as their starting point and played new notes (Note-Level), processed the audio to create entirely new sounds (Effect-Level), and rearranged the scored material down to the very beat level (Score-Level). The result of these interactions allowed the ensemble to perform the original composition as the instrument itself.

4.5 DISCUSSION

This chapter has presented a historical overview of networked music ensembles, and their exploration of interconnectivity. The field of networked music has lead to new forms of live performance that include the use of social configurations enabling music through games, and algorithms utilizing the multiple streams of input data coming across a network. The Machine Orchestra was presented as a contemporary example of networked music ensembles, and the development of

a shared social instrument in the form of musical robotics was presented as a new contribution to the field. The use of this shared instrument, and the types of interactions within the ensemble were presented in the description of several pieces performed by The Machine Orchestra. These pieces illustrated how pre composed music, with the parts being distributed among the ensemble, enables musicians to improvise with greater detail and focus then is possible as solo performers. Additionally, the pieces also showed how these larger ensembles can help issues of embodiment, by using the multiple performers to bridge the gap between the audience's perceptions of the performer actions, and the actual sounds being produced. Lastly, the use of shared social instruments leverages the networks ability to share information between computers. This allows the musical robotics to act as a physical point at which all members of the ensemble can perform and control musical expression. The use and interaction of this shared instruments creates new modes of interaction within an ensemble, and opens new worlds of performance.

Chapter 5

CONCLUSION

> "No people could live without first valuing; if a people will maintain itself, however, it must not value as its neighbour valueth.
> Much that passed for good with one people was regarded with scorn and contempt by another: thus I found it. Much found I here called bad, which was there decked with purple honours."
>
> —Friedrich Wilhelm Nietzsche (1896)

This chapter presents a summary of the ideas explored in this thesis, tying them together to make the case that live computer music is an evolution of the performer composer. Additionally, this chapter presents the main contributions from the previous three chapters. Finally, an overview is presented of my philosophy of live performance that has developed as a result of this research, and the future work it will lead to.

1 SUMMARY

Much of this thesis has presented interaction contexts (see section 1.1) as the basis for understanding how live computer music has lead to an evolution of the performer composer. These interaction contexts describe different modes of performance; including note level, effect level, and score level interactions. Live computer music's ability to automate tasks has now enabled musicians to simultaneously perform in more than one context. This simultaneous use of interaction contexts amounts to the propagation of the performers will, stemming from a single physical action, and disseminating through a system into

multiple musical events. As the move from monophony to polyphony was an evolution of melodic composition, so is the move from acoustic performance to computer-mediated performance an evolution of the performer composer. This thesis explores this evolution in three key areas of live computer music: interface design, interactive musical systems, and networked computer music ensembles.

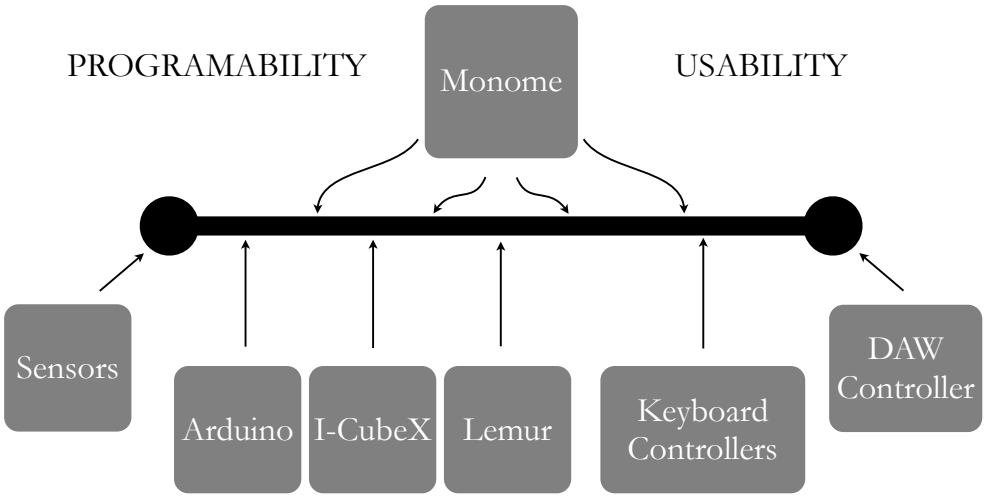


Figure 33: The Monome can be both highly programmable or immediately usable

This thesis has shown how online communities have impacted the process of designing new interfaces for musical expression, providing technical resources, musical software, and iteratively modifying devices to allow for new mappings between the hardware and software. The development and use of these interfaces speaks to the decoupling of physical action and sound actuation. Even if a computer musician does not design their own custom controller, they still map the physical device to virtual controls inside the computer. These mappings allow a single device to potentially control an orchestra of virtual instruments. This process of mapping makes every computer musician a digital luthier, both crafting the instrument and performing with it. This represents an evolution of the performer composer, as virtuosic computer musicians now master both performing and mapping of their physical interfaces.

However, there is a downside to this highly customizable approach (Cook 2001). The increase in modularity requires an initial investment in order to set up the

desired functionality, such as learning the technical languages and skills required to understand the system behind the controller. This allows the user to create a custom interface but also creates an initial decrease in "plug-n-play" productivity; however, this decrease in productivity can be mitigated to some extent by access to information and experts within online communities such as Monome and Arduino. In contrast, fixed functionality provides immediate productivity but often prevents the interface from communicating in exactly the way the user desires, thereby imposing a particular interaction between the musician and the sound generation. Customization and immediate usability can be thought of as extremities of a spectrum (See Figure 33). At one end can be placed sensors, microcontrollers and software development, while the opposite end holds volume controls, panning knobs, filter knobs or any input or output device permanently assigned to only a single task. Interfaces such as the Monome effectively sit over a very large area of this spectrum, allowing for both complete hardware customization and immediate use. This broad usage is due to several factors stemming from an online community-based design approach, including open-source hardware/software, and a strong community involvement in the device's application development.

While interfaces enable the physical interaction between musician and computer, the systems to which those interfaces are mapped define the complexity of sound generation. Basic mappings between an actuator and a single sound provides note level interaction, however more complex mappings are possible through the use of interactive musical systems. These systems take single actions from performers, and use this information to contextualize the output of a virtual performer. These virtual performers are trained up during rehearsals, and emulate the relationship between the human musician and other members of an ensemble. In doing so, the system allows for a computer musician to simultaneously perform in the note level context, while influencing the output of other virtual performers on a score level context, in essence influencing an entire ensemble as a conductor. This ability for a computer musician to have direct focus on a one aspect of a musical performance, while simultaneously

influencing semi-autonomous systems, contributes further to the evolution of the performer composer.

While the development and use of interactive musical systems may help to distribute the performer's will, allowing a musician to simultaneously perform in multiple interaction contexts, there are potential challenges. To start, the complexity of the interaction may become difficult to manage during a performance, with the musician becoming unsure of a consistent response from their actions. However, this can be mitigated in part by adding score level controls into a system to ensure that large compositional events are synchronized. Even with this sort of functionality implemented into the system, a performer must still be careful to insure that the increased diffusion of a musician's actions do not lead to an unwanted increase in the audience's perception of a disembodied performance. Combining these systems with networked computer music ensembles provides an opportunity to balance the perceived embodied and disembodied aspects of live computer music.

Networked computer music ensembles enable new social modes of performance, as well as opportunities for building on top of the ways in which traditional ensembles have performed. Computer musicians in these ensembles are capable of utilizing the same interaction contexts as solo computer musicians, but must design their systems/instruments with different constraints in mind. The solo computer musician seeks to expand the expressive potential of their actions by automating musical parts, developing complex mapping schemes between controls and sound, and implementing interactive systems like those described in Chapter 3. Networked computer music however, in part alleviates the need for this type of broad control by sharing the parts of a composition among many different talented computer musicians. As previously mentioned, this decreases the number of simultaneous parts that each musician is responsible for, and allows them to focus more on the parts they do have. This increased focus potentially leads to more detailed improvisation, and allows for social interaction between performers or using shared social instruments such as musical robotics. This social interaction between musicians provides an embodied component to

the performance, creating an avenue for audiences to connect with the actions of the ensemble.

However, musicians in these types of ensembles have different performance considerations then solo computer musicians. Issues such as over-playing become a serious concern. For solo computer musicians who are used to attempting to play all the parts by themselves, stepping back and playing less may be a challenge at first, but also presents new opportunities for musical dialogues with other computer musicians. These musical dialogues may happen in a similar manner to acoustic ensembles, through an exchange of note-level musical ideas, or they may happen through new social instruments such as the network itself, or musical robotics. This concept of shared social instruments is unique to networked music, and provides yet another example of the evolution of the performer composer brought on by live computer music.

Together, these ideas provide the computer musician with new tools to share their musical expression using real, robotic, and synthetic ensembles. The emergence of online communities has created a space to share, modify, and develop new interfaces for musical expression. Computer musicians map these interfaces in individual ways, allowing a single device to control note-level, effects-level, and score-level interaction contexts. This control can be further extended by the creation of interactive musical systems and autonomous agents. These systems allow a single physical action from the musician to control multiple independent lines of musical performance. Lastly, these tools can be integrated in networked musical ensembles where this performance data can be shared between performers, creating social instruments. Such shared social instruments can even take the form of musical robotics, providing a physical instrument that is played by the entire ensemble. These tools extend the ability of a single performer composer beyond the limitations of physical agency, and instead allow for the musical intentions of a performer to be realized. This distribution of musical will into a system represents the evolution of the performer composer.

5.1.2 IMPROVISATION IN LIVE COMPUTER MUSIC

This thesis has made the case that live computer music represents an evolution of the performer composer, and has also presented my research into developing tools to further enable this evolution. How then are these tools to be used in an improvisational context? This section will present a general description of several approaches to improvisation, and how these use the different interaction contexts, and differ from acoustic music. These approaches are often combined into hybrid versions or variations during performances, and so do not represent an exhaustive taxonomy.

One approach is to start from nothing, and then create layered loops by performing all the notes or sounds on each layer. This has the benefit of allowing the audience to correlate the creation of each layer to specific sounds within the composition. Additionally, the performer is usually only modifying or playing a single sound per layer, making the connection between physical action and sound more obvious to the audience. Improvising in this way constitutes using the note-level context to play or perform the layers, possibly using the sound processing-level context if the layers are affected, and lastly using the score-level context to keep all the loops going, muting them or creating new ones. While this approach does satisfy the audience's desire for an embodied performance, it can also lead to compositionally limiting situations. By requiring all the music to be made one layer at a time, dramatic vertical shifts in composition, where multiple layers simultaneously change, can be difficult to achieve. Muting of layers is certainly possible, but dramatic shifts in harmonic content can be difficult without first building those layers as well.

A second approach is the use of musical material that is prepared prior to the performance. This material can be manipulated, effected, and re-arranged in real-time during a performance, and can be comprised of audio loops or MIDI loops. This approach allows for score-level control of the music as a performer moves from one loop to another, sound processing-level control if the audio is processed or effected, and not-level control if the musical material re-arranged

enough as to be perceived as completely new musical ideas. An example of this might be the chopping and re-arranging of a melodic line, where the result is an entirely new melody. This essentially uses the loops as an instrument, with slices of the sound equating to the notes. While his approach has the benefit of allowing major shifts in the composition, as the previously prepared ideas make moving from one piece of musical material to another feasible, the use of prepared material means that the audience does not see the musician play every note. This can potentially dilute the audience's sense of embodiment, and make understanding and connecting to the performance more difficult.

Finally, a third approach is to improvise with the computer as a system or circuit. These systems are semi-autonomous, and create the opportunity, in varying degrees, for a musical dialogue between the human musician and the machine. This approach may afford score-level control by allowing the musician explicit control over the system or algorithmic process, essentially enabling them to force the system into a different state. Sound processing-level control of the audio is possible through processing or affecting either the output of the system, or the input to the system. Lastly, the note-level context is dependent on the design of the system. Strictly inputting to a semi-autonomous system or algorithm amounts to influence, and as such not explicit control; however, systems such as the interactive musical agents described in this thesis may be only listening to the notes being played into another instrument, and then reacting to that information. This approach to improvising during performance can prove to be the most difficult for audiences to follow as an action from the human musician may cause many different reactions from the system. Furthermore, the level of influence imparted on the system may not be easy to discern.

The three approaches described here all leverage the computers ability to automate tasks in the background while a musician's focus is on something else. Whether it is looping layers of audio, providing random access to prerecorded material, or running a complex algorithmic system, computers enable the musician to extend their control over a live composition.

5.2 CONTRIBUTIONS

The following section provides a summary of the work and presents the main contributions of the thesis. These contributions are as follows:

1. The description of online community driven iterative interface design, and several new interfaces resulting from this process. This approach to interface design is shown to help drive innovation, and create new tools for the computer musician.

2. The description of a search algorithm for modeling multiple streams of continuous control data from two performers. The algorithm was implemented as a plugin for use inside modern digital audio workstations.

3. The description of a shared social robotic instrument.

5.2.1 ONLINE COMMUNITY BASED ITERATIVE DESIGN AND THE CHRONOME

The diffusion of action afford by live computer music is initially enabled through the use of a physical interface. As the mappings between physical actions and sound production can be unique to each performer, developing interfaces without any predefined relationship between physical actuator and sound generator is crucial to allowing live computer musicians to create custom complex performance systems. Information regarding these interfaces has previously been shared at academic or research institutes and communities; however, the advent of online communities has also allowed the broader public to participate in the development of these devices. A small number of community members modify and add new functionality to existing interfaces, and these modifications then become integrated into the use of the larger community over time. Once integrated, the ability to discuss and share information about the way in which these interfaces are being used helps to drive innovation and spur new ideas.

This thesis also presented the Arduinome and the Chronome as examples of interfaces developed by the author through interaction with the online Monome community. The Arduinome was a collaborative project that ported the Monome to the Arduino platform, increasing the potential for modifications and development. The Chronome built off this work, adding RGB LEDs and pressure sensitivity to the original design. All the technical information regarding these two interfaces has been shared with the online community, and has led to modifications and re-imaginings of the musical uses of these interfaces.

5.2.2 S2MP AND AN INTERACTIVE SYSTEM FOR CONTINUOUS CONTROL

This thesis presented a novel approach to creating an interactive system for modeling continuous control data, based off work using a search algorithm called S2MP (Martin et al. 2011). The algorithm uses a weighting between similarity of events, and similarity of sequence, and allows the system to match against new sequences not seen in the training database.

This system was implemented as a plugin, and was shown to be able to train through rehearsals, and reproduce continuous control data in the style of one performer given an input stream from another. The thesis also presented an approach for simultaneously sampling multiple sources of continuous control data from two different performers. This approach collapses a total of 128 controller inputs into a single 1-dimensional vector, and only stores events as they change, allowing for a sparse collection.

This system represents a step towards creating virtual versions of a single performer, with the goal being to eventually allow computer musicians to control and influence autonomous computer generated ensembles.

5.2.3 SHARED SOCIAL MUSICAL ROBOTICS

This thesis also presented the concept of a shared social robotic instrument. These instruments are unique in that a network of computer musicians are virtually linked through a shared physical instrument. The shared use of a physical instrument places constraints on what sounds can be simultaneously played by the ensemble, provides a physical location for group musical interaction, and creates a point of physical action that an audience may associate with increased embodied performance. Most interestingly, the shared social robotic instruments provide a new space in which to explore group musical performance and improvisation.

5.3 FUTURE WORK AND PHILOSOPHY

This thesis has presented tools and ideas that have lead to an evolution of the performer composer. These tools are now being used to create new forms of music, new interactions between performers and their instruments, and new relationships between musicians and the audience. Future work will focus on taking these ideas, as well as the new modes of interaction afforded by them, and working towards integrating it all into a new live performance aesthetic. One that is based not just on physical interaction, but also on the distribution of the musician's intent. Future areas of research will:

- Explore ways in which the knowledge and work shared in online communities can be integrated or leveraged in teaching interface design to students.

- Look at leveraging machine learning systems such as the Wekinator (Fiebrink 2011) to rapidly prototype interactive systems and qualitatively compare different approaches for use in performance, and examine the relationship between performer, agent, and audience.

- Explore the potential to develop robotic instruments that are designed to facilitate shared control, possibly creating mechanisms with behaviors that only manifest through shared control.

5.3.1 BRIDGING THE GAP BETWEEN PERFORMER AND AUDIENCE

Whether performing as a solo computer musician, or within a networked ensemble, the relationship between the musician and the audience is complex. A great strength of live computer music is the ability for each performer to create unique and individual mappings between their physical interfaces, and the systems that generate the sound; however, this individualistic approach to interacting with the computer as an instrument can present a challenge for audiences. That challenge is one of understanding the interaction and intents of the performer. What sounds are the results of the musician's actions? How much of the music is generated during the performance? How much of the music is prepared before hand? What is the level of skill involved to perform the music? What is an audience to expect from live computer music? These questions are all central to understanding live computer music and the musical space in which it is growing and maturing.

I have found, in my own performances that a small dilution of an embodied note-level connection, can lead to confusion from certain members of the audience, while other audience members are more comfortable with the plurality of roles a live computer musician plays. So how then are computer musicians to navigate these interaction contexts while performing live? John Croft writes:

> "It is a question of the specificity of the relation: if many perceptibly different inputs generate outputs with no pertinent differences (in other words, if the aesthetically pertinent mapping is many to one), then the liveness is merely procedural and not aesthetic – pre-recorded sounds would do the job as well or better. At the other extreme, if the mapping is too explicit, too transparently one-to-one, the result is not only tedious but may have the effect of shifting the procedural into the foreground, turning the

> piece into a lamentable 'showcase' of the technology. ('Look – I do this, and the computer does that!')" (Croft 2007)

The three interaction contexts described in this thesis span the two extremes described by Croft: the many-to-one, and the one-to-one. The evolution of the performer composer represents the ability to perform across this continuum, using the performer's physical actions to disseminate their musical intent. This diffusion of physical action can be difficult for audiences to follow, and can lead to confusion or the audience feeling as though the performance is disembodied. It has been suggested that an audience's desire for an embodied performance may be a form of nostalgia (Croft 2007; d' Escriván 2006), and that as audiences' stereotypes and performance expectations mature and grow, live computer music may see less of this type of criticism.

This process of maturation is similar to the ideas of decentering, technological mediation, and recentering presented by Kockelkoren (Kockelkoren 2003). Live computer music's technology has an initial decentering effect as it alters the roles traditionally possible in live music. Technological mediation is where live computer music currently is, and is described by Kockelkoren as, "the cultural process in which technology extends our ability to perceive, redistributes social relations, and thereby elicits new visual language and conferral of meanings." Once audiences undergo this process they will reach a recentering, and a new understanding of how a performer composer is able to play live music. This process of technological mediation is not solely the responsibility of the audience alone. The musicians must also develop tools and methods for bridging the gap between performers and the audience, and in doing so find a space in which live computer music may flourish.

So can the actions of live computer music be understood by using the same values and aesthetics as live acoustic music? To be sure, the two share many similarities: they both strive to present interesting musical ideas to an audience; they both strive to allow a musician to express their musical ideas through real time interaction with a sound producing instrument; they both create an interaction between the audience and the performers; as well as many other

commonalities between the two. The similarities presented here are very general and broad, but nonetheless illustrate that live computer music is certainly related to live acoustic music, and as such many of the criteria by which a live acoustic performance is judged will also apply to judging computer music. However, the subtle and important differences between live acoustic music and live computer music are great enough to warrant a separate, or at least derivative, set of aesthetics.

The computer musician is different from their acoustic counterpart in that they can turn their physical action into a multitude of simultaneous actions that carry the performers intent rather then merely the physical energy. This intent can be used to control instruments that are themselves systems, producing emergent behavior and semi-autonomous music. These types of distributed interactions often happen inside the computers, and are not easily perceivable by the audience. Without the perceived physical agency of the human musician, audiences may lose faith in the authenticity of the performance.

The use of the word faith is important, because the question of critically understanding live computer music can be framed as a question of faith. In a live acoustic music performance, an audience member may not know how to play any of the instruments they are seeing on stage, but they assume that what they see and hear is authentic, i.e., they have faith that the performance is actually happening as they perceive it. Conversely, the faith that audiences have in the authenticity of performing musicians is challenged when an artist is caught lip-syncing or faking a performance. Until the moment the backing track skips, or the wrong music plays, the audience believes the performance to be real. This illustrates that although the audience believes that the physical actions they see are crucial parts of the validation of the musical performance, they take much more of that connection on faith then would at first seem obvious.

In light of this unspoken faithful pact between audience and musician, it can now be understood why applying the aesthetics of live acoustic music to live computer music will not work. The faith that an audience has in an acoustic

performance is predicated on the physical actions of the musicians creating clear relationships with the sounds being heard. The evolution of the performer composer afforded by live computer music enables the diffusion of physical action into many separate and not necessarily related musical events. This then would seem to work against the reinforcement of the current faith that an audience accustomed to acoustic music would have.

One solution is to attempt to address this issue by embodying the diffusion of the musician's intent. In essence manifesting the multiple virtual actions into physical forms that are understandable and relatable to an audience. This is good start, and does help to bridge the computer musician and the audience, but it also places the responsibility on the performer, and if taken too far may subvert musical and artistic intents in order to provide a technological demonstration of the link between action and sound. I argue that in addition to this embodied approach that a new audience will emerge, willing to take a leap of faith and embrace both the musician's physical actions and intent. An audience that understands this marks the coming of age of a new musicianship, comprised of both performers and audience members that are growing increasingly familiar with the agency of computer music.

5.3.2 FINAL THOUGHTS

This thesis has argued that live computer music represents an evolution of the performer composer; empowered by the ability of the computer to distribute the musical will of the performer into a system. This distribution of intent extends the existing mapping between physical action and sound generation, and allows a single performer to simultaneously control multiple sonic events, across several different instruments.

This shift in mappings, from the physical to the virtual, parallels many of the current shifts happening in our world today. Just as the interaction with a physical instrument can now be connected to a network of virtual instruments, so does our interaction with people shift from the physical world into a distributed network of virtual social connections. Social networking, global video

chats, and even forms of entertainment such as online role-playing games like World of Warcraft [26] or second life, [27] all point to a disembodied set of interactions. We may very well be on the verge of a shift in our perception of what constitutes reality, with an acceptance of purely abstracted interactions. If this is true, then live computer music can be seen as part of this abstracted connection to reality.

It is my hope that this evolution is seen as an expansion of what is possible for musicians, and not as a division between them. This chapter begins with a quote that describes the process of creating values in order to define our identities, and in a way, that is what this thesis has done. Live computer music has a different set of values from acoustic music, and therefore is a different and separate entity. Ultimately, in order to understand live computer music, a new set of values must be used. However, the irony in Nietzsche's quote should not be lost. That irony is that the very values we need to define us are merely fabrications created by us. The values presented in this thesis define and create a shape and description of what live computer music can be, but those same values can soon become walls that create an "other" with different musics, and at their worst serve to exclude new ideas. In light of this, it is my hope that the evolution of the performer composer, mediated through the use of the computer, is seen as an evolution in live music that expands what is possible to all performer composers. With this in mind, I am excited to hear and see what interactions the musicians of tomorrow will explore.

[26] World of Warcraft - http://us.battle.net/wow/en/

[27] Second Life - http://secondlife.com/

Appendix A

RELATED PUBLICATIONS

Portions of the work completed over the course of this thesis is contained in the following publications.

1 JOURNAL PUBLICATIONS

Vallis, O., Diakopoulos, D., Hochenbaum, J., Kapur, A. 2012 "Building on the Foundations of Network Music: Exploring Interaction Contexts and Shared Robotic Instruments."
Organised Sound, 17(1).

Vallis, O., Kapur, A. 2011 "Community-Based Design: The Democratization of Musical Interface Construction." *Leonardo Music Journal*, 21.

Kapur, A., Darling, M., Diakopoulos, D., Murphy, J., Hochenbaum, J., Vallis, O., Bahn, C. 2011 "The Machine Orchestra: An Ensemble of Human Laptop Performers and Robotic Musical Instruments." *Computer Music Journal*, 35(4).

2 INTERNATIONAL REFEREED CONFERENCES

Vallis, O.,Hochenbaum, J., Murphy, J., Kapur, A. 2011 "The Chronome: A Case Study in Designing New Continuously Expressive Musical Instruments."
Proceedings of the Australasian Computer Music Conference (ACMC). Auckland, New Zealand.

Vallis, O., Hochenbaum, J., Kapur, A. 2010 "A Shift Towards Iterative and Open-Source Design For Musical Interfaces." *Proceedings of the 2010 International Conference on New Interfaces for Musical Expression.* Sydney, Australia.

Kapur, A., Darling, M., Wiley, M., Vallis, O., Hochenbaum, et al., 2010 "The Machine Orchestra." *Proceedings of the International Computer Music Conference.* New York City, New York.

Hochenbaum, J., Vallis, O., Diakopoulos, D., Murphy, J., Kapur, A. 2010 "Designing Expressive Musical Interfaces For Tabletop Surfaces." *Proceedings of the 2010 International Conference on New Interfaces for Musical Expression.* Sydney, Australia.

Diakopoulos, D., Vallis O., Hochenbaum J., Murphy, J., Kapur, A. 2009 "21st Century Electronica: MIR Techniques for Classification and Performance" *Proceedings of the 2009 International Society on Music Information Retrieval Conference.* Kobe, Japan.

Hochenbaum, J., Vallis, O. 2009 "Bricktable: A Musical Tangible Multi-Touch Interface"
Proceedings of Berlin Open Conference '09. Berlin, Germany.

Hochenbaum, J., Vallis, O., Akten, M., Diakopoulos, D., Kapur, A. 2009 "Musical Applications for Multi-Touch Surfaces" *1st Workshop on Media Arts, Science, and Technology*. Santa Barbara, USA.

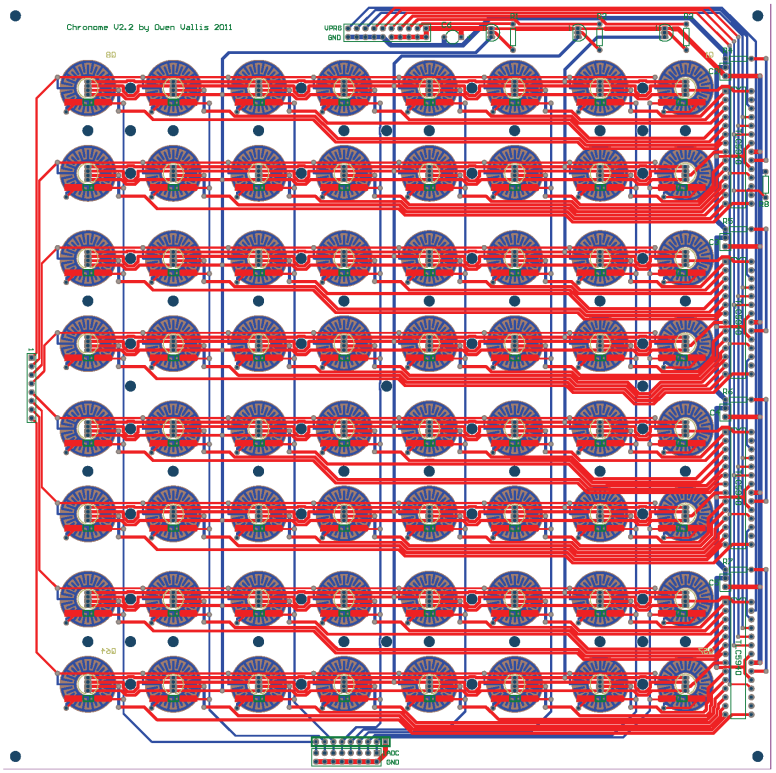
Appendix B

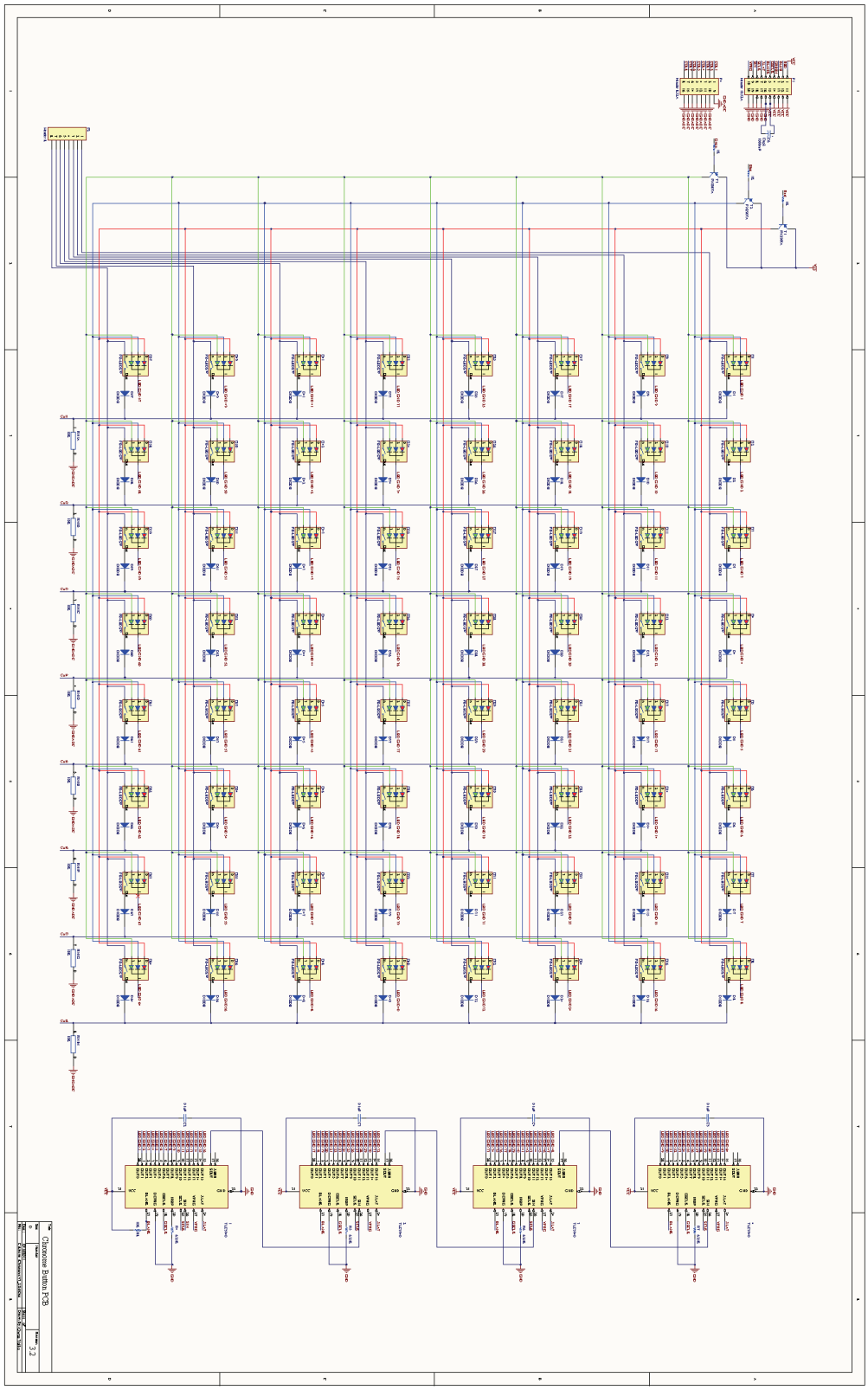
CHRONOME TECHNICAL FILES

The following section contains the technical files shared online regarding building a Chronome. More detailed information can be found online at:

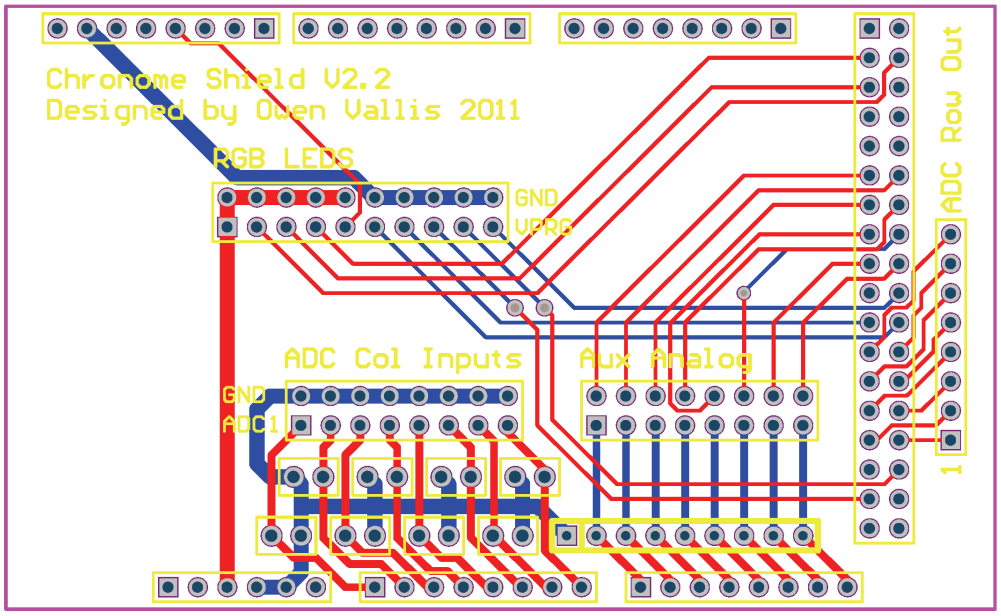
http://flipmu.com/work/chronome

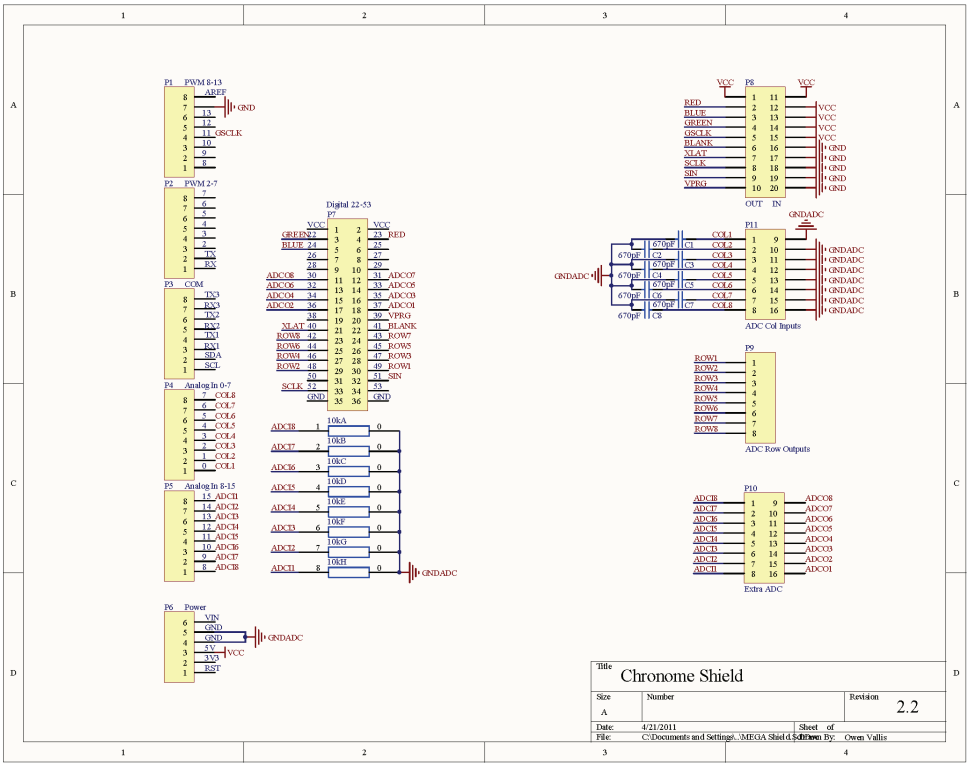
1 MAIN PCB AND SCHEMATIC





2 Arduino Mega shield PCB and schematic





3 SERIAL PROTOCOL

```
chronome serial protocol
owen vallis - contact@flipmu.com
//based off of the monome serial protocol series 256/128/64
//by brian crabtree

revision: 004

from device:

message id:   (1) pressure
bytes:        3
format:              iiii.xxx .yyy..dd dddddddd
                     i (message id) = 1
                     x (x value) = 0-7 (three bits)
                     y (y value) = 0-7 (three bits)
                     d (data value) =  0 - 1024 (ten bits)

decode:              id match: byte 0 & 0xf0 == 16
                     x: byte 0 & 0x0f
                     y: byte 1 >> 4
                     d: uint16_t val = ((byte 1 & 0x0f) << 8) | byte 2


to device:

message id:   (1) rgb_led_on
bytes:        2
format:              1...iiii 0xxx0yyy
                     i (message id) = 1
                     x (x value) = 0-7 (three bits)
                     y (y value) = 0-7 (three bits)
encode:              byte 0 = id | 0x80 = 129
                     byte 1 = ((x << 4) | y) & 0x7f


message id:   (2) rgb_led_off
bytes:        2
format:              1...iiii xxxxyyyy
                     i (message id) = 2
                     x (x value) = 0-7 (three bits)
                     y (y value) = 0-7 (three bits)
encode:              byte 0 = id | 0x80  = 130
                     byte 1 = ((x << 4) | y) & 0x7f


message id:   (3) rgb_led_color
bytes:        5
format:              1...iiii 0xxx0yyy 0rrrrrrr 0ggggggg 0bbbbbbb
                     i (message id) = 3
                     x (x value) = 0-7 (three bits)
                     y (y value) = 0-7 (three bits)
                     r (red value) = 0 - 127 (7 bits)
                     g (green value) = 0 - 127 (7 bits)
                     b (blue value) = 0 - 127 (7 bits)
encode:              byte 0 = id | 0x80  = 131
                     byte 1 = ((x << 4) | y) & 0x7f

                     byte 2 = (r & 0x7f)
                     byte 3 = (g & 0x7f)
                     byte 4 = (b & 0x7f)
```

```
message id:   (4) rgb_led_all_state
bytes:        1
format:             1..siiii
                    i (message id) = 4
                    s (test state) = 0-1
encode:             byte 0 = id | 0x80 | (s << 4) = 132 | (s << 4)


message id:   (5) rgb_row
bytes:        2
format:             1yyyiiii aaaaaaaa
                    i (message id) = 5
                    y (row to update) = 0-7 (three bits)
                    a (row data 0-7) = 0-255 (eight bits)
encode:             byte 0 = id | 0x80 | (y << 4) = 133 | (y << 4)
                    byte 1 = a (row data 0-7)


message id:   (6) rgb_col
bytes:        2
format:             1xxxiiii aaaaaaaa
                    i (message id) = 6
                    x (col to update) = 0-7 (three bits)

                    a (row data 0-7) = 0-255 (eight bits)
encode:             byte 0 = id | 0x80 | (x << 4) = 134 | (x << 4)
                    byte 1 = a (row data 0-7)
```

4 FIRMWARE FOR THE ARDUINO MEGA

```
/*
 * "ChronomeFirmware" - Arduino Based RGB Pressure Sensitive
 * Monome Clone by Owen Vallis 09/23/2010
 *
 * ------------------------------------------------------------
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be
 * useful, but WITHOUT ANY WARRANTY; without even the implied
 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
 * PURPOSE.  See the GNU General Public License for more details.
 * ------------------------------------------------------------
 *
 * Parts of this code is based on Matthew T. Pandina's excellent
 * TLC5940 C Library, with pins updated to work with the Arduino
 * MEGA. For those portions, he asked that his copyright be added
 * to the code.
 *
 * Copyright 2010 Matthew T. Pandina. All rights reserved.
 * Redistribution and use in source and binary forms, with or
 * without modification, are permitted provided that the
 * following conditions are met:
 *
```

```
 * 1. Redistributions of source code must retain the above
 *    copyright notice, this list of conditions and the following
 * disclaimer.
 * 2. Redistributions in binary form must reproduce the above
 * copyright notice, this list of conditions and the following
 * disclaimer in the documentation and/or other materials
 * provided with the distribution.
 *
 * Thanks to Brad Hill, Martijn Zwartjes, Jordan Hochenbaum,
 * Johnny McClymont, Tim Exley, and Jason Edwards for answering
 * my questions along the way.
 *
 * Please DO NOT email monome with technical questions and/or
 * help regarding this code or clone. They are in NO WAY
 * responsible or affiliated with this project other than they
 * were our inspiration and we used many of their methods and
 * pulled from their code.
 *
 * Additionally, while we are availble and willing to help as
 * much as possible, we too CANNOT be held responsible for
 * anything you do with this code.  Please feel free to report
 * any bugs, suggestions or improvements to us as they are all
 * welcome.  Again, we cannot be held responsible for any damages
 * or harm caused by the use or misuse of this code or our
 * instructions.  Thank you for understanding.
 *
 * Links:
 * http://www.flipmu.com - Our website - Click "Chronome Project"
 * on the Navigation Menu under Work.
 * www.monome.org - the "original" monome and our inspiration
 */

// supports uint8_t and uint16_t
#include <stdint.h>
// Definition of interrupt names
#include <avr/interrupt.h>
// ISR interrupt service routine
#include <avr/io.h>

//******************** TLC5940 pin definitions ******************
// MEGA PWM PIN 11
#define GSCLK 11
#define GSCLK_DDR DDRB
#define GSCLK_PORT PORTB
#define GSCLK_PIN PB5
// MEGA MOSI PIN 51
#define SIN 51
#define SIN_DDR DDRB
#define SIN_PORT PORTB
#define SIN_PIN PB2
// MEGA SCK PIN 52
#define SCLK 52
#define SCLK_DDR DDRB
#define SCLK_PORT PORTB
#define SCLK_PIN PB1
// MEGA PIN 41
#define BLANK 41
#define BLANK_DDR DDRG
#define BLANK_PORT PORTG
#define BLANK_PIN PG0
// MEGA PIN 40
#define XLAT 40
#define XLAT_DDR DDRG
#define XLAT_PORT PORTG
```

```
#define XLAT_PIN PC1
// MEGA PIN 39
#define VPRG 39
#define VPRG_DDR DDRG
#define VPRG_PORT PORTG
#define VPRG_PIN PG2
// MEGA PIN 22
#define REDTR 22
// MEGA PIN 23
#define GREENTR 23
// MEGA PIN 24
#define BLUETR 24

// MEGA PINS 49-42 ROWS are on PORTL
#define ROWS PORTL

// Additional SPI PIN defs (Not used but set)
// MEGA MISO PIN 50
#define DATAIN 50
// MEGA SS PIN 53
#define SLAVESELECT 53

//******************** Variables ******************************
//********************** Macros  ******************************
#define TLC5940_N 4
#define numColors (uint8_t)3

#define setLow(port, pin) ((port) &= ~(1 << (pin)))
#define setHigh(port, pin) ((port) |= (1 << (pin)))

#if (16 * TLC5940_N > 255)
#define channel_t uint16_t
#else
#define channel_t uint8_t
#endif
#define numChannels ((channel_t)16 * TLC5940_N)

#if (24 * TLC5940_N > 255)
#define gsData_t uint16_t
#else
#define gsData_t uint8_t
#endif

#define gsDataSize ((gsData_t)24 * TLC5940_N)
#define numChannels ((channel_t)16 * TLC5940_N)


uint8_t gsData[numColors][gsDataSize];
uint8_t gsStateData[numColors][gsDataSize];
uint16_t previousButtonValue[8][8];

boolean led13;

//******************* Serail Functions setup ******************
uint8_t tolerance = 7;

void sendSerial(uint8_t Data)
{
  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = Data;
}
```

```
//************** Serail Functions From the Octinct **************
//Debugging definitions: uncomment the line to turn it on
//Draw colour is forced to red if the serial receive buffer has
//more than the specified number of characters in it
#define REDALERT 100

/* Size of the serial buffer before the chronome is forced to parse
it continually. The buffer size is 128 bytes, and if it gets there
the chronome can (and will) crash. The largest command size is 9
bytes, so 119 is an absolute maximum value. Set it lower than this to
be safe.
If the chronome hits this limit, it will start to flicker, and might
miss commands, but it won't crash… Probably.
*/
#define TOOFULL 100

//Variables for interpreting the serial commands
uint8_t address, state, x, y, pos;
uint16_t r, g, b;
uint8_t ready = true;

//For interrupt timing; only to do intermediate clock speeds
/* Divide interrupt frequency by a factor of FREQ. It is preferable
to keep FREQ as small as possible, and control the frequency of the
interrupts using the hardware clock. Setting it to 1 disables this
entirely, which, if it works, is ideal; this should be the same as
commenting out the "#define FREQ" statement entirely.
*/

//How many interrupts occur before the serial commands are read
#define FREQ 1
#if FREQ > 1
  byte int_counter = 0;
#endif

//The timer interrupt routine, which periodically interprets the
//serial commands
ISR(TIMER2_OVF_vect) {
  //Reenable global interrupts, otherwise serial commands will
  //get dropped
  sei();

#if FREQ > 1
  if(++int_counter == FREQ){
//Only do this once every FREQ-th interrupt
    int_counter = 0;
#endif //FREQ
    do{
//This do ensures that the data is always parsed at least once  //per
cycle
      if(Serial.available()){
#ifdef REDALERT
//if REDALERT is defined, draw colour turns red when the buffer
//is getting dangerously full
        if(Serial.available() > REDALERT){
          for(int x = 0; x < 64; x++)
          {
            TLC5940_SetGS(x, 4095, 0);
            TLC5940_SetGS(x, 0, 1);
            TLC5940_SetGS(x, 0, 2);
          }
        }
#endif //REDALERT
```

```
        if(ready){
//If the last command has finished executing, read in the next
//command and reset the command flag
          address = Serial.read();
          ready = false;
        }

//if the MSB doesn't equal 1, then we are missing our address
//message. Trash byte and read again.
        if((address & 0x80) != 0x80){
          ready=true;
          break;
        }

        switch (address & 0xf) {
//Execute the appropriate command, but only if we have received
//enough bytes to complete it. We might one day add "partial
//completion" for long command strings.
        case 2: //rgb_led_on
          if( Serial.available()) {
            int byte1 = Serial.read();
            x = byte1 >> 4;
            y = byte1 & 0xf;
            pos = (x)+(y*8);

            TLC5940_SetGSState(pos, true);
            ready=true;
          }
          break;
        case 3: // rgb_led_off
          if( Serial.available()) {
            int byte1 = Serial.read();
            x = byte1 >> 4;
            y = byte1 & 0xf;
            pos = (x)+(y*8);

            TLC5940_SetGSState(pos, false);
            ready=true;
          }
          break;
         case 4: // rgb_led_color
          if( Serial.available() > 3 ) {
            uint8_t pos = Serial.read();
            x = (pos >> 4);
            y = (pos & 0x0F);
            pos = (x)+(y*8);
            r = (uint16_t)(Serial.read() * 32);
            if(r > y * 35) {
              r = r - (y * 35);
            }
            g = (uint16_t)(Serial.read() * 32);
            if(g > y * 35) {
              g = g - (y * 35);
            }
            b = (uint16_t)(Serial.read() * 32);
            if(b > y * 35) {
              b = b - (y * 35);
            }
            TLC5940_SetGS(pos, r, 0);
            TLC5940_SetGS(pos, g, 1);
            TLC5940_SetGS(pos, b, 2);
            ready=true;
          }
          break;
```

```
        case 5: //rgb_led_all_on
          {
            boolean state = (address >> 4) & 0x01;
            for (int pos = 0; pos < 64; pos++) {
                TLC5940_SetGSState(pos, state);
            }
            ready=true;
          }
          break;
        case 6: //rgb_led_row
          {
            if( Serial.available()) {
              uint8_t ledRow = (address >> 4) & 0x07;
              uint8_t rowState = Serial.read();

              for (uint8_t col = 0; col < 8; col++) {
                  uint8_t state = (rowState >> col) & 0x01;
                  TLC5940_SetGSState((ledRow * 8) + col, state);
              }
              ready=true;
            }
          }
          break;
        case 7: //rgb_led_col
          {
            if( Serial.available()) {
              uint8_t ledCol = (address >> 4) & 0x07;
              uint8_t colState = Serial.read();

              for (uint8_t row = 0; row < 8; row++) {
                  uint8_t state = (colState >> row) & 0x01;
                  TLC5940_SetGSState(ledCol + (row * 8), state);
              }
              ready=true;
            }
          }
          break;
         default:
           break;
        }
      }
    }

//If the serial buffer is getting too close to full, keep
//executing the parsing until it falls below a given level
//This might cause flicker, or even dropped messages, but it
//should prevent a crash.
    while (Serial.available() > TOOFULL);
#if FREQ > 1
  }
#endif //FREQ
}
```

```
//********************** RGB Function ************************
//set all GrayScale Color
void TLC5940_SetAllGS(uint16_t value) {
  uint8_t tmp1 = (value >> 4);
  uint8_t tmp2 = (uint8_t)(value << 4) | (tmp1 >> 4);
  for (uint8_t i = 0; i < numColors; i++){
    gsData_t j = 0;
    do {
      gsData[i][j++] = tmp1; // bits: 11 10 09 08 07 06 05 04
      gsData[i][j++] = tmp2; // bits: 03 02 01 00 11 10 09 08
      gsData[i][j++] = (uint8_t)value; // bits: 07 06 05 04 03 02
                                              //01 00
    }
    while (j < gsDataSize);
  }
}

//set a single GrayScale Color
void TLC5940_SetGS(channel_t channel, uint16_t value, uint8_t color)
{
  channel = numChannels - 1 - channel;
  uint16_t i = (uint16_t)channel * 3 / 2;
  switch (channel % 2) {
  case 0:
    gsData[color][i++] = (value >> 4);
    gsData[color][i++] = (gsData[color][i] & 0x0F) |
                         (uint8_t)(value << 4);
    break;
  default: // case 1:
    gsData[color][i++] = (gsData[color][i] & 0xF0) |
                         (value >> 8);
    gsData[color][i++] = (uint8_t)value;
    break;
  }
}

//turn on or off an LED
void TLC5940_SetGSState(channel_t channel, boolean state) {
 channel = numChannels - 1 - channel;
 for (uint8_t n = 0; n < numColors; n++){
  uint16_t i = (uint16_t)channel * 3 / 2;
  switch (channel % 2) {
  case 0:
    gsStateData[n][i++] = gsData[n][i] * state;
    gsStateData[n][i++] = (gsStateData[n][i] & 0x0F) |
                          ((gsData[n][i] & 0xF0) * state);
    break;
  default: // case 1:
    gsStateData[n][i++] = (gsStateData[n][i] & 0xF0) |
                          ((gsData[n][i] & 0x0F) * state);
    gsStateData[n][i++] = gsData[n][i] * state;
    break;
  }
 }
}
```

```
//ISR for clocking in the next Color's GSData.
ISR(TIMER3_COMPA_vect) {
  static uint8_t color = 0;

  PORTA = 0x07;

  setHigh(BLANK_PORT, BLANK_PIN);
  setHigh(XLAT_PORT, XLAT_PIN);
  setLow(XLAT_PORT, XLAT_PIN);
  setLow(BLANK_PORT, BLANK_PIN);

  PORTA &= ~(1 << color);

//Below this we have 4096 cycles to shift in the data for the
//next cycle
  for (gsData_t i = 0; i < gsDataSize; i++) {
    SPDR = gsStateData[color][i];
    while (!(SPSR & (1 << SPIF)));
  }

  color = (color + 1) % numColors;
}

//******************** Button Functions ************************
void readADC() {
  for( uint8_t row = 0; row < 8; row++){
    // incrment and set row high
    ROWS = (1 << row);

    // let the board settle after we shift a row
    delayMicroseconds(100);

    //check each column's value
    for( uint8_t col = 0; col < 8; col++)
    {
      uint16_t currentButtonValue = analogRead(col);

      //if we have changed then send it out
      if(abs(previousButtonValue[row][col] - currentButtonValue)
         > tolerance || (previousButtonValue[row][col] != 0
         && currentButtonValue == 0))
      {
        //This is to avoid the noise near zero
        if(currentButtonValue > 10 || currentButtonValue == 0) {
           sendSerial(0x10 | ((col) & 0x0F));
           sendSerial((row << 4) | (uint8_t)(currentButtonValue
                                             >> 8));
           sendSerial((uint8_t)currentButtonValue);
        }
      }

      //store current value
      previousButtonValue[row][col] = currentButtonValue;

      delayMicroseconds(10);
    }
  }
}
```

```
//************************ Arduino Loops *************************
//Setup Device
void setup(){
  Serial.begin(57600);


  //************** SETUP PINS **************
  pinMode(GSCLK, OUTPUT);
  pinMode(SCLK, OUTPUT);
  pinMode(VPRG, OUTPUT);
  pinMode(XLAT, OUTPUT);
  pinMode(BLANK, OUTPUT);
  pinMode(SIN, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SLAVESELECT,OUTPUT);
  pinMode(REDTR,OUTPUT);
  pinMode(GREENTR,OUTPUT);
  pinMode(BLUETR,OUTPUT);
  pinMode(13, OUTPUT);

  for( int i = 0; i < 8; i++){
    pinMode(42+i, OUTPUT);
  }

  digitalWrite(SLAVESELECT,HIGH); //disable device
  setLow(GSCLK_PORT, GSCLK_PIN);
  setLow(SCLK_PORT, SCLK_PIN);
  setHigh(VPRG_PORT, VPRG_PIN);
  setLow(XLAT_PORT, XLAT_PIN);

  //************** SET ADC **************
  ROWS = (1 << 0); //Set the first Chronome Row High

  //************** SET SPI **************
  //Enable SPI, Master, set clock rate fck/2
  SPCR = (1 << SPE) | (1 << MSTR);
  SPSR = (1 << SPI2X);

  //Clear SPI data Registers
  byte clr;
  clr=SPSR;
  clr=SPDR;

  //************** SET TIMERS **************
  //Dont need to call sei(); because Arduino already does this
  //Clear TIMER1 Reg back to default
  TCCR1A = 0x00;
  TCCR1B = 0x00;
  //Enable timer 1 Compare Output channel A in toggle mode
  TCCR1A |= (1 << COM1A0);
  //Configure timer 1 for CTC mode
  TCCR1B |= (1 << WGM12);
  //Set up timer to fCPU (no Prescale) = 16Mhz/8 = 2Mhz
  //Set CTC compare value to pulse PIN at 2Mhz
  //(1 / Target Frequency) / (1 / Timer Clock Frequency) - 1
  TCCR1B |= (1 << CS11);
  //Full period of PIN 11 pulse requires 2 ticks (HIGH, LOW)
  //So PIN 11 @ 2Mhz = (2 ticks (HIGH, LOW)) = 1Mhz
  OCR1A = 0;

  //Clear TIMER3 Reg back to default
  TCCR3A = 0x00;
  TCCR3B = 0x00;
  //Configure timer 3 for CTC mode
```

```
  TCCR3B |= (1 << WGM32);
  //Set up timer to fCPU (no prescale) = 16Mhz/8 = 2Mhz
  TCCR3B |= (1 << CS31);
  //Set CTC compare value to 4096 @ half TIMER1 frequency
  //So (4096*2) @ 2Mhz = 4096 @ 1Mhz
  OCR3A = (4096*2) - 1;
  //Enable Timer/Counter3 Compare Match A interrupt
  TIMSK3 |= (1 << OCIE3A);

  //Setup the timer interrupt for Serial
  TCCR2A = 0;
  TCCR2B = 0<<CS22 | 1<<CS21 | 1<<CS20;

  //Timer2 Overflow Interrupt Enable
  TIMSK2 = 1<<TOIE2;

  //************** SET FirstCycle **************
  //Default all channels to all white
  TLC5940_SetAllGS(4095);
  //Default all LED states to off
  for(int pos = 0; pos < 64; pos++) {
    /*uncomment for setting a default color other than white
      TLC5940_SetGS(pos, 2000, 0);
      TLC5940_SetGS(pos, 0, 1);
      TLC5940_SetGS(pos, 4095, 2);
      */
      TLC5940_SetGSState(pos, false);
  }

  PORTA = 0x07;

  setHigh(BLANK_PORT, BLANK_PIN);
  setLow(VPRG_PORT, VPRG_PIN);
  setHigh(XLAT_PORT, XLAT_PIN);
  setLow(XLAT_PORT, XLAT_PIN);
  setHigh(SCLK_PORT, SCLK_PIN);
  setLow(SCLK_PORT, SCLK_PIN);
  setLow(BLANK_PORT, BLANK_PIN);

  PORTA = 0x03;
}

//Run
void loop() {
  //read the buttons
  readADC();
}
```

Appendix C

COMPARATIVE SURVEY OF LOCAL NETWORK ENSEMBLES AND SOLO LIVE COMPUTER MUSIC

The following section presents a survey of the members of The Machine Orchestra, and their thoughts on performing as part of an ensemble versus performing as a solo computer musician. A public survey was made available to all the existing and previous members of The Machine Orchestra, with a total of 14 participants replying to the survey. The musicians were asked to rate their familiarity with both live computer music and networked music ensembles on a scale of 1-10, with 1 indicating that they have never heard of the topic and 10 indicating that they would consider themselves experts. The familiarity with performing live computer music was fairly high with an average response of 7.4, while the familiarity with performing networked music was a slightly lower with an average of 5.4 out of 10. Additionally, the average size of computer music ensembles that respondents had performed with was two musicians, while the largest was 40 performers.

The following survey focuses on comparing and contrasting performing solo live computer music, and networked ensemble performance: which of the two approaches to live computer music do members of the Machine Orchestra prefer; what rolls do they see themselves playing as part of an ensemble; do ensembles afford any new modes of performance; and what are the challenges of performing computer music in an ensemble?

1 Do you prefer performing solo, or as part of an ensemble?

Every participant in the survey is currently, or has been a past member of The Machine Orchestra. With this in mind, each musician was asked if they preferred to perform as a solo computer musician or as part of a network ensemble. While 50% of the musicians preferred performing solo, only 28.6% preferred to perform in networked ensembles, with 21.4% not expressing a preference for either style of performance.

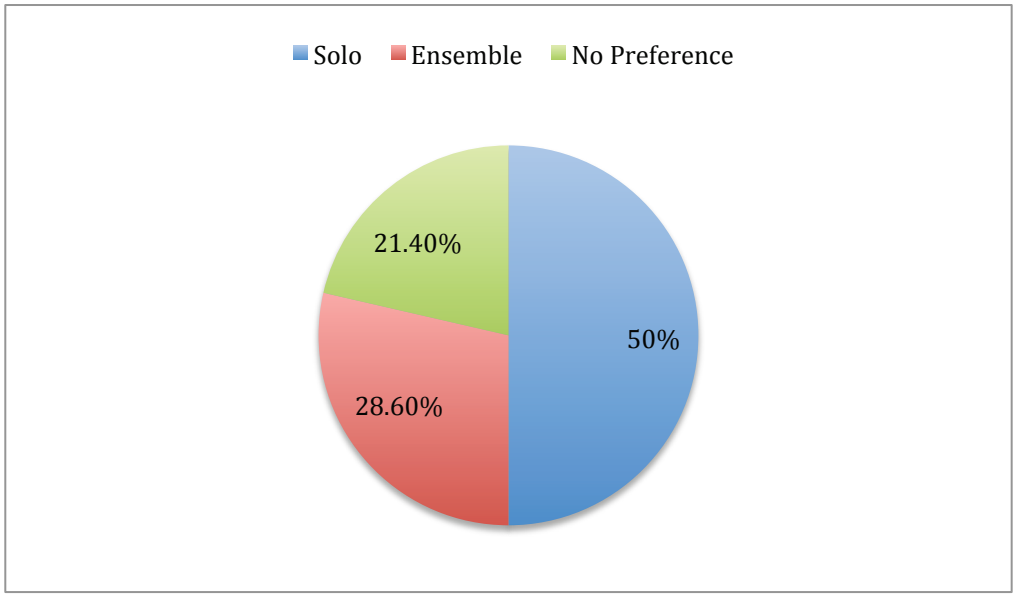


Figure 34 Preference performing solo computer music vs. networked ensembles

The musicians that preferred performing solo offered two main reasons for their choice: singular control over the performance, and simplified technical requirements. The musicians stated that they felt solo performance offered greater control over the composition, and sound design of the piece, while eliminating the chance of miscommunication or mistakes from other performers. With this mindset, the computer seems to be an ideal instrument for these control minded solo performers, allowing for musicians to extend their expressive potential through leveraging process such as automation.

The musicians that preferred performing in networked ensembles also gave several different reasons for their choice, with their responses centering on the interaction between performers. One reply stated that the distribution of parts among the ensemble afforded each individual musician more time to "react, improvise, and come up with original content…" This sharing of parts makes every musician responsible for only a few elements of the piece, and leads to increased improvisations complexity as all performers begin to modify their parts and musically communicate with each other. The same level of complex improvisation would be difficult for a solo computer musician to perform, although interactive musical systems like the ones discussed in Chapter 3 may be able to help close this gap. Other musicians stated that their prior experience playing in acoustic ensembles translated more to networked ensembles then it did to solo computer music, and that computer music groups allowed for "intense inter-performer collaboration" during performances.

Interestingly, several musicians did not have a preference for either solo or networked performance, and stated that both approaches to live computer music were interesting for same reasons mentioned above. One reply commented, "… both [are] fun for different reasons. Solo, everything can happen exactly the way you envision; [while] with a group you can trust, you can feed off new ideas that you would never manifest by yourself!"

Table 1 Reasons for solo or group performance preference

Preference	Response
Solo	I feel I don't' have to sacrifice my compositions as much when I do solo performances versus group
Solo	Less variables to control
Solo	More control over the composition process.
Solo	Playing solo would cut a lot of complications
No Preference	They are both fun for different reasons. Solo, everything can happen exactly the way you envision; with a group you can trust, you can feed off new ideas that you would never manifest by yourself!
Group	I prefer small group computer music performance. I define small groups as ensembles with fewer than four players. Ensembles of this size allow for intense inter-performer collaboration.
Solo	Clocking is not accurate enough to reproduce the exacting rhythmic qualities of my work on multiple computers.
Group	Groups provide a more even distribution of responsibility, allowing each individual to have more time to react, improvise, and come up with original content than a solo performer. Ideally I prefer the computer to be a part of an ensemble rather than the entire ensemble.
Group	Combining the social contexts and interactions of musics from differing cultures with new technologies is very interesting and challenging.
Solo	I have full control over every aspect of the arrangement and sound design without having to worry if someone else will be making any mistakes or changes that I'm not comfortable with.
No Preference	I equally enjoy performing in groups and solo, although I have more experience and am more adept at performing solo.
Group	I find playing in an electronic ensemble is much like playing in a band. Most of my experience as a musician has been in playing with other musicians. It makes the most since to me and I find it far more interesting and enjoyable than my solo electronic efforts.
Solo	Less chance of something going wrong during live performance, more convenient as a performer to be in total control of the whole setup, don't have to worry about miscommunication while performing
No preference	Both have their benefits and drawbacks. Performing solo allows for the most freedom in performance decisions while lacking the sense of camaraderie inherent in a group performance. Performing in groups allows for greater creativity and variety in approaching the performance but can be limiting and risky in terms of sync, overall mix, etc.

2 DESCRIBE YOUR ROLL WITHIN THE ENSEMBLE?

The participants of the survey were asked to describe the musical roles they performed within their ensembles. Most musicians described themselves as playing a section of the ensemble such as bass, melody, or rhythm. However, Several musicians described themselves as co-composer/producer/performer. This is interesting as it points the multiple roles that computer musicians see themselves in, and would position them to take advantage of performing at the note level, sound processing level, and the score level interactions contexts.

Table 2 Descriptions of each musician's roll within the ensemble

Response
Acoustic musician, string section leader.
Co-composer / Producer / Performer.
Generally I usually focus on the live arrangement.
I provided rhythmic elements, drone sounds, and soundscape elements.
I usually played the melody.
Live electronic music through laptop.
Time keeper/melodicist.
Composer /performer.
Mostly bass/bass synth in the large one; completely varies in smaller ones.
Sample triggering/processing. Live processing of signals from other member's instruments.
I generally take a roll of combining audio aspects with visual aspects. In the primary performance I have done as an ensemble, I performed all video aspects and no audio.
One of my electronic music projects, "Dead Waiter", started as a hybrid ensemble with multiple musicians. I basically wrote the songs and then had musicians help me perform them live. I played laptop, electric piano, and organ. The hybrid approach is always the most enjoyable to me because of the mixture of electronic and organic elements.
I was usually playing a specific type of instrumentation for whatever the song called for. Usually I'm most comfortable with drums and groove.
Performing with robotic instruments via network. Performing melodic and rhythmic material using controllers.

3 DOES BEING PART OF AN ENSEMBLE ENABLE NEW MODES OF PERFORMANCE THAT YOU DON'T EXPERIENCE WHEN PLAYING SOLO, AND IF SO PLEASE DESCRIBE THEM?

Every musican interviewed responeded yes when asked if new modes of performance were enabled by networked music ensembles. The reasons given varied from the enabling of call and response interaction between musicians, to the manipluation of material that is not self-generated. With this in mind, the new modes described by the musicans seemed to center on group communication, and interaction with material outside of one's own direct control. These modes of interaction can be viewed as social, and seem to be in line with network music's historical focus on interconnectivy between performers.

Interestingly, most replies did not describe new modes of performance; rather, the musician's felt that the distribution of parts among the ensemble enabled each musican to "focus on minutiae which would be impossible were they the only performer on stage". One musican replied that although this might not qualify as a new mode of performance, that "there is less to be done by one person in an ensemble, so that one person can hone more on elaborating on particular modes to explore new ideas winthin those modes." In general, most of the musicans replied that being able to split up the parts enabled more opportunities for improvisation than a solo performer with only two hands.

Table 3 Description of new performance modes afforded by computer ensembles

Response
More control over different parameters of the music.
Multiple layers of interactivity.
Being an aspect of a texture instead of driving the full sound.
Call/response more degrees of improvisation.
The human difference.
Being able to split up parts mainly, i.e. someone on drums, the other guy on basses or vice versa. Essentially anything 4 hands can do that 2 cannot. Same with feet if Softsteps, et al are involved.
First of all, the anxiety of live performance can be equally dispersed! As far as the music itself, the two things that are most obvious to me are the ability to have much more interesting dynamics and timbre. In an ensemble, you can actually NOT play and let parts evolve and change with more fluidity. That's much harder when you're the only one creating the music live.
It allows the musicians to take on different responsibilities for performing, such as delegating "instrumentation" as well as, considering audiovisual performance, the jobs of visual performer and audio performer, be they combined or separate.
Because I'm not physically capable of controlling all aspects of a song, being able to focus on certain elements of the song allows me to have a lot more control and musical depth (usually of drums).
By performing in an ensemble, individual electronic musicians can focus on minutiae that would be impossible were they the only performer on stage.
Gives the opportunity for manipulation of material that's not self-generated. Also allows for forms of "behind the scenes" preparation since you're not always obliged to be outputting signal.
Puts more focus on listening to each other as well as the music, thinking for more than one person while playing live.
Maybe not new "modes." There is less to be done by one person in an ensemble, so that one person can hone more one elaborating on particular modes to explore new ideas within those modes.
Even though I don't feel like the compositions are as tight, there are exponentially more options for things we can do as I am working with two more hands. Also, its nice to split the audio/visual control.

4 WHAT ARE THE CHALLENGES OF PERFORMING IN A NETWORKED ENSEMBLE?

The musicians were asked to describe any challenges they perceived with performing in networked ensembles. The responses fell into two main criticisms: networking issues such as maintaining a stable sync signal, or losing network connection; and musical complications in dealing with other performers, such as lack of communication, performance issues like over playing, and handling aesthetic differences.

The issue of working with multiple musicians within an ensemble is not unique to computer music; however, computer musicians, who usually compose and perform as individuals, may initially have difficulty adapting to these communal performance situations. This adaptation may be exacerbated by the fact that computer musicians are often working on developing solo live performance systems that are capable of broad control over all aspects of a composition. Once these types of systems have been developed, and learned as instruments, it may be challenging to "unlearn" this broad control, and instead learn to play a more focused part of the composition.

Table 4 Descriptions of the challenges of performing in computer ensembles

Response
Getting synced always seems to be the hardest issue to handle, although generally possible.
Relying on other's musicianship.
Sometimes the challenge is syncing up and depending on your partner.
With more options come more problems to troubleshoot.
Aligning focus, releasing control.
Decision-making is usually more democratic, which can lead to minor disagreements.
The human difference.
Losing network connection syncing difficulties.
The inevitable challenge of getting two or more performers to be in sync, not just literally but artistically as well.
When something unexpected happens during a performance, performers need to be able to quickly come up with a solution together.
A unified aesthetic can be hard to come by. Complexity in simplicity, especially when performing with live musicians, a lot of the time without specific direction, they'll overplay things instead of sitting back and becoming part of the texture.
Absolutely! If more than one computer is being used, the networking issues can be a huge pain. Additionally with electronic music, you have to create your sounds from scratch, sometimes even the instruments you are going to use. It's not like showing up with your violin and knowing your limitations or exactly what you can or cannot contribute. With traditional ensembles, your instrument usually dictates your role in the performance. With electronic music, unless the roles are decided upon in advance, you have to create your role and make sure that your contribution works with everything else that's going on. This becomes even more complicated when the ensemble is writing the music. However, as complex as the process can get, when it works, it can be far more interesting and enjoyable than a solo performance.
Timing has always been a challenge. The usual challenges associated with any group performance also exist: knowing when to step to the forefront of the sound and when to sit back and let others fill the space. With a laptop capable of creating an orchestra's worth of sound sitting in front of each performer, knowing when not to play becomes critical.
Technical computer issues is the biggest problem like syncing, sounds not sounding correct like how they were mixed at home. Coming to firm decisions or goals are harder with people that are very open-minded but it can also be annoying if a person in the group basically leads every aspect of the performance without any input from the other members.

5 HUMAN ETHICS APPROVAL OF THE SURVEY

This anonymous survey was conducted with the approval of Dr. Allison Kirkman, Chair of the Victoria University Human Ethics Committee, and Dr. Greer Garden, Associate Professor of music at NZSM. Final approval was given in the form of the following email received Thursday, 3 May 2012.

> Dear Owen
>
> Thank you for your applications. The three that involve anonymous surveys have been signed off by Associate Professor Greer Garden (as Head of School) and you can commence these now. The 4th one, the survey of contemporary approaches to live computer music, is not marked as anonymous. Is this because you are going to personally ask people the questions or is it because you will know each of the people who are answering the survey? Once I know more about the method of delivery I can assess the application.
>
> Could you consider these points and respond to me please.
>
> Best wishes,
>
> Allison Kirkman

The survey included here is one of the three anonymous surveys approved by Dr. Greer Garden, with the other surveys mentioned in the email ultimately being removed from the final thesis.

Appendix D

Probabilities and Markov Models

Probability has a long history of use in musical composition. The simplest example of probability-based composition is the use of chance procedures, with a famous example of this approach being John Cage's use of the I Ching. Chance procedures require the assignment of musical values to the outputs of a random process, such as rolling a die, with the value of each role being used to create a composition. John cage described his use of chance as a means of removing the self or the artist's "ego" from the compositional process (Reynolds 1979); however, the composer is still required to create the mappings between musical events and the random values the process generates, with Cage himself describing his role as consisting of choosing what questions to ask (Cage 1991). This would imply that it is impossible for the artist to ever fully remove themselves, or their ego, from the act of creation. They can only ever create distance between their aesthetic choices, and the influence that those choices impart upon the art itself. This is an extremely important consideration as the development of an interactive musical agent is pursued. As human artists will be designing these systems, it may be beneficial to think of these systems, as extensions of our own creative will through the application of algorithms, rather than the attempt to create an entirely autonomous agent.

With this in mind, another example of probability-based composition can be seen in the work of Iannis Xenakis. Xenakis was opposed to the idea of pure "chance" being used for composition, as he felt it abolished the role of the composer (Bois 1967). Chance procedures can lead to arbitrary relations between musical events, leaving the listener unable to discern a form or shape to the

composition. Xenakis developed compositional approaches drawing from mathematical fields such set theory, stochastic processes, and game theory. Through the use of these ideas, he was able to impart form and structure on the algorithmic processes. One such piece called Analogiques, used the probabilistic system known as a Markov Model (Ames 1989).

A *Markov Model* was originally created in 1906 by a mathematician named Andrei Andreevich Markov, and is a way of representing the likelihood of moving to one of many possible sates, given a current state. Musically, this is the likelihood of moving from a current note to other notes, where each transition between notes has its own probability associated with it. More specifically, a Markov Model describes the domain of a problem as a matrix of finite *transition probabilities*, where the *destination state* is dependent only on the current *source state*, and not on any previous states further in the past. This dependency on the current state is known as the *Markov Property*, and sequences generated by a Markov Model that satisfy this condition are known as a *Markov Chains*.

To see how this differs from a chance procedure, imagine there is a six-sided die, and we want to know the probability of seeing a given number. Assuming the die is not loaded, or biased in any way, we should have 1 in 6 chance of seeing any of the numbers, regardless of any previous numbers we've seen before. If we assign each number from the die to a note value within a diatonic scale—say 1 is equal to C, and 6 is equal to A—then we may compose a piece of music by rolling the die to decide each subsequent note. However, each note would have no particular relation to any of the notes preceding it, including the current note being played. In composing tonal music, it may be preferable to impose a hierarchy based upon a scale, with a preference for moving between certain notes. These preferences can be expressed as probabilities. For example, we may say that when in the key of C, a note of G will move to C 80% of the time, and move to A 20% of the time. A random roll of the die does not allow us to impart such relationships between the values returned by the process, whereas a Markov Model will. In fact, a Markov Chain can allow us to represent the probability of

transitioning from a particular note, to all other possible notes in the given scale (see Figure 35).

	C	D	E	F	G	A
C	30	10	10	10	30	10
D	40	0	10	0	20	40
E	0	2	8	50	30	10
F	15	0	15	10	30	0
G	80	4	2	2	2	10
A	90	0	0	10	0	0

Figure 35: A Markov Model representing the transition probabilities for the set of notes C through A. The notes on the left represent the source states, and the notes along the top are destination states. The values in the matrix are the transition probabilities for moving from a source state to a destination state.

1 MOVING IN A MARKOV MODEL

A Markov Model describes a transition matrix that can be used to create sequences of states known as Markov Chains. These chains are created in steps (see Figure 36), where each step matches an input state presented to the model with one of the available source states. This match determines the row of transition probabilities, which are then used to find the next destination state. It is crucial that the sum of all transition probabilities within the row does not exceed 100%; therefore all values must be normalized.

The following will give a description of the process of obtaining new destination states. The current state is used as the key and compared against all the source states in the transition matrix. A match between the current state and one of the source states is then used to determine the row of the transition matrix–henceforth referred to as the i^{th} row of the transition matrix–which will provide the transition probabilities. Once the i^{th} row has been determined, a random number between 0 and 1 is generated and then compared against the probability

stored in every column of the row–henceforth referred to as the j^{th} column of the transition matrix. If the random number is greater then the transition probability currently being compared against, then the value of the probability is subtracted from the random number, and the result is then used as the updated random number in the next comparison (see line 7 of Figure 36). This process is repeated until the random number becomes less then the j^{th} transition probability being compared, at which point the destination state associated with the j^{th} column is returned (see line 4 of Figure 36).

Algorithm 1 Returning the next state in a Markov Model

```
1: rnd ← (0.0, 1.0)
2: for j = 0 to (J - 1) do
3:     if rnd > P_ij then
4:         DestinationState ← j
5:         return
6:     end if
7:     rnd ← rnd - P_ij
8: end for
```

Figure 36: Algorithm for determining the next destination state in a Markov Model given a row of transition probabilities

2 N-TH ORDER MARKOV MODELS

Using the previously played note to provide a context for deciding which note to play next allows for a statistical model of movement between two notes. However, since musical phrases are rarely made up of only two notes, it would be better if the model could provide a larger context, consisting of more notes. One approach to this is to look at the current, one-note Markov Model as a 1st order model, since it only has one note to provide the context. Adding another note, creating a source state of two notes, creates a 2nd order model. This 2nd order model would have a complete 1st order model for every 2nd order source state. As shown in Figure 37, this would look like a cube.

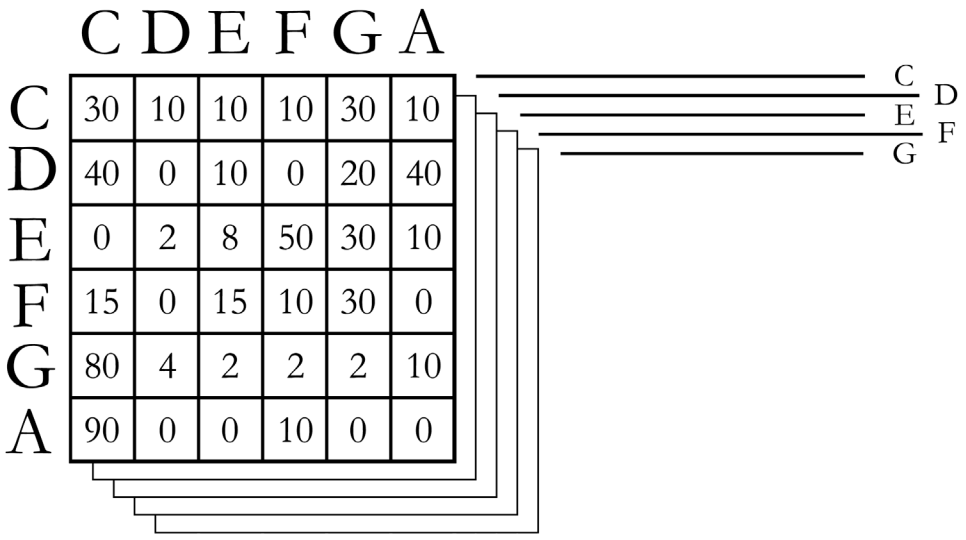


Figure 37: 2nd order Markov Model

As the length of the source state sequence increases, the order of the model gets higher, and visualizing the model as a shape becomes impossible. Fortunately, every N^{th} order model has an equivalent 1^{st} order formulation. This type of Markov Model can be represented as a matrix whose source states are sequences, with the number of source states in the matrix increasing as the order increases, while the number of destination states will remain the same (see Figure 38).

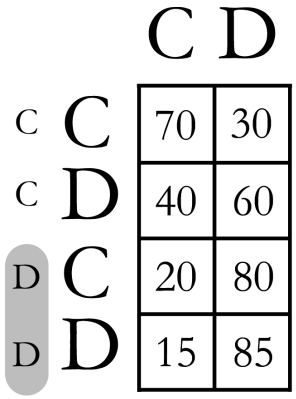


Figure 38: 2nd order Markov Model shown as 1st order Markov Model

While the source states in these models are described by sequences of states– therefore providing a larger context for determining the next destination state– they only describe a change in pitch. Additional information can be added to each state within these source state sequences, effectively transforming them into sequences of vectors. For example, the first source state in Figure 38 is made up

the notes C and C. For each of these notes, it is also possible to add additional information about the velocity and note duration. However, each added piece of information increases the complexity of the Markov Model.

3 TRAINING THE MARKOV MODEL

As the Markov Models become more complex, the transition matrices also become larger in size. This increase in size can make the matrix prohibitively difficult to fill in by hand, and the increased complexity can make the relationships between each transition more complex to understand. Additionally, manually deriving the statistical values of the matrix that represent the complex relationships of a particular system requires the evaluation of a large amount of data. An alternate approach to populating the matrix by hand is to enable the Markov Model to automatically learn the transition probabilities through training.

When training the model, it is only required to store a sparse collection of non-zero transition probabilities. If a transition has a probability of 0% then it has never occurred, and therefore does not need to be stored as part of the model. Each transition probability greater than 0% can be stored as the source state, the destination state, and the transition probability, with the transition probability represented as the number of times that particular transition has been seen by the Markov Model during training (see Figure 39). This allows for a much smaller amount of data to be stored when representing the model, and also speeds up the search time when attempting to match source states. When using the model, all stored transition probabilities that share the same source state are returned, effectively making up the row of all the destination states that have non-zero transition probability. Lastly, the values for transition probabilities are stored as positive whole numbers representing the number of times that particular transition has been seen. These values need to be normalized before the destination state can be chosen.

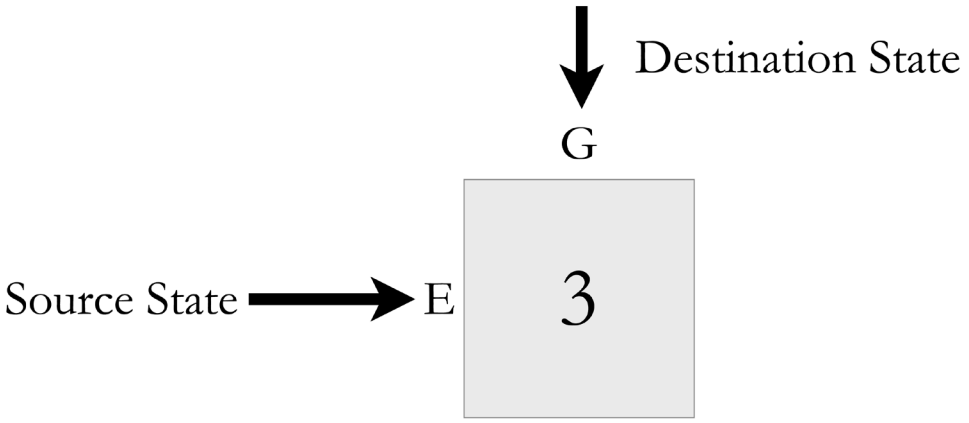


Figure 39: The description of a transition probability, containing the Source State, Destination State, and the number of times the transition has been observed during training

Although storing the Markov Model as a sparse collection reduces the amount of data required to describe the model, and increases the speed of use, there are data structures that are more efficient.

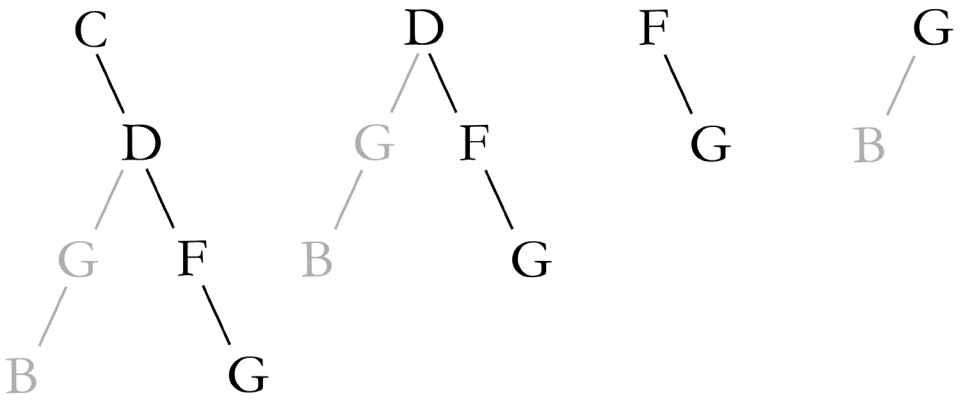


Figure 40: Markov source states stored as tree structure. This allows for searching variable length source state sequences.

One such data structure is a tree (Pachet 2002), where each node in the tree points to the destination states that are the associated with the sequence built from the root of the tree. In Figure 40 the root notes C, D, F, and G all represent a 1st order source state, from which previously seen sequences of source states can be built. During use, the system can attempt to match an input sequence with the longest matching sources state sequence stored in the trees. Once a match has been found, the last node in the sequence holds the transition probabilities for all destination states attributed to that source state sequence.

For example, if the Markov Model tree in Figure 40 is presented with the input sequence F G D C, then the tree would return all the transition probabilities and destination states associated with the sequence terminating at the left most G in the first tree (see Figure 41). However, if the same Markov Model were presented with the input sequence G A D C, then the tree would return all the transition probabilities and destination states associated with the sequence terminating at the D in the first tree (see Figure 42)

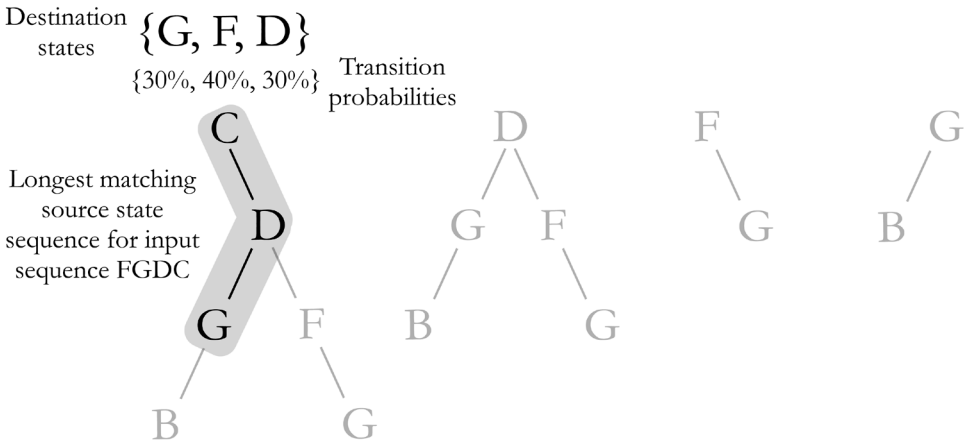


Figure 41: Longest matching source state sequence for input sequence F G D C

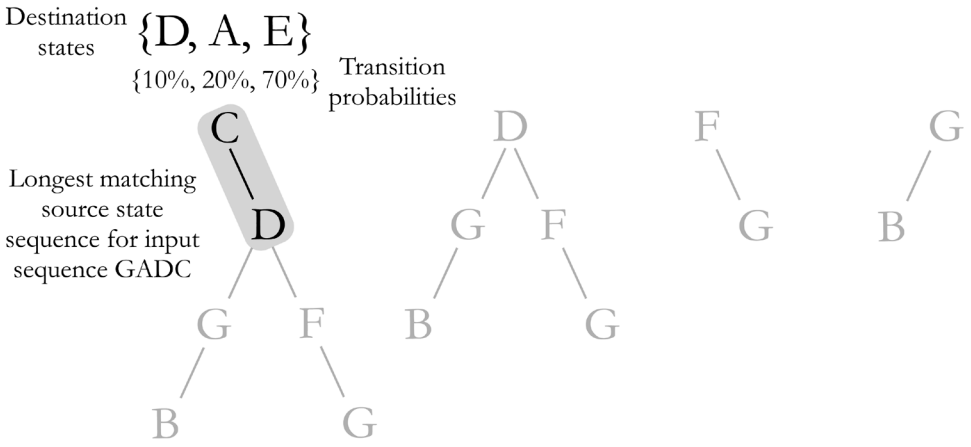


Figure 42: Longest matching source state sequence for input sequence G A D C

4 MARKOV MODELS FOR INTERACTIVE MUSICAL AGENTS

In order for Markov Models to be suitable for creating interactive musical agents, they must be able to react to real-time input provided by a human performer. Markov Models are capable of describing a domain and producing sequences that are statistically consistent with the modeled system. However, the models are traditionally set up to take the previous destination state as the next source state for the next iteration through the matrix. Setup in this way, it is not possible to allow for external inputs to influence the system.

One possible solution during training is to include in the source state both the performer to be modeled, and the performer that will be the live input. Although this will allow for the inclusion of the live input during performance, it also increases the complexity of the source states, and decreases the likelihood of finding a matching source state during performance. An alternate approach was proposed for the Continuator system (Pachet 2002) that applies a fitness function to the output of the model. This function takes the destination state provided by the model as the inputs, and a context provided by the human performer. Additionally, the fitness function can be weighted to impart greater or lesser influence over the output. This could be roughly equated to the amount of influence the band[28] is having on the musical choices of the virtual musician. Although this technique allows for a live performer to influence the output of the Markov Model, it also modifies the probability that the model will produce the resulting Markov Chain. Essentially, applying a fitness function to the output of the Markov Model changes the model itself.

An additional approach to contextualizing the Markov Model is the use of constraint satisfaction problems. This approach takes a set of constraints, and then applies them to a set of domains for a set of variables (Anders 2007). In

[28] A band in this context could be a solo live computer musician, or an ensemble of computer musicians.

Pachet's recent work with variable-order Markov Models (Pachet and Roy 2011; Pachet, Roy, and Barbieri 2011), he explores this approach for creating new matrices, from a master trained matrix, using constraints. In this case, the performing musician would provide the constraints, the domain would be the Markov Matrix, and the variables would be all the possible transition probabilities. The constraints would negate the likelihood of certain transition probabilities, in effect creating a new matrix from the existing master matrix, where constrained transition probabilities have been set to 0. Pachet goes on to discuss the difficulty in adjusting the matrix transition probabilities to ensure that the likelihood of returning a given sequence, relative to some other possible sequence, is the same as in the original matrix. Simply re-normalizing the rows would effectively alter these relationships, therefore effectively altering the modeled system. Pachet's earlier versions of the continuator avoided this re-normalization issue by applying a fitness test to the sequences once they were generated. Although this worked, it meant that the system either needed to generate sequences until one passed the fitness test, or that the system needed to alter the output of the Markov Model, which would again lead to changes in the model.

5 IMPLEMENTATION

In practice, a simple first order Markov Model proved to be very easy to train up and run in real-time; however, the model was only built to control the values of a single knob. In order to expand the model to control multiple knobs, two options were considered: creating a sources state that contains each controller's value; or creating a separate Markov Model for each Controller. Both solutions presented possible issues.

The first approach creates a very large domain, meaning most of the time there will not be a match between the input state and a non-zero source state, and a solution such as quantization to the closest matching source state will be needed (see the following section for a discussion on this issue). The second approach is to have a model for each controller, which simplifies the domain of each model,

but also makes the different controllers independent of each other. This is a possible issue as during performance, several controls are used together to get more complex processing, and these relationships are a critical part of the interaction.

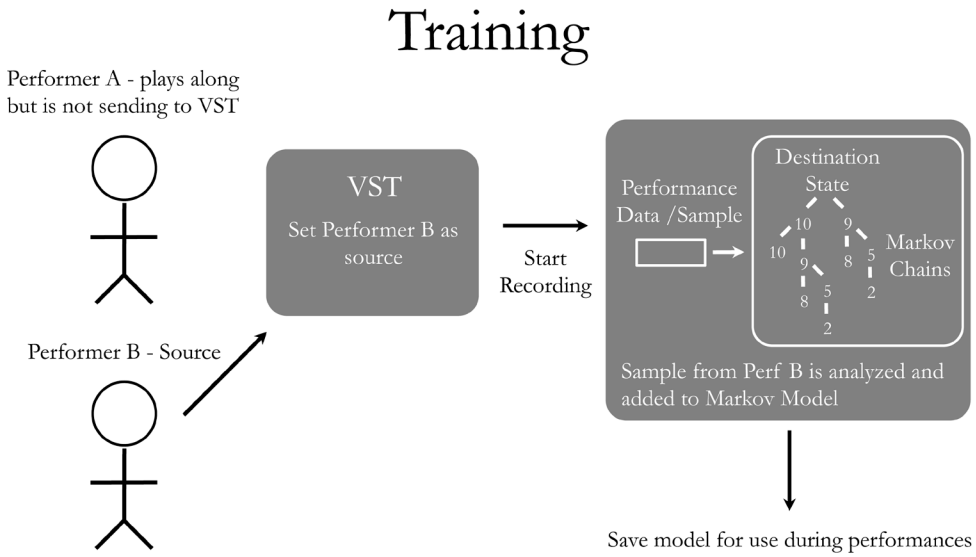


Figure 43: Training diagram of Markov Model

A plugin Markov Model was built for testing, with the incoming value of the controllers being sampled at 200 Hz. With the source state only consisting of the previous value, a match was usually found; however, as the context provided by the source state was very short, the resulting sequences tended to be erratic. As the length of the source state was increased, the number of matches went down, and the quantization to the nearest Markov Chain also introduced erratic output.

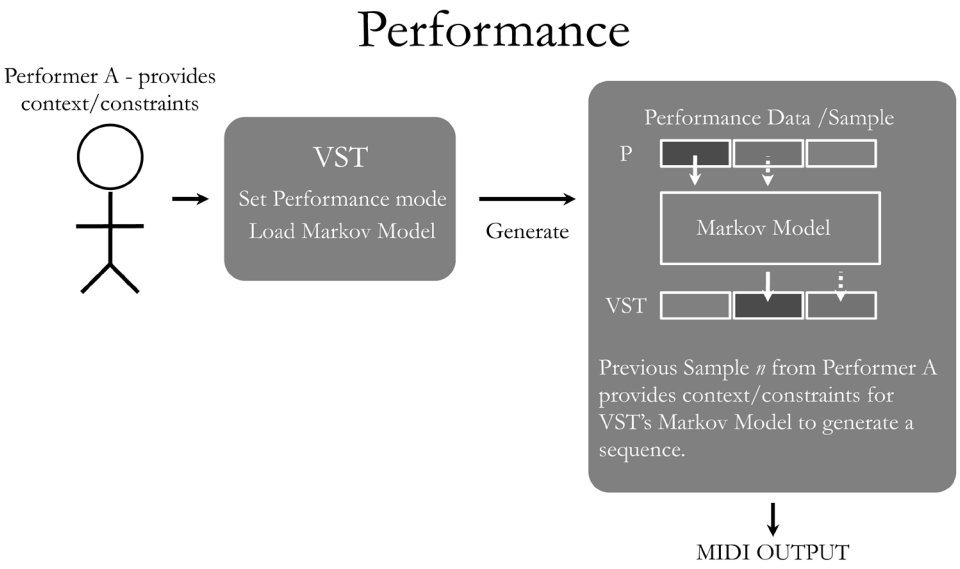


Figure 44: Performance diagram of Markov Model

While Markov Models are relatively simple to train, and efficient to run in real-time, they prove challenging to control using an external context, and difficult to find methods for handling the case of not finding a match without altering the probabilistic distribution. Even with these challenges, Markov Models look like a promising approach to developing interactive musical agents for use with control change data. Combining a variable length Markov Model, like the one described by Pachet, with other techniques learned from search and regression approaches may yield a working system.

6 Challenges with using Markov Models

One of the most difficult limitations of using a Markov Model is that they are not capable of handling input states for which it has no statistical information. For example, if a Markov Model is trained to produce melodic lines, and a note is input that has not previously been seen by the model, then there is no way of handling the unseen input state (see Figure 45). The unseen state effectively has a row of 0% transition probabilities associated with it, resulting in no paths from the source state to any of the destination states. This situation becomes increasingly likely as the complexity and number of source states increases, and

leads to a paradoxical case where increasingly detailed source states provide more "accurate" statistical models of the system, but also increase the potential number of source states with 0% transition probabilities.

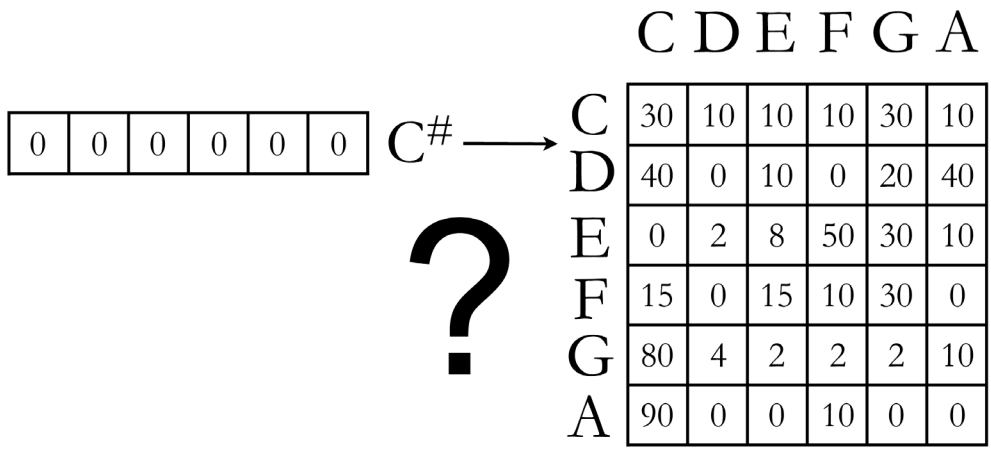


Figure 45: Input to Markov Model that does not match any previously seen source state

While different solutions to this challenge exist, many of them have drawbacks. Several of these solutions are as follows:

1. The system may choose a random source state in the model; however, this would alter the statistical likelihood of the Markov Model producing the resulting Markov Chain, effectively altering the model itself.

2. The system may choose the destination state with the highest probability.[29] However, the resulting Markov Chain will exhibit the same issue as described in the first solution.

3. The system may choose the nearest source state to the input state. This will help preserve the statistical relationship between the model and the Markov Chain, but it also implies that the model does not completely represent the domain.

[29] Taking the sum of the probabilities in a column, and then dividing the result by the sum of the probabilities of the entire matrix can determine this destination state.

4. The system may have multiple models that describe the domain, with each model reducing the amount of data used to describe the states (Pachet 2002). If a more specific model fails to find a match for the input source, then the next most complex model is tested. This process is continued until a match is found for the input source. However, as the models become more general, they also become a less accurate representation of the system.

These solutions provide various methods for handling unseen input states, and allow for Markov Models to be a robust approach to statistically modeling the change in musical parameter values over time. However, it may be that through the application of other approaches, such as search algorithms, that more elegant solutions for handling these unseen input states will appear.

Appendix E

SEARCH-BASED ALGORITHMS

In the simplest implementations, search systems will take in a state, or sequence of states, and attempt to find a matching state or sequence within the database. Assuming that states in the database are stored in the order they originally happened, then the states can be described as a list of sequential events. The search can then compare the input sequence against all subsequences of the same size in the database. This type of exhaustive search can become slow as the number of states stored in the database increases. This leads to a situation where increased data collection creates a more accurate model, but also increase the number of compares required to find a match. This can increase the search times, and render the system too slow for live performance use. Dannenberg suggested constraining the search by associating the current position within a score with a location in the database, and limiting the search to just before and after that position; however, Rowe argues that improvisational music does not have a fixed score, and therefore does not benefit from such constraints.

Even if computers become fast enough, that the time it takes to perform an exhaustive search is not an issue, then there is still the complication of finding a perfect key-value match between the input and the database. When playing from a score, part of the perceived musicality of the performance is the minute variation imparted upon the written notes. Additionally, musicians occasionally miss or drop a note, or often embellish a score with additional ones. This implies that every performance will be subtly different, and makes it unlikely that an exact match of the input sequences will be found in the database. This problem becomes even more challenging with improvisation, as there is no scored material to constrain what notes might be played. While this issue is similar to

matching input states to source states within Markov Models, the solutions we will discuss are very different.[30]

Lastly, search algorithms must ultimately produce an output sequence of states. These sequences can be built by concatenating the next state in the database after the end of the matching sequence (see Figure 46). In the figure below, the sequence CDFGADG is matched in the database. Once the match is found, the algorithm returns the next value stored in the database that occurs after the sequence. In the figure below the value returned is C, and this value is then used as the next output from the system.

Figure 46: Matching sequences return the next stored state form the database

While this approach works for creating melodies, or other generative compositional tasks, an alternate method must be used when designing interactive musical agent systems. The aim of an interactive musical agent is to take input, and then return contextually related output such as auto accompaniment from a score, or sequences from other performers. In Figure 47, the database is designed such that it samples and links two sequences during training. This key/value linking is similar to a dictionary, or hash map, and allows for the search to return values representing a second performer, by using live input as a key. In the figure below, the live input sequence CDFGADG is used as the key to find a match in the database; however, unlike the previous example, upon finding the match the system uses the next state C to find and return the second performer's value E. With this approach, it is possible to take input from a live performer, and use it to find sequences representing another performer.

[30] Exploration of the similarity between the search algorithms evaluated here and the Markov Models discussed in Appendix D may lead to interesting hybrid systems; specifically regarding approaches to handling the matching of sequences.

Figure 47: Matching sequences return values from a second linked sequence

1 AUTO ACCOMPANIMENT

One of the earliest implementation of search algorithms for interactive musical agents was Dannenberg's (1984) use in developing an auto accompaniment system. This system attempts to listen to input from a performer, use this input to determine the current location of the performance within a fixed score, and then provide the appropriate accompaniment. Additionally, the system attempted to handle any deviation of the human performer from the written score.

If the actual performance and the written score can be seen as two sequences of states, then Dannenberg's system is attempting to find the best match between these two sequences. With Dannenberg defining 'best' as: "The *best* match is the longest common subsequence of the two streams (Dannenberg 1984)." The challenge is in allowing for extra notes that are not in the written score, or to recognize when the musician has left a note out of the performance. We will describe an overview of the system in order to provide a context for discussing the challenges of constraining a search for improvisational systems. A full description of the system can be found in (Dannenberg 1984).

The system uses a matrix to compare the live input against the written score. The score is represented as rows, while the live input is added as columns. When a new note arrives at the computer, it is appended to the column sequence, and then the two sequences along each axis are compared. A perfect performance would create a match along the diagonal of the matrix, beginning from the corner where the two sequences start. For every note that is a match, the number

in that cell is increased, however, if the note does not match then the cell is filled with the previous value (see Figure 48).

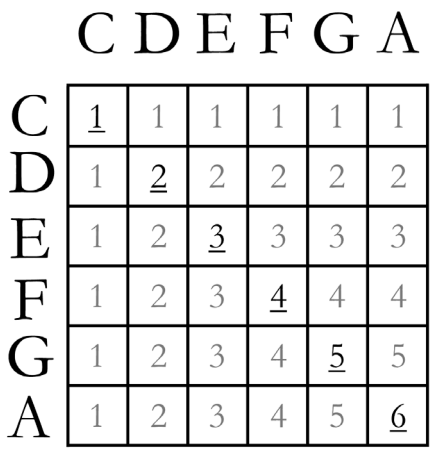


Figure 48: Matrix with ideal rating scores comparing input sequence along the columns, and scored sequence down the rows

This matrix compares the live input to be against a fixed score, while allowing for deviations in performance. In reviewing Dannenberg's approach Rowe describes four alterations to the fixed score that the matrix can test for (Rowe 2001).

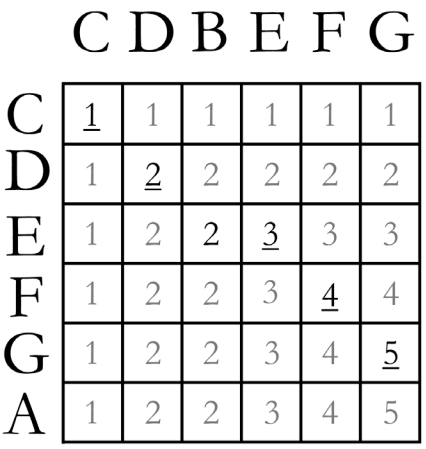


Figure 49: Insertion test: this tests for notes played by the performer that are not in the original score

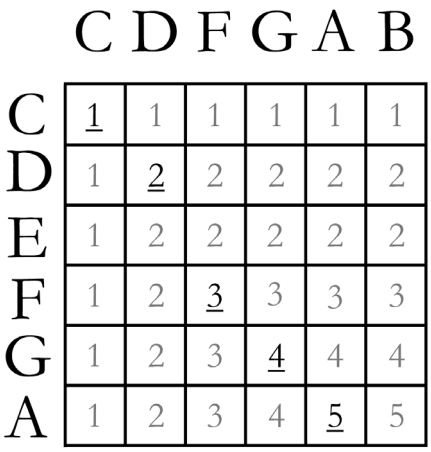


Figure 50: Deletion test: this tests for notes from the score that are skipped over by the performer

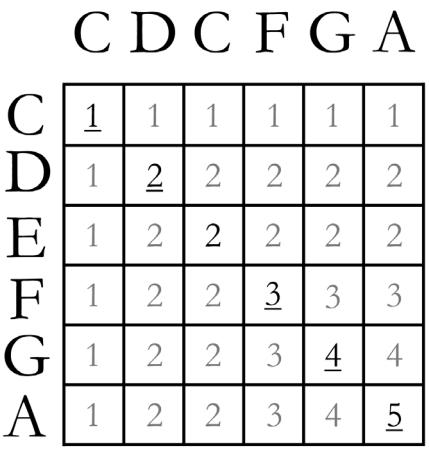


Figure 51: Substitution test: this tests for notes substituted by the performer

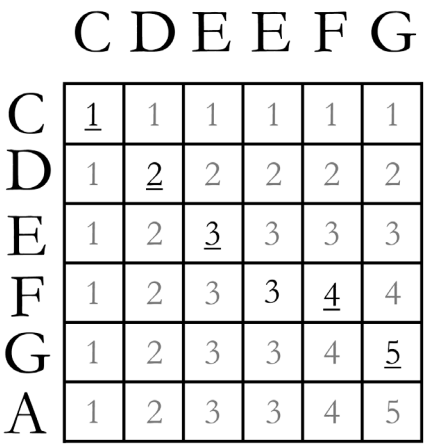


Figure 52: Repetition test: this looks for notes from the score that are repeated by the performer

With these four matrix tests, the system is able to determine the correct position within the score, regardless of performer deviation or embellishments, and

generate contextualized accompaniment. Additionally, the search is limited to the scored note before and after the current position within the score (see Figure 53). This constraint increase the search speed, and can be applied as it is assumed that a performer will not deviate from the score by more than one or two notes at a time.

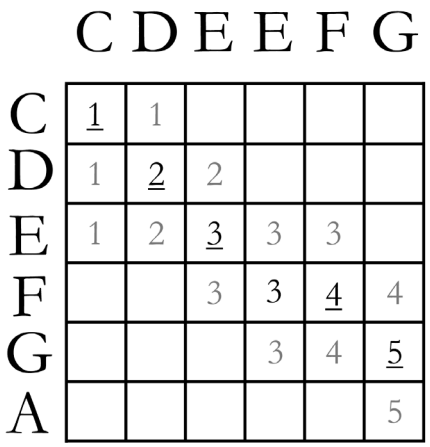


Figure 53: Constrained search to speed up sequence comparison

Appendix F

REGRESSION SYSTEMS

A third approach to designing interactive musical agents is the use of regression algorithms. Regression is a process by which a function is fit to a dataset in such a way, that for new inputs, the function will return the average value for that input (see Figure 54). Additionally, once the function is fit, the system is capable of handling inputs it has not previously seen. The function essentially interpolates between the data it has been presented with during the training phase, and is able to make an informed "guess" at an output value. This could be very useful for musical situations, where new combinations of a performance might lead the system to output new ideas. A thorough overview of regression can be found in (Fiebrink 2011).

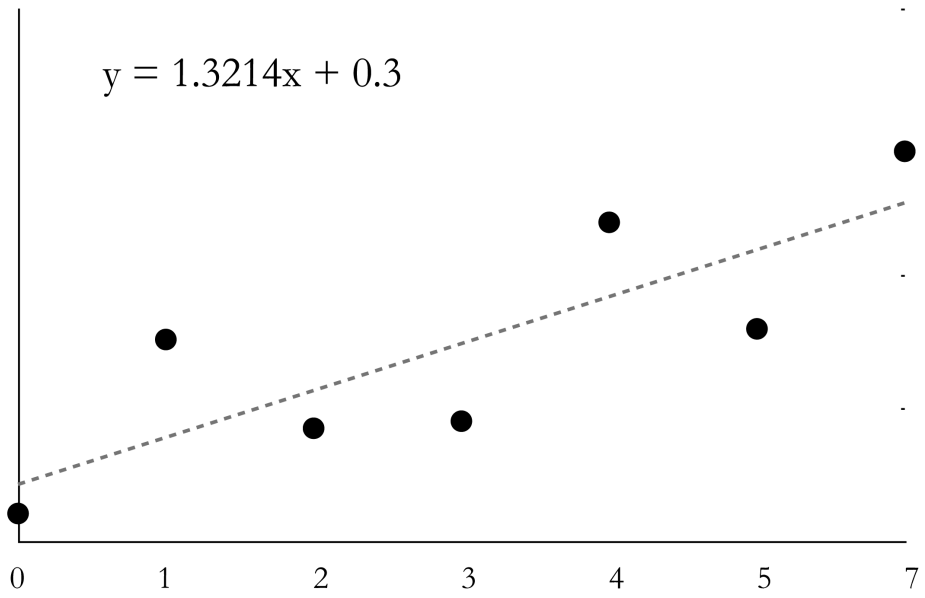


Figure 54: Basic linear regression

The simplest example of regression is fitting a straight line to a 2D scatter plot of data (see Figure 54) through a process such as least squares.[31] These functions can also be extended to handle mapping multiple inputs to a single output. Each parameter can be represented by a function that is plot on its own 2D graph. These functions can then be combined in such a way as to solve for complex multi-dimensional mappings.

Artificial neural networks, or ANNs, can be thought of as an advanced method of implementing these more complex linear regression[32] models (see Figure 55).

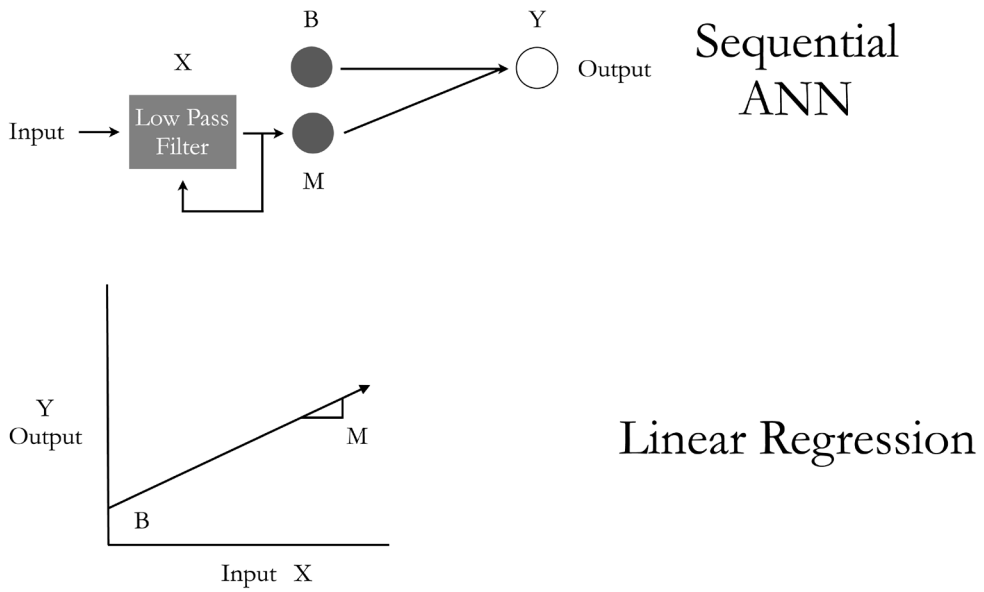


Figure 55 Similarity between linear regression and basic sequential ANN

Not all data will best be described using a linear function, and fitting a nonlinear function using basic regression requires greatly increasing the number of input parameters in order to accommodate the higher order polynomial functions. ANNs are capable of automatically fitting these nonlinear functions through the

[31] Least squares attempts to minimize the average squared error between the value produced by the function, and the actual value from the training set.

[32] Depending on the activation function used, such as a sigmoid function, ANNs can also be classifiers.

use of *Hidden Layers*. These layers are placed between the inputs and outputs of the system, and can be thought of interconnected stages of several regression systems. Through the use of these hidden layers, an ANN is capable of fitting more complex nonlinear functions then would be feasible using basic regression.

1 MULTI-VARIATE LINEAR REGRESSION

There are many ways to implement regression systems, from basic linear regression to more complex Artificial Neural Nets (ANNs) with multiple hidden layers. Basic linear systems represent straight line functions through the data, and are easy to build, simple to train, but may not be able to fit more complex, non-linear relationships; conversely ANNs are powerful models capable of fitting complex, non-linear functions, but they can be harder to build, and may require a large amount of training to get the functions to closely represent the data. As a starting point, the research presented here built a network of linear regression systems, with each one modeling a different controller, and is described as multi-variate linear regression.

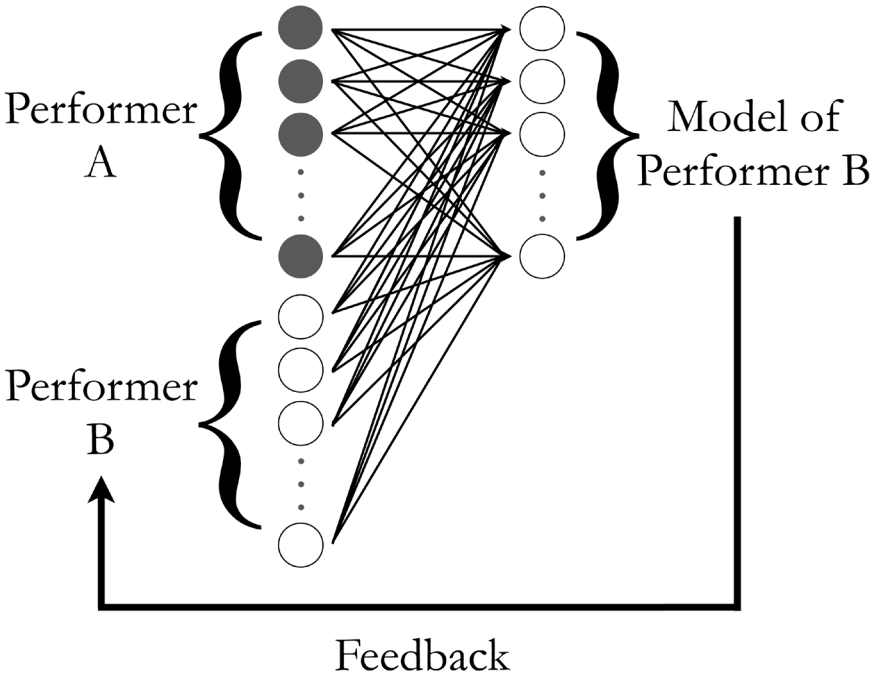


Figure 56 Diagram of multi-variate linear regression model

A basic linear regression system can be thought of as a simplified ANN, with no hidden layers. Viewing the system in this way allows for the design to be visualized in terms of an ANN. One of the reasons for describing the system in this way is that it affords the ability to conceptualize the system as different but related networks of connections between multiple input and output parameters (see Figure 56). Essentially, each output can be thought of as an independent network of connections to all of the input parameters. This view of a connected network has become critical to my understanding of the development of interactive musical agents, and helped to inform my research and design using other approaches, such as the S2MP algorithm described in section 3.4.4.

With this interconnected architecture, each output in the system can be thought of as an independent system, and therefore the entire system can be thought of as a collection of independent regression systems. Multi-variate linear regression then fits a regression function for every output. These individual linear

regression problems represent a supervised learning problem, and as such are normally solved through an iterative process such as gradient descent. These approaches attempt to minimize the error between the function and all the data points. However, a shortcut closed-form solution known as the normal equation can be used to derive these functions directly (see Equation 1). The system uses this normal equation to then solve the linear regression problem for each output separately, effectively creating a matrix of linear regression functions.

$$\boldsymbol{\theta} = (\boldsymbol{X^T X})^{-1} \boldsymbol{X^T} \vec{y}$$

Equation 1 Normal Equation

Once the model is complete, the system takes new inputs from performer A and the previously calculated outputs for the virtual performer B, and uses these to generate new output. The models will return continuous values however, requiring the output to be quantized into the discrete steps used by the system. For musical applications this would be equivalent to "tuning" each output to the nearest semi-tone. Additionally, this continuous output can be looked at as the strength of belief in a particular value being the output.

2 CHALLENGES WITH USING REGRESSION BASED SYSTEMS

While the approach that was just described allows for all the inputs to influence the output, there still remains an issue of providing a temporal context to the system. Unlike the Markov Models or the search algorithms, the inputs to the regression system only ever see one state of the performance at a time. Each single state during a musical performance can lead to many different notes, e.g., C# could perhaps equally lead to G or F. With the averaging potential of a regression system, C# would end up returning G# instead of one of the two notes. In order to know which of the two notes the system should output, a greater context of what has previously been played must be presented to the system.

One solution to this issue is to contextualize the inputs by providing memory of the previous states. This can be achieved by applying a low pass filter to the inputs, effectively feeding a decaying amount of previous states into the current state. Rowe describes the early work on these types of systems, and labels the process as *Sequential Neural Networks* (Rowe 2001).

Appendix G

COMPARISONS AND REQUIREMENTS

Over the course of this research it has become clear that designing an effective interactive musical agent for use with control change messages has several requirements: it must be able to learn or train from previous performances; it must be able to link, or infer relationships between different controls; in a manner similar to a musician listening and responding to the rest of the band, it must be able to take external input in to contextualize, or influence the output of the system; it must be fast enough to work in real time; and finally, it should have enough memory of past events as to provide a context allowing for a musical dialogue between the human performer and the system. The following sections will evaluate these requirements for each of the three approaches described in the previous appendices.

1 TRAINING

In order for an interactive system to adapt and become better at improvising, there needs to be a method for learning from previous rehearsals (Vercoe and Puckette 1985). Through training, a model can begin to develop a picture of what a performer might do during a piece, and what responses are appropriate. This training can happen either offline, with the system analyzing the data after the performance, or in real-time, allowing the model to adapt during an actual performance of a piece.

A Markov Model allows for real-time learning by using the source and destination state pairs to update the transition probabilities. Real-time training of a Markov Model is relatively easy as it can take new inputs and alter the transition matrix, allowing the model to change during a performance. This can

allow the system to respond to new information and change its behavior during a performance. One potential issue with this process would be influencing a model whose transition probabilities are based on large numbers of previously seen data. Adding a small number of events to a model like this would not alter the transition probabilities very much. Inversely, a transition matrix that had only seen a small amount of data may exhibit large changes in transition probabilities during a performance.

Search algorithms can be trained in real-time, adding new Key/Value pairs during a performance. However, as mentioned in section 3.4.2 the pruning of "bad examples" may be needed to prevent the system from finding and matching poorly played material. Additionally, training a search system can create large databases, with much of the data being very similar. Using data structures like KD trees may be a good solution to reducing the amount of training data stored to represent the model. This would also help to increase the search speed of the systems.

Regression models take input training data and build a function to describe the model instead of attempting to find a match. This function will interpolate between the data presented during training, and allows the system to return values for inputs it has never seen before. However, with small amounts of training data, the interpolation can be very coarse, while larger amounts of data can create a more accurate model. Additionally, unlike the large databases created by the search approach, the training output of a regression system results in a matrix of functions. As the amount of training data increases, the functions themselves change, but the total number of functions stays the same. Lastly, these systems will most likely be trained off-line as they usually require an iterative approach to building the model. It may be possible to build the models in real-time as separate processes in the system, but this makes training more complex then probability or search based systems.

2 INFERRING RELATIONSHIPS

As shown in Figure 27, there are several different configurations of inputs to outputs. The complexity of training a model is dependent on these configurations, with the most complex configuration being a system that listens to all inputs from both the human performer and the model, and then relates all inputs to each output. Additionally, the ability of the model to generate output in the style of a particular performer is also related to these configurations, as some of the performer's behaviors may be the result of the relationship between several input parameters.

The Markov Model is the most challenging of the three approaches for linking relationships between multiple controllers. When linking multiple inputs, each source state in the Markov Model can be thought of as a snapshot of the inputs, and as discussed in Appendix D as the number of inputs described by the source state increases complexity also increases. This creates challenges with finding a matching source without altering the probability of the resulting Markov Chain. However, the variable length Markov Model designed by Pachet (Pachet 2002) has shown that there are graceful compromises for handling these situations.

Searching algorithms such as S2MP (see section 3.4.3) are capable of handling complex state descriptions due to their ability to generalize during the matching process. S2MP does not need to find an exact match, but rather returns a similarity score based on item set members and order. This thesis presents a novel technique for linking these complex relationships between controller states using sorted sets, sparse sampling, and single vectors that represent the 2D controller number/controller value relationship (see section 3.4.4). The use of this kind of generalized searching combined with the complex input representation, may also be useful in Markov Models for matching source states.

With regression based approaches, linking multiple inputs to multiple outputs is possible through the interconnected networks created by approaches like multi-variate linear regression (see Figure 56); however, as the number of inputs

increases, the resulting functions may require greater amounts in training data to allow for musically usable outputs.

3 EXTERNAL CONTROL

With all three approaches the output of the model is fed back into inputs during performance. This allows for the system to play along without any outside input, essentially remaining self-contained and autonomous. However, in order for the systems described in the previous appendices to be interactive, they must not only autonomously generate new material with which a human musician can react, but also similarly allow external input to contextualize or influence the model. There are several different ways that this external influence can be imparted on the model (see Figure 27).

Applying external control to a Markov Model has been discussed in Pachet's work (Pachet 2002; Pachet and Roy 2011). The methods presented in Pachet's work are either influencing the output of the model through the use of a fitness function, or by applying constraints to the transition matrix itself. A fitness function ultimately amounts to altering or influencing the underlying probabilities, and therefore the model itself. Pachet's 2011 paper presents an alternative Constraints Based Programming approach that attempts to compensate for this change in probabilities, and thereby maintains the original probabilities of the Markov Chains. However, neither the fitness function nor the constraints approach is clear on exactly what data from the musician's input is to be used to contextualize the model. One possible fitness function could take in an "activity level" from the human performer, basically acting as a damper to restrict the output value range. Regardless of what is chosen, it seems that these fitness functions must be explicitly decided by the developers of the system, rather then inferred by the connections or relationships between inputs and outputs. One exception could be to use inputs from both the human and the interactive agent as source states, but this creates the ballooning of source sates mentioned earlier in Appendix D.

With search algorithms it is possible to represent all relationships between inputs and outputs by collapsing the controller numbers and values into a single vector and sampling the state at a regular interval (see section 3.4.4). The complexity brought on by representing all the controllers can be managed by only storing the controller values that have changed since the last sample. Additionally, no other information, such as probabilities, is required to be stored with the database. This is because the regular sample rate implies time, and the returned value is then simply the next value in the database. Regression based systems are similar in that the interconnected relationships described in Appendix F also allow for all inputs from both the human and the virtual performer.

4 SPEED

While all three approaches discussed in the previous appendices are capable of being used to model a virtual performer, they must be able to run in real-time in order to be useful as an interactive musical agent.

Using a trained Markov Model is essentially a two-step process consisting of a search, and the returning of a probabilistically derived result. The speed of the system depends largely on the search portion of the system, and suffers from similar speed issues as the search based approaches. However, Markov Models will only store one version of any given source state, while a search based system may store many different examples of the same, or almost exactly the same sequence. This means Markov models may be able to represent the same model as a search based system while using far less data.

As mentioned above, search based systems may store multiple examples of the same sequence of data. This leads to the situation described in 3.4.5, where the size of the database grows as more training data it is presented to it. Effectively, as the model learns more, and becomes more accurate, the database grows and the search time increases. While placing constraints on the searches results in far fewer searches being carried out, this only puts the problem off. At some point the database will become large enough that the system will again become too

slow to use in real-time. However, with clever segmentation, and possible offline data clustering, it might be possible to push that point far enough away so as to not be a concern.

Regression based systems use a fixed number of functions, and as such the speed of the system is tied to how fast the computer can solve the functions. Assuming the computer can complete all the calculations in time, any new training data should not significantly increase the time it takes to calculate new outputs.

5 CONTEXT AND MEMORY

Music happens in time, and musicians performing together decide what they will play in the future based off of what they have played in the past. Any attempt to design an effective interactive musical agent must take this musical memory into account when deciding what events will be played next.

Context can be added to Markov Models through the use of sequences of source states, but the longer the sequence is, the harder it will be to find a matching source state. Variable length Markov models are a nice solution to the problem, as they allow the longest sequence available to be used. This will ensure the largest context available is used to generate new events, but the approach still requires a way to handle the case of not finding any matching source states during performance. Pachet solves for this by storing multiple representations of the model, with each one being a data reduced version of the previous one. This increasing coarseness of description allows the system to start with the greatest detail possible, and then work towards more general descriptions of the state. This process increases the likelihood that a match will eventually be found, while providing the largest possible context.

The S2MP algorithm presented in this thesis requires the samples to be provided at a regular rate. These samples then represent a sequential record of the history of a performance. Increasing the context for the next event is a simple matter of increasing the size of the sequence of samples searched for. However, increasing the size of the sequence also increases the search time, and therefore slows down

the algorithm. Part of designing an effective search algorithm requires balancing these two requirements, i.e., using large enough sequences so as to provide a meaningful context, and optimizing the algorithm so it is fast enough to use in real-time.

Providing a regression system with memory of past events is achieved through low pass filtering the inputs as described in Appendix F. These types of systems are known as sequential neural nets, although they can apply to simple linear regression systems as well. The lower the cutoff frequency applied to the input, the more influence that previous states will have on generating output.

Table 5 Overview of algorithms for designing interactive musical agents

	Probability	**Search**	**Regression**
Training	Training can happen in real-time. Over time, it may become difficult to alter the probabilities with new training data	Training can happen in real-time. Size of database continues to grow when presented with new training data.	Training is usually done offline. Can interpolate between known and unknown data, but requires a lot of varied data to accurately describe the model
Inferring relationships	Can be challenging to link multiple parameters together.	Able to handle multiple simultaneous controls Good generalization possible in finding a match	Able to handle multiple simultaneous controls Able to handle matching new and unseen input sequences
External control	Constraint satisfaction problems can be used to influence the transition matrix without altering the overall probabilities. Relationships between external controls and the model must be explicitly set.	Can simultaneously take all inputs to the system. Both internal and external.	Can simultaneously take all inputs to the system. Both internal and external.

	Probability	**Search**	**Regression**
Speed	Once the model is trained, and assuming the performance only uses previously seen data, then this system is fast	As model holds more data it becomes more accurate; however the search also becomes slower	Assuming the computer can handle solving all the functions in the model, this system is fast.
Context and Memory	Musical context is provided through sequences of source states. Using Variable Length Markov Models, these systems can ensure the longest possible sequence is used during performance.	Sampling at a regular rate creates a sequence of samples that have time implied by the order. Providing loner context requires using larger numbers of samples during the search.	Sequential Neural networks that low pass the inputs enable past events to influence the output of the system.

BIBLIOGRAPHY

Ames, Charles. 1987. "Automated Composition in Retrospect: 1956-1986." *Leonardo* 20 (2) (January 1): 169–185.

———. 1989. "The Markov Process as a Compositional Model: a Survey and Tutorial." *Leonardo*: 175–187.

Anders, Torsten. 2007. "Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System". Belfast: Queen's University.

Arfib, Daniel, Jean-Michel Couturier, and Loic Kessous. 2005. "Expressiveness and Digital Musical Instrument Design." *Journal of New Music Research* 34 (1): 125–136.

Bahn, Curtis, Tomie Hahn, and Dan Trueman. 2001. "Physicality and Feedback: a Focus on the Body in the Performance of Electronic Music." In *Proceedings of the International Computer Music Conference*, 44–51.

Bahn, Curtis, and Dan Trueman. 2001. "Interface: Electronic Chamber Ensemble." In *The Conference on New Interfaces for Musical Expression*, 1–5.

Banzi, Massimo. 2008. *Getting Started with Arduino*. O'Reilly Media / Make.

Barbosa, Alvaro. 2003. "Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation." *Leonardo Music Journal* 13: 53–59.

Barbosa, Alvaro, Jorge Cardoso, and Gunter Geiger. 2005. "Network Latency Adaptive Tempo in the Public Sound Objects System." In *The Conference on New Interfaces for Musical Expression*, 184–187.

Berdahl, Edgar, Hans-Christoph Steiner, and Colin Oldham. 2008. "Practical Hardware and Algorithms for Creating Haptic Musical Instruments." In *The Conference on New Interfaces for Musical Expression*. Genova, Italy.

Birnbaum, David, Rebecca Fiebrink, Joseph Malloch, and Marcelo M. Wanderley. 2005. "Towards a Dimension Space for Musical Devices." In *The Conference on New Interfaces for Musical Expression*, 192–195.

Bischoff, John, Rich Gold, and Jim Horton. 1978. "Music for an Interactive Network of Microcomputers." *Computer Music Journal* 2 (3): 24–29.

Bois, Mario. 1967. *Iannis Xenakis, the Man and His Music; a Conversation with the Composer and a Description of His Works*. London: Boosey & Hawkes Music Publishers.

Cáceres, Juan-Pablo, and Chris Chafe. 2010. "JackTrip: Under the Hood of an Engine for Network Audio." *Journal of New Music Research* 39 (3) (September): 183.

Cáceres, Juan-Pablo, Robert Hamilton, Deepak Iyer, Chris Chafe, and Ge Wang. 2008. "To the Edge with China: Explorations in Network Performance." In *The 4th International Conference on Digital Arts*.

Cage, John. 1991. "An Autobiographical Statement." *Southwest Review* 76 (1): 59.

Chafe, Chris, and Michael Gurevich. 2004. "Network Time Delay and Ensemble Accuracy: Effects of Latency, Asymmetry." In *The AES 117th Convention*, 6208.

Clayton, Martin. 2001. *Time in Indian Music : Rhythm, Metre, and Form in North Indian Rag Performance: Rhythm, Metre, and Form in North Indian Rag Performance*. Oxford University Press.

College, Minnesota Justin London Professor of Music Carleton. 2004. *Hearing in Time : Psychological Aspects of Musical Meter: Psychological Aspects of Musical Meter*. Oxford University Press.

Collins, Nick. 2003. "Generative Music and Laptop Performance." *Contemporary Music Review* 22 (4): 67–79.

———. 2006. "Towards Autonomous Agents for Live Computer Music: Realtime Machine Listening and Interactive Music Systems." Edited by Alan Blackwell. Cambridge, Music and Science.

———. 2010. "Contrary Motion: An Oppositional Interactive Music System." In *The Conference on New Interfaces for Musical Expression*. Sydney, Australia.

Collins, Nicolas. 1991. "Low Brass: The Evolution of Trombone-propelled Electronics." *Leonardo Music Journal: Journal of the International Society for the Arts, Sciences and Technology* 1 (1) (January 1): 41–44.

Cook, Perry R. 1992. "A Meta-wind-instrument Physical Model, and a Meta-controller for Real-time Performance Control." In *The International*

Computer Music Conference, 273–276. Ann Arbor, MI: MPublishing, University of Michigan Library.

———. 2001. "Principles for Designing Computer Music Controllers." In *The Conference on New Interfaces for Musical Expression*, 1–4.

———. 2009. "Re-Designing Principles for Computer Music Controllers: A Case Study of SqueezeVox Maggie." In *The Conference on New Interfaces for Musical Expression*, 303–307.

Cook, Perry R., and Colby N. Leider. 2000. "SqueezeVox: a New Controller for Vocal Synthesis Models." In *The International Computer Music Conference*.

Cope, David. 2005. *Computer Models of Musical Creativity*. The MIT Press.

Croft, John. 2007. "Theses on Liveness." *Organised Sound* 12 (1): 59–66.

D' Escriván, Julio. 2006. "To Sing the Body Electric: Instruments and Effort in the Performance of Electronic Music." *Contemporary Music Review* 25 (1-2): 183–191.

Dannenberg, Roger B. 1984. "An On-line Algorithm for Real-time Accompaniment." In *Proceedings of the 1984 International Computer Music Conference*, 193–198.

Dannenberg, Roger B., Belinda Thom, and David Watson. 1997. "A Machine Learning Approach to Musical Style Recognition."

Diakopoulos, Dimitri, and Ajay Kapur. 2010. "Argos: An Opensource Application for Building Multi-Touch Musical Interfaces." In *The International Computer Music Conference*.

Downie, J. Stephen, Donald Byrd, and Tim Crawford. 2009. "Ten Years of ISMIR: Reflections on Challenges and Opportunities." In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, 13–18.

Driessen, Peter F., Thomas E. Darcie, and Bipin Pillay. 2011. "The Effects of Network Delay on Tempo in Musical Performance." *Computer Music Journal* 35 (1) (March 1): 76–89.

Drummond, Jon. 2009. "Understanding Interactive Systems." *Organised Sound* 14 (02): 124–133.

Eigenfeldt, Arne. 2006. "Kinetic Engine: Toward an Intelligent Improvising Instrument." In *Proceedings of the Sound and Music Computing Conference*, 97–100.

Emmerson, Simon. 2000. "'Losing Touch?': The Human Performer and Electronics." *Music, Electronic Media and Culture*: 194–216.

Fiebrink, Rebecca. 2011. "Real-time Human Interaction with Supervised Learning Algorithms for Music Composition and Performance". Princeton, NJ, USA: Princeton University, Computer Science.

Fiebrink, Rebecca, Dan Trueman, Cameron Britt, Michelle Nagai, Konrad Kaczmarek, Michael Early, MR Daniel, Anne Hege, and Perry R. Cook. 2010. "Toward Understanding Human-computer Interaction in Composing the Instrument." In *The International Computer Music Conference*.

Freed, Adrian. 2008. "Application of New Fiber and Malleable Materials for Agile Development of Augmented Instruments and Controllers." In *The Conference on New Interfaces for Musical Expression*.

Gao, Mike, and Craig Hanson. 2009. "LUMI: Live Performance Paradigms Utilizing Software Integrated Touch Screen and Pressure Sensitive Button Matrix." In *The Conference on New Interfaces for Musical Expression*.

Gresham-Lancaster, Scot. 1998. "The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music." *Leonardo Music Journal* 8: 39–44.

Groot, Rokus de. 1997. "Ockeghem and New Music in the Twentieth Century." *Tijdschrift Van De Koninklijke Vereniging Voor Nederlandse Muziekgeschiedenis* 47 (1/2) (January 1): 201–220.

Hamanaka, Masatoshi, Masataka Goto, Hideki Asoh, and Nobuyuki Otsu. 2003. "A Learning-based Jam Session System That Imitates a Player's Personality Model." In *International Joint Conference on Artificial Intelligence*, 18:51–58.

Hochenbaum, Jordan, and Owen Vallis. 2009. "Bricktable: A Musical Tangible Multi-touch Interface." In *The Berlin Open Conference*. Berlin, Germany.

Jordà, Sergi, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. 2007. "The reacTable: Exploring the Synergy Between Live Music Performance

and Tabletop Tangible Interfaces." In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 139–146.

Jorda, Sergi, Martin Kaltenbrunner, Gunter Geiger, and Ross Bencina. 2005. "The Reactable*." In *The International Computer Music Conference*, 579–582.

Kaltenbrunner, Martin, and Ross Bencina. 2007. "reacTIVision: a Computer-vision Framework for Table-based Tangible Interaction." In *The 1st International Conference on Tangible and Embedded Interaction*, 69–74.

Kane, Brian. 2007. "Aesthetic Problems of Net Music." In . Spark Festival. University of Minnesota. internal-pdf://Kane_2007-1008294145/Kane_2007.pdf.

Kapur, Ajay. 2007. "Digitizing North Indian Music: Preservation and Extension Using Multimodal Sensor Systems, Machine Learning and Robotics."

Kapur, Ajay, Michael Darling, Dimitri Diakopoulos, Jim Murphy, Jordan Hochenbaum, Owen Vallis, and Curtis Bahn. 2011. "The Machine Orchestra: An Ensemble of Human Laptop Performers and Robotic Musical Instruments." *Computer Music Journal* 35 (4): 49–63.

Kapur, Ajay, Michael Darling, and Raahki Kapur. 2012. "Don't Forget the Machines: Orchestra of Humans, Laptops, and Robots." In *1st Symposium on Laptop Ensembles & Orchestras*, 80–81. Baton Rouge, Louisiana.

Kiefer, Chris, Nick Collins, and Geraldine Fitzpatrick. 2008. "HCI Methodology for Evaluating Musical Controllers: A Case Study." In *The Conference on New Interfaces for Musical Expression*.

Kockelkoren, Petran. 2003. *Technology: Art, Fairground and Theatre*. NAi Publishers.

Krefeld, Volker, and Michel Waisvisz. 1990. "The Hand in the Web: An Interview with Michel Waisvisz." *Computer Music Journal* 14 (2) (July 1): 28–33.

Lazzaro, John, and John Wawrzynek. 2001. "A Case for Network Musical Performance." In *The International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 157–166. NOSSDAV '01. New York, NY, USA: ACM.

Lerdahl, Fred, and Ray Jackendoff. 1996. *A Generative Theory of Tonal Music*. The MIT Press.

Levin, Golan. 1999. "Interface Metaphors and Signal Representation for Audiovisual Performance Systems". MIT. http://www.flong.com/texts/essays/thesis_proposal/.

Lewis, George E. 2000. "Too Many Notes: Computers, Complexity and Culture in Voyager." *Leonardo Music Journal* 10 (January 1): 33.

Malloch, Joseph, David Birnbaum, Elliot Sinyor, and Marcelo M. Wanderley. 2006. "Towards a New Conceptual Framework for Digital Musical Instruments." In *The 9th International Conference on Digital Audio Effects*, 49–52.

Martin, Aengus, Craig T. Jin, Alistair McEwan, and William L. Martens. 2011. "A Similarity Algorithm for Interactive Style Imitation." In *ICMC*. Huddersfield, UK.

Mathews, Max, and Andrew Schloss. 1989. "The Radio Drum as a Synthesizer Controller." In *The International Computer Music Conference*.

Murphy, Jim, Ajay Kapur, and Carl Burgin. 2010. "The Helio: A Study of Membrane Potentiometers and Long Force Sensing Resistors for Musical Interfaces." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 459–462.

Nierhaus, Gerhard. 2009. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer.

Nietzsche, Friedrich Wilhelm. 1896. *Thus Spoke Zarathustra: A Book for Everyone and No One*. New York and London: Macmillan.

Nishibori, Yu, and Toshio Iwai. 2006. "Tenori-on." In *The Conference on New Interfaces for Musical Expression*, 172–175. Paris, France: IRCAM.

Pachet, François. 2002. "The Continuator: Musical Interaction With Style." *Journal of New Music Research* 31 (1).

Pachet, François, and Pierre Roy. 2011. "Markov Constraints: Steerable Generation of Markov Sequences." *Constraints* 2 (16): 148–172.

Pachet, François, Pierre Roy, and Gabriele Barbieri. 2011. "Finite-Length Markov Processes with Constraints." In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 635–642. Barcelona, Spain.

Paradiso, Joe. 2004. "Wearable Wireless Sensing for Interactive Media." In *First International Workshop on Wearable and Implantable Body Sensor Networks*.

Paradiso, Joseph A. 1999. "The Brain Opera Technology: New Instruments and Gestural Sensors for Musical Interaction and Performance." *Journal of New Music Research* 28 (2) (June 1): 130.

Polansky, Larry. 1994. "Live Interactive Computer Music in HMSL, 1984-1992." *Computer Music Journal* 18 (2): 59–77.

Pressing, Jeff. 1990. "Cybernetic Issues in Interactive Performance Systems." *Computer Music Journal* 14 (1): 12.

Reynolds, Roger. 1979. "John Cage and Roger Reynolds: A Conversation." *The Musical Quarterly* LXV (4) (October): 573–594.

Rowe, Robert. 2001. *Machine Musicianship*. MIT Press.

Saneifar, Hassan, Sandra Bringay, Anne Laurent, and Maguelonne Teisseire. 2008. "S2mp: Similarity Measure for Sequential Patterns."

Schloss, Andrew. 2003. "Using Contemporary Technology in Live Performance: The Dilemma of the Performer." *Journal of New Music Research* 32 (3): 239–242.

Schnell, Norbert, and Marc Battier. 2002. "Introducing Composed Instruments, Technical and Musicological Implications." In *Proceedings of the 2002 Conference on New Interfaces for Musical Expression*, 1–5.

Sergi, Jordà Puig. 2005. "Digital Lutherie - Crafting Musical Computers for New Musics' Performance and Improvisation". Universitat Pompeu Fabra.

Smallwood, Scott, Dan Trueman, Perry R. Cook, and Ge Wang. 2008. "Composing for Laptop Orchestra." *Computer Music Journal* 32 (1) (April 1): 9–25.

Stobart, Henry, and Ian Cross. 2000. "The Andean Anacrusis? Rhythmic Structure and Perception in Easter Songs of Northern Potosí, Bolivia." *British Journal of Ethnomusicology* 9 (2): 63–92.

Temperley, David. 2004. *The Cognition of Basic Musical Structures*. The MIT Press.

Trueman, Dan, and Perry R. Cook. 2000. "BoSSA: The Deconstructed Violin Reconstructed." *Journal of New Music Research* 29 (2): 121–130.

Vallis, Owen, and Ajay Kapur. 2011. "Community-Based Design: The Democratization of Musical Interface Construction." *Leonardo Music Journal* 21.

Van Nort, Doug. 2009. "Instrumental Listening: Sonic Gesture as Design Principle." *Organised Sound* 14 (2): 177–187.

Vercoe, Barry. 1984. "The Synthetic Performer in the Context of Live Performance." In *Proceedings of the International Computer Music Conference*, 189–191.

Vercoe, Barry, and Miller S. Puckette. 1985. "Synthetic Rehearsal: Training the Synthetic Performer." In *Proceedings of ICMC*, 275–278.

Wanderley, Marcelo M., and Nicola Orio. 2002. "Evaluation of Input Devices for Musical Expression: Borrowing Tools from Hci." *Computer Music Journal* 26 (3): 62–76.

Wang, Ge, and Perry R. Cook. 2003. "ChucK: A Concurrent, On-the-fly Audio Programming Language." In *The International Computer Music Conference*, 219–226.

Weinberg, Gil. 2002. "The Aesthetics, History, and Future Challenges of Interconnected Music Networks." In *The International Computer Music Conference*, 349–356.

———. 2005. "Interconnected Musical Networks: Toward a Theoretical Framework." *Computer Music Journal* 29 (2) (June 1): 23–39.

Weinberg, Gil, Scott Driscoll, and Mitchell Parry. 2005. "Musical Interactions with a Perceptual Robotic Percussionist." In *Robot and Human Interactive Communication, 2005. ROMAN 2005. IEEE International Workshop On*, 456–461.

Wiley, Meason, and Ajay Kapur. 2009. "Multi-Laser Gestural Interface—Solutions for Cost-Effective and Open Source Controllers." In *The Conference on New Interfaces for Musical Expression*.

Wright, Matthew, and David Wessel. 1998. "An Improvisation Environment for Generating Rhythmic Structures Based on North Indian 'Tal' Patterns."

Xenakis, Iannis. 1971. *Formalized Music*. Bloomington: Indiana University Press (IN) Bloomington, IN.